

XSL Formatting Objects

The second half of the Extensible Style Language (XSL) is the formatting language. This is an XML application used to describe how content should be rendered when presented to a reader. Generally, a style sheet uses the XSL transformation language to transform an XML document into a new XML document that uses the XSL formatting objects vocabulary. While many hope that Web browsers will one day know how to directly display data marked up with XSL formatting objects, for now an additional step is necessary in which the output document is further transformed into some other format such as PDF.

Overview of the XSL Formatting Language

XSL formatting objects provide a more sophisticated visual layout model than HTML+CSS (even CSS2). Formatting supported by XSL formatting objects but not supported by HTML+CSS includes non-Western layout, footnotes, margin notes, page numbers in cross references, and more. In particular, while CSS is primarily intended for use on the Web, XSL formatting objects are designed for more general use. You should, for instance, be able to write an XSL style sheet that uses formatting objects to lay out an entire printed book. A different style sheet should be able to transform the same XML document into a Web site.



In This Chapter

Understanding the XSL formatting language

Formatting objects and their properties

Formatting and styling pages

Inserting rules in text

Embedding graphics in a rendered document

Linking to URI targets

Inserting lists in text

Replacing characters

Using sequences

Footnotes

Floats

Understanding how to use the XSL formatting properties



A Word of Caution about the XSL Formatting Language

XSL is still under development. The XSL language has changed radically in the past, and will change again in the future. This chapter is based on the April 21, 1999 (fourth) draft of the XSL specification. By the time you are reading this book, this draft of XSL will probably have been superseded and the exact syntax of XSL will have changed. The formatting objects part of the specification is, if anything, even less complete than the transformation language specification. If you do encounter something that doesn't seem to work quite right, you should compare the examples in this book against the most current specification.

To make matters worse, no software implements all of the April 21, 1999 draft of the XSL specification, even just the formatting objects half. In fact, so far there's exactly one partial implementation of XSL formatting objects, James Tauber's FOP, which converts XML documents using the XSL formatting objects into PDF. There are no Web browsers that can display a document written with XSL formatting objects.

Eventually, of course, this should be straightened out as the standard evolves toward its final incarnation and more vendors implement XSL formatting objects. Until then, you're faced with a choice: You can either work out on the bleeding edge with XSL in its current, incomplete, unfinished state and try to work around all the bugs and omissions you'll encounter, or stick with a more established technology, such as CSS, until XSL is more solid.

Formatting Objects and Their Properties

There are exactly 51 XSL formatting object elements. Of the 51 elements, most signify various kinds of rectangular areas. Most of the rest are containers for rectangular areas and spaces. In alphabetical order, these formatting objects are:

- | | |
|---|--|
| ♦ <code>bidirectional-override</code> | ♦ <code>inline-graphic</code> |
| ♦ <code>block</code> | ♦ <code>inline-included-container</code> |
| ♦ <code>character</code> | ♦ <code>inline-rule</code> |
| ♦ <code>display-graphic</code> | ♦ <code>inline-sequence</code> |
| ♦ <code>display-included-container</code> | ♦ <code>layout-master-set</code> |
| ♦ <code>display-rule</code> | ♦ <code>list-block</code> |
| ♦ <code>display-sequence</code> | ♦ <code>list-item</code> |
| ♦ <code>first-line-marker</code> | ♦ <code>list-item-body</code> |
| ♦ <code>float</code> | ♦ <code>list-item-label</code> |
| ♦ <code>flow</code> | ♦ <code>multi-case</code> |
| ♦ <code>footnote</code> | ♦ <code>multi-properties</code> |
| ♦ <code>footnote-citation</code> | ♦ <code>multi-property-set</code> |

- ♦ multi-switch
- ♦ multi-toggle
- ♦ page-number
- ♦ page-number-citation
- ♦ page-sequence
- ♦ region-after
- ♦ region-before
- ♦ region-body
- ♦ region-end
- ♦ region-start
- ♦ root
- ♦ sequence-specification
- ♦ sequence-specifier-alternating
- ♦ sequence-specifier-repeating
- ♦ sequence-specifier-single
- ♦ simple-link
- ♦ simple-page-master
- ♦ static-content
- ♦ table
- ♦ table-and-caption
- ♦ table-body
- ♦ table-caption
- ♦ table-cell
- ♦ table-column
- ♦ table-footer
- ♦ table-header
- ♦ table-row

The XSL formatting model is based on rectangular boxes called *areas* that can contain text, empty space, or other formatting objects. As with CSS boxes, each area has borders and padding on each of its sides, although CSS margins are replaced by XSL indents. An XSL formatter reads the formatting objects to determine which areas to place where on the page. Many formatting objects produce single areas (at least most of the time), but due to page breaks, word wrapping, hyphenation, and other aspects of fitting a potentially infinite amount of text into a finite area, some formatting objects do occasionally generate more than one area.

Note

A box that contains space is not the same as a box that contains whitespace characters. A box containing empty space refers to a physical blank area on the page or screen, for example the margins on the left and right sides of this page. This is not the same as the space characters between the words on this page.

The formatting objects differ primarily in what they contain. For example, the `list-item-label` formatting object is a box that contains a bullet, a number, or another indicator placed in front of a list item. A `list-item-body` formatting object is a box that contains the text, sans label, of the list item. And a `list-item` formatting object is a box that contains both the `list-item-label` and `list-item` formatting objects.

The formatting objects are further divided into four different kinds of rectangular areas:

1. area containers
2. block areas
3. line areas
4. inline-areas

These form a rough hierarchy. Area containers contain other smaller area containers and block areas. Block areas contain other block areas, line areas, and content. Line areas contain inline areas. Inline areas contain other inline areas and content. More specifically:

- ♦ An area container is the highest-level container in XSL. It can be positioned at precise coordinates inside the area that contains it. It can contain either other, smaller area containers or a sequence of block areas and display spaces. You can think of a page of this book as an area container that contains five other area containers: the header, the main body of the page, the footer, and the left and right margins. (In this example, the margin areas contain no content.) Formatting objects that produce area containers include `region-body`, `region-before`, `region-after`, `region-start`, and `region-end`.
- ♦ A block area represents a block-level element such as a paragraph or a list item. Although block areas may contain other block areas, there should always be a line break before the start and after the end of each block area. A block area, rather than being precisely positioned by coordinates, is placed sequentially in the area that contains it. As other block areas are added and deleted before it or within it, the block area's position shifts as necessary to make room. A block area may contain line areas, display spaces, and other block areas that are sequentially arranged in the containing block area. A block area also may contain a single graphic image. Formatting objects that produce block areas include `block`, `display-graphic`, `display-link`, `display-rule`, and `list-block`.
- ♦ A line area represents a line of text inside a block. For example, each separate line in this list item is a line area. Line areas can contain inline areas and inline spaces. There are no formatting objects that correspond to line areas. Instead, the formatting engine calculates the line areas as it decides how to wrap lines inside block areas.
- ♦ Inline areas are parts of a line such as a single character, a footnote reference, or a mathematical equation. Inline areas can contain other inline areas and inline spaces. Formatting objects that produce inline areas include `character`, `inline-graphic`, `inline-link`, `inline-rule`, `inline-sequence` and `page-number`.

The fo Namespace

XML elements for XSL formatting objects are placed in the `http://www.w3.org/XSL/Format/1.0` namespace with this declaration in an XSL stylesheet:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns:fo="http://www.w3.org/XSL/Format/1.0"
  result-ns="fo">
```

About 99 times out of 100, the chosen prefix is `fo`. Consequently, you almost always see the following elements with the `fo` prefix in this form:

- | | |
|--|--|
| ♦ <code>fo:bidi-override</code> | ♦ <code>fo:multi-switch</code> |
| ♦ <code>fo:block</code> | ♦ <code>fo:multi-toggle</code> |
| ♦ <code>fo:character</code> | ♦ <code>fo:page-number</code> |
| ♦ <code>fo:display-graphic</code> | ♦ <code>fo:page-number-citation</code> |
| ♦ <code>fo:display-included-container</code> | ♦ <code>fo:page-sequence</code> |
| ♦ <code>fo:display-rule</code> | ♦ <code>fo:region-after</code> |
| ♦ <code>fo:display-sequence</code> | ♦ <code>fo:region-before</code> |
| ♦ <code>fo:first-line-marker</code> | ♦ <code>fo:region-body</code> |
| ♦ <code>fo:float</code> | ♦ <code>fo:region-end</code> |
| ♦ <code>fo:flow</code> | ♦ <code>fo:region-start</code> |
| ♦ <code>fo:footnote</code> | ♦ <code>fo:root</code> |
| ♦ <code>fo:footnote-citation</code> | ♦ <code>fo:sequence-specification</code> |
| ♦ <code>fo:inline-graphic</code> | ♦ <code>fo:sequence-specifier-alternating</code> |
| ♦ <code>fo:inline-included-container</code> | ♦ <code>fo:sequence-specifier-repeating</code> |
| ♦ <code>fo:inline-rule</code> | ♦ <code>fo:sequence-specifier-single</code> |
| ♦ <code>fo:inline-sequence</code> | |
| ♦ <code>fo:layout-master-set</code> | ♦ <code>fo:simple-link</code> |
| ♦ <code>fo:list-block</code> | ♦ <code>fo:simple-page-master</code> |
| ♦ <code>fo:list-item</code> | ♦ <code>fo:static-content</code> |
| ♦ <code>fo:list-item-body</code> | ♦ <code>fo:table</code> |
| ♦ <code>fo:list-item-label</code> | ♦ <code>fo:table-and-caption</code> |
| ♦ <code>fo:multi-case</code> | ♦ <code>fo:table-body</code> |
| ♦ <code>fo:multi-properties</code> | ♦ <code>fo:table-caption</code> |
| ♦ <code>fo:multi-property-set</code> | |

- ♦ `fo:table-cell`
- ♦ `fo:table-column`
- ♦ `fo:table-footer`
- ♦ `fo:table-header`
- ♦ `fo:table-row`

In this chapter, I will use the `fo` prefix without further comment.



Namespaces are discussed in Chapter 18, *Namespaces*. Until then, all you have to know is that the names of all XSL formatting object elements begin with `fo:`.

Formatting Properties

When taken as a whole, the various formatting objects in an XSL document specify the order in which content is to be placed on pages. However, all the details of formatting including but not limited to page size, element size, font, color, and a lot more are specified by XSL properties. These formatting properties are represented as attributes on the individual formatting object elements.

The details of many of these properties should be familiar from CSS. Work is ongoing to ensure that CSS and XSL use the same names to mean the same things. For example, the CSS property `font-family` means the same thing as the XSL `font-family` property; and although the syntax for assigning values to properties is different in CSS and XSL, the syntax of the values themselves is exactly the same. To indicate that the `fo:block` element is formatted in some approximation of Times, you might use this CSS rule:

```
fo:block {font-family: New York, Times New Roman, Times, serif}
```

The XSL equivalent is to include a `font-family` attribute in the `fo:block` start tag in this way:

```
<fo:block  
  font-family="New York, Times New Roman, Times, serif">
```

Although this is superficially different, the style name (`font-family`) and the style value (`New York, Times New Roman, Times, serif`) are exactly the same. CSS's `font-family` property is specified as a list of font names, separated by commas, and in order from first choice to last choice. XSL's `font-family` property is specified as a list of font names, separated by commas, and in order from first choice to last choice. Both CSS and XSL understand the keyword `serif` to mean an arbitrary serif font.

Note

As of the fourth draft of the XSL draft specification on which this chapter is based, complete synchronization between equivalent CSS and XSL properties isn't quite finished. This should be cleaned up in the next draft.

Of course, XSL formatting objects support many properties that have no CSS equivalent, such as `font-size-adjust`, `ligature`, `character`, and `hyphenation-keep`. You need to learn these to take full advantage of XSL. The standard XSL properties follow:

- ♦ `auto-restore`
- ♦ `azimuth`
- ♦ `background`
- ♦ `background-attachment`
- ♦ `background-color`
- ♦ `background-image`
- ♦ `background-position`
- ♦ `background-repeat`
- ♦ `border`
- ♦ `border-after-color`
- ♦ `border-after-style`
- ♦ `border-after-width`
- ♦ `border-before-color`
- ♦ `border-before-style`
- ♦ `border-before-width`
- ♦ `border-bottom`
- ♦ `border-bottom-color`
- ♦ `border-bottom-style`
- ♦ `border-bottom-width`
- ♦ `border-collapse`
- ♦ `border-color`
- ♦ `border-end-color`
- ♦ `border-end-style`
- ♦ `border-end-width`
- ♦ `border-left`
- ♦ `border-left-color`
- ♦ `border-left-style`
- ♦ `border-left-width`
- ♦ `border-right`
- ♦ `border-right-color`
- ♦ `border-right-style`
- ♦ `border-right-width`
- ♦ `border-spacing`
- ♦ `border-start-color`
- ♦ `border-start-style`
- ♦ `border-start-width`
- ♦ `border-style`
- ♦ `border-top`
- ♦ `border-top-color`
- ♦ `border-top-style`
- ♦ `border-top-width`
- ♦ `border-width`
- ♦ `bottom`
- ♦ `break-after`
- ♦ `break-before`
- ♦ `caption-side`
- ♦
- ♦ `cell-height`
- ♦ `character`
- ♦ `clear`
- ♦ `clip`
- ♦ `color`
- ♦ `column-count`
- ♦ `column-gap`
- ♦ `column-number`
- ♦ `column-width`
- ♦ `country`
- ♦ `cue`
- ♦ `cue-after`
- ♦ `cue-before`

- ♦ digit-group-sep
- ♦ direction
- ♦ elevation
- ♦ empty-cells
- ♦ end-indent
- ♦ ends-row
- ♦ extent
- ♦ external-destination
- ♦ float
- ♦ flow-name
- ♦ font
- ♦ font-family
- ♦ font-height-override-after
- ♦ font-height-override-before
- ♦ font-size
- ♦ font-size-adjust
- ♦ font-stretch
- ♦ font-style
- ♦ font-variant
- ♦ font-weight
- ♦ format
- ♦ height
- ♦ href
- ♦ hyphenate
- ♦ hyphenation-char
- ♦ hyphenation-keep
- ♦ hyphenation-ladder-count
- ♦ hyphenation-push-char-count
- ♦ hyphenation-remain-char-count
- ♦ id
- ♦ indicate-destination
- ♦ inhibit-line-breaks
- ♦ initial
- ♦ initial-page-number
- ♦ internal-destination
- ♦ keep-with-next
- ♦ keep-with-previous
- ♦ language
- ♦ last-line-end-indent
- ♦ left
- ♦ length
- ♦ letter-spacing
- ♦ letter-value
- ♦ line-height
- ♦ line-height-shift-adjustment
- ♦ line-stacking-strategy
- ♦ margin
- ♦ margin-bottom
- ♦ margin-left
- ♦ margin-right
- ♦ margin-top
- ♦ max-height
- ♦ max-width
- ♦ may-break-after-row
- ♦ may-break-before-row
- ♦ min-height
- ♦ min-width
- ♦ name
- ♦ n-columns-repeated
- ♦ n-columns-spanned

- ♦ n-digits-per-group
- ♦ n-rows-spanned
- ♦ orphans
- ♦ overflow
- ♦ padding
- ♦ padding-after
- ♦ padding-before
- ♦ padding-bottom
- ♦ padding-end
- ♦ padding-left
- ♦ padding-right
- ♦ padding-start
- ♦ padding-top
- ♦ page-break-inside
- ♦ page-height
- ♦ page-master-blank-even
- ♦ page-master-even
- ♦ page-master-first
- ♦ page-master-last-even
- ♦ page-master-last-odd
- ♦ page-master-name
- ♦ page-master-odd
- ♦ page-master-repeating
- ♦ page-width
- ♦ pause
- ♦ pause-after
- ♦ pause-before
- ♦ pitch
- ♦ pitch-range
- ♦ play-during
- ♦ position
- ♦ precedence
- ♦ provisional-distance-between-starts
- ♦ provisional-label-separation
- ♦ reference-orientation
- ♦ ref-id
- ♦ richness
- ♦ right
- ♦ row-height
- ♦ rule-orientation
- ♦ rule-style
- ♦ rule-thickness
- ♦ scale
- ♦ score-spaces
- ♦ script
- ♦ sequence-src
- ♦ show-destination
- ♦ size
- ♦ space-above-destination-block
- ♦ space-above-destination-start
- ♦ space-after
- ♦ space-before
- ♦ space-between-list-rows
- ♦ space-end
- ♦ space-start
- ♦ span
- ♦ speak
- ♦ speak-header
- ♦ speak-numeral
- ♦ speak-punctuation
- ♦ speech-rate

- ♦ start-indent
- ♦ starts-row
- ♦ state
- ♦ stress
- ♦ switch-to
- ♦ table-height
- ♦ table-layout
- ♦ table-omit-middle-footer
- ♦ table-omit-middle-header
- ♦ table-width
- ♦ text-align
- ♦ text-align-last
- ♦ text-decoration
- ♦ text-indent
- ♦ text-shadow
- ♦ text-transform
- ♦ title
- ♦ top
- ♦ vertical-align
- ♦ visibility
- ♦ voice-family
- ♦ volume
- ♦ white-space-treatment
- ♦ widows
- ♦ width
- ♦ word-spacing
- ♦ wrap-option
- ♦ writing-mode
- ♦ z-index

Transforming to Formatting Objects

XSL formatting objects are a complete XML vocabulary used to arrange elements on a page. A document that uses XSL formatting objects is simply a well-formed XML document that uses this vocabulary. That means it has an XML declaration, a root element, child elements, and so forth. It must adhere to all the well-formedness rules of any XML document, or formatters will not accept it. By convention, a file that contains XSL formatting objects has the three-letter suffix `.fob`. However, it might have the suffix `.xml` because it also is a well-formed XML file.

Listing 15-1 is a simple document marked up using XSL formatting objects. The root of the document is `fo:root`. This element contains a `fo:layout-master-set` and a `fo:page-sequence`. The `fo:layout-master-set` element contains `fo:simple-page-master` child elements. Each `fo:simple-page-master` describes a kind of page on which content will be placed. Here there's only one very simple page, but more complex documents can have different master pages for first, right, and left, body pages, front matter, back matter, and more; each with a potentially different set of margins, page numbering, and other features.

Content is placed on copies of the master page using a `fo:page-sequence`. The `fo:page-sequence` contains a `fo:sequence-specification` specifying the order in which the different master pages should be used. Next, it contains a `fo:flow` child that holds the actual content to be placed on the master pages in the specified sequence. The content here is given as two `fo:block` children each have a `font-size` property of 20 points and a `font-family` property of `serif`.

Listing 15-1: A simple document using the XSL formatting object vocabulary

```
<fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">

  <fo:layout-master-set>
    <fo:simple-page-master page-master-name="only">
      <fo:region-body/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence>

    <fo:sequence-specification>
      <fo:sequence-specifier-single page-master-name="only"/>
    </fo:sequence-specification>

    <fo:flow>
      <fo:block font-size="20pt" font-family="serif">
        Hydrogen
      </fo:block>
      <fo:block font-size="20pt" font-family="serif">
        Helium
      </fo:block>
    </fo:flow>

  </fo:page-sequence>

</fo:root>
```

Although you could write a document such as the one in Listing 15-1 by hand, that would lose all the benefits of content-format independence achieved by XML. Normally you write an XSL style sheet that uses the XSL transformation vocabulary to transform the source document into the formatting object vocabulary. Listing 15-2 is the XSL style sheet that produced Listing 15-1 by transforming the previous chapter's Listing 14-1.

Listing 15-2: A transformation from a source vocabulary to XSL formatting objects

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
  xmlns:fo="http://www.w3.org/XSL/Format/1.0"
  result-ns="fo" indent-result="yes">

  <xsl:template match="/">
    <fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">

      <fo:layout-master-set>
        <fo:simple-page-master page-master-name="only">
          <fo:region-body/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <fo:page-sequence>

        <fo:sequence-specification>
          <fo:sequence-specifier-single
            page-master-name="only"/>
        </fo:sequence-specification>

        <fo:flow>

          <xsl:apply-templates select="//ATOM"/>
        </fo:flow>

      </fo:page-sequence>

    </fo:root>
  </xsl:template>

  <xsl:template match="ATOM">
    <fo:block font-size="20pt" font-family="serif">
      <xsl:value-of select="NAME"/>
    </fo:block>
  </xsl:template>

</xsl:stylesheet>
```

Using FOP

At the time of this writing, no browser can directly display XML documents transformed into XSL formatting objects. There is only one piece of software that can work with a file marked up with XSL formatting objects, James Tauber's FOP. FOP is a free Java program that converts FO (formatting object) documents to

Adobe Acrobat PDF files. You can download the latest version of FOP at <http://www.jtauber.com/fop/>.

At the time of this writing, the available version of FOP is 0.6.0, which incompletely supports a subset of the formatting objects and properties in the fourth draft of XSL. FOP is a Java program that should run on any platform with a reasonably compatible Java 1.1 virtual machine. To install it, just place the `fop.jar` archive in your CLASSPATH. The `com.jtauber.fop.FOP` class contains the `main()` method for this program. Run it from the command line with arguments specifying the input and output files. For example:

```
C:\XML\BIBLE\15>java com.jtauber.fop.FOP 15-1.fob 15-1.pdf
James Tauber's FOP 0.6.0
auto page-height: using 11in
auto page-width: using 8in
successfully read and parsed 15-1.fob
laying out page 1...
done page 1.
successfully wrote 15-1.pdf
```

Here `15-1.fob` is the input XML file that uses the formatting object vocabulary. `15-1.pdf` is the output PDF file that can be displayed and printed by Adobe Acrobat or other programs that read PDF files.

Although PDF files are themselves ASCII text, this isn't a book about PostScript, so there's nothing to be gained by showing you the exact output of the above command. If you're curious, open the PDF file in any text editor. Instead, Figure 15-1 shows the rendered file displayed in Netscape Navigator using the Acrobat plug-in.

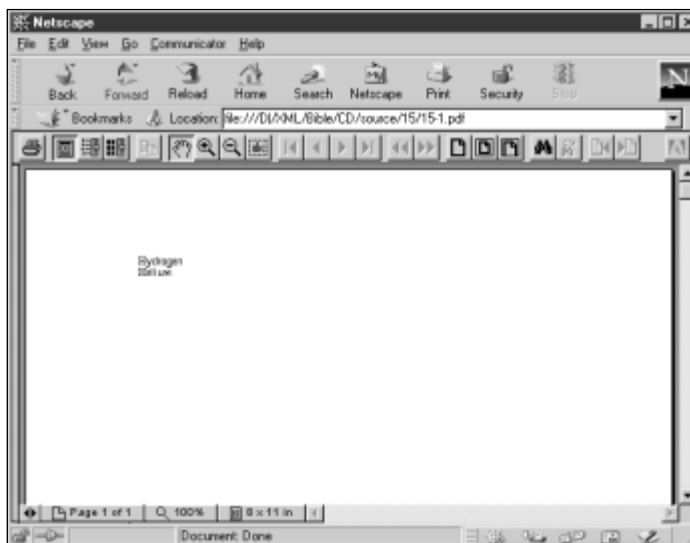


Figure 15-1: The PDF file displayed in Netscape Navigator

PDF files are not the only or even the primary eventual destination format for XML documents styled with XSL formatting objects. Certainly, one would hope that Web browsers will directly support XSL formatting objects in the not-too-distant future. For now, PDF files are the only available format, so that's what I show in this chapter. Eventually there should be more software that can read and display these files.

Page Layout

The root element of a formatting objects file is `fo:root`. This element contains one `fo:layout-master-set` element and zero or more `fo:page-sequence` elements. The `fo:root` element generally has an `xmlns:fo` attribute with the value `http://www.w3.org/XSL/Format/1.0` and may (though it generally does not) have an `id` attribute. The `fo:root` element exists just to declare the namespace and be the document root. It has no direct affect on page layout or formatting.

Master Pages

The `fo:layout-master-set` element is a container for all the different master pages used by the document. Simple page masters are similar in purpose to Quark XPress master pages or PowerPoint slide masters. Each defines a general layout for a page including its margins, the sizes of the header, footer, body area of the page, and so forth. Each actual page in the rendered document is based on one master page, and inherits certain properties like margins, page numbering, and layout from that master page.

Simple Page Masters

Each master page is represented by a `fo:simple-page-master` element. A `fo:layout-master-set` may contain one or more of these. A `fo:simple-page-master` element defines the layout of a page including the size of its before region, body region, after region, end region, and start region. Figure 15-2 shows the typical layout of these parts. The body is everything in the middle that's left over.

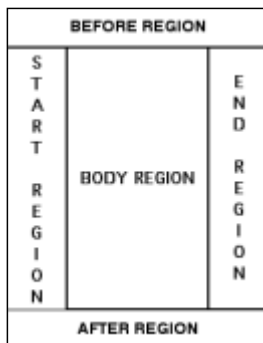


Figure 15-2: The layout of the parts of a simple page of English text

Note

In normal English text, the end region is the right side of the page and the start region is the left side of the page. This is reversed in Hebrew or Arabic text, because these languages read from right to left. In almost all modern languages, the before region is the header and the after region is the footer, but this could be reversed in a language that wrote from bottom to top.

The designer sets the size of the body (center) region, header, footer, end region, and start region as well as the distances between them using the appropriate region child elements. These are:

- ♦ `fo:region-before`
- ♦ `fo:region-after`
- ♦ `fo:region-body`
- ♦ `fo:region-start`
- ♦ `fo:region-end`

Each of the five regions of a simple page master may be filled with content from a `fo:flow` or `fo:static-content` element.

The `simple-page-master` element generally has three main attributes:

1. `page-master-name`: the name of this page master that page sequences will use to select the master page a particular page will be based on
2. `page-height`: the height of the page
3. `page-width`: the width of the page

The `page-height` and `page-width` can be subsumed into a single shorthand size property. If they are not provided, then the formatter chooses a reasonable default based on the media being used (e.g. 8.5" by 11").

For example, here is a `fo:layout-master-set` containing two `fo:simple-page-master` elements, one for even (left) pages and one for odd (right) pages. Both specify an 8.5-by-11-inch page size. Both have top and bottom margins of 0.5 inches. Each has an inner margin of 0.5 inches and an outer margin of 1 inch, as is common for facing pages.

```
<fo:layout-master-set>
  <fo:simple-page-master page-master-name="even"
    height="8.5in"        width="11in"
    margin-top="0.5in"    margin-bottom="0.5in"
    margin-left="1.0in"   margin-right="0.5in">
    <fo:region-body/>
  </fo:simple-page-master>
```

```

<fo:simple-page-master page-master-name="odd"
    height="8.5in"      width="11in"
    margin-top="0.5in"  margin-bottom="0.5in"
    margin-left="0.5in" margin-right="1.0in">
  <fo:region-body/>
</fo:simple-page-master>
</fo:layout-master-set>

```

Other attributes commonly applied to page masters include:

- ♦ **Attributes that affect the margins of the page:** `margin-bottom`, `margin-left`, `margin-right`, `margin-top`, `margin`
- ♦ **Attributes that affect the direction of the writing on the page:** `writing-mode`, `reference-orientation`

Region Properties

The five regions (`before`, `after`, `body`, `start`, `end`) share the same basic properties. These include:

- ♦ **Attributes that determine how content that overflows the borders of the region is handled:** `clip`, `overflow`
- ♦ **Attribute that determine how the content is wrapped in columns:** `column-count`, which is the number of columns in the region, and `column-gap`, which is the distance between columns
- ♦ **Attributes that affect the background of the region:** `background`, `background-attachment`, `background-color`, `background-image`, `background-repeat`, `background-position`
- ♦ **Attributes that affect the border of the region:** `border-before-color`, `border-before-style`, `border-before-width`, `border-after-color`, `border-after-style`, `border-after-width`, `border-start-color`, `border-start-style`, `border-start-width`, `border-end-color`, `border-end-style`, `border-end-width`, `border-top-color`, `border-top-style`, `border-top-width`, `border-bottom-color`, `border-bottom-style`, `border-bottom-width`, `border-left-color`, `border-left-style`, `border-left-width`, `border-right-color`, `border-right-style`, `border-right-width`, `border`, `border-top`, `border-bottom`, `border-left`, `border-right`, `border-color`, `border-style`, `border-width`
- ♦ **Attributes that affect the padding of the region:** `padding-bottom`, `padding-left`, `padding-right`, `padding-top`, `padding-bottom`, `padding-start`, `padding-end`, `padding-before`, `padding-after`, `padding`
- ♦ **Attributes that affect the margins of the region:** `margin-bottom`, `margin-left`, `margin-right`, `margin-top`, `margin`, `margin`, `space-before`, `space-after`, `start-indent`, `end-indent`

- ♦ Attributes that affect the direction of the writing in the region: `writing-mode`, `reference-orientation`

Most of these properties should be familiar from the CSS properties of the same name. Reasonable defaults are picked for all these values if they're not explicitly set. By adjusting them, you affect the overall layout of the page.

Additionally, the four outer regions (before, after, start, and end but not body) have an `extent` property that determines the size of the region. The size of the body is determined by whatever's left over in the middle after the other four regions are accounted for.

For example, here is a `fo:layout-master-set` that makes all outer regions one inch. Each region is given a two-pixel black border. Furthermore, the page itself has a half-inch margin on all sides.

```
<fo:layout-master-set>
  <fo:simple-page-master page-master-name="only"
    height="8.5in" width="11in"
    margin-top="0.5in" margin-bottom="0.5in"
    margin-left="1.0in" margin-right="0.5in">
    <fo:region-start extent="1.0in"
      border-color="black" border-width="2px"/>
    <fo:region-before extent="1.0in"
      border-color="black" border-width="2px"/>
    <fo:region-body
      border-color="black" border-width="2px"/>
    <fo:region-end extent="1.0in"
      border-color="black" border-width="2px"/>
    <fo:region-after extent="1.0in"
      border-color="black" border-width="2px"/>
  </fo:simple-page-master>
</fo:layout-master-set>
```

The body pages based on this page master will be 5.5 inches wide and 8 inches high. That's calculated by subtracting the size of everything else on the page from the size of the page.

Page Sequences

As well as a `fo:layout-master-set`, each formatting object document will generally contain one or more `fo:page-sequence` elements. Each page sequence contains three things in the following order:

- ♦ One `fo:sequence-specification` element defining the order in which the master pages are used

- ♦ Zero or more `fo:static-content` elements containing text to be placed on every page
- ♦ One `fo:flow` element containing data to be placed on each page in turn

The main difference between a `fo:flow` and a `fo:static-content` is that text from the flow isn't placed on more than one page, whereas the static content is. For example, the lines you're reading now are flow content that only appear on this page, whereas the part and chapter titles at the top of the page are static content that is repeated from page to page.

The `fo:sequence-specification` provides a list of the master pages for this sequence. Each page in the sequence has an associated page master that defines how the page will look. Listing 15-1 only used a single master page, but it is not uncommon to have more; for instance, one for the first page of a chapter, one for all the subsequent left-hand pages, and one for all the subsequent right-hand pages. For instance, there might be one simple page master for a table of contents, another for body text, and a third for the index. In this case, there is one page sequence each for the table of contents, the body text, and the index.

The `fo:flow` element contains, in order, the elements to be placed on the page. As each page fills up with elements from the flow, a new page is created with the next master layout in the sequence specification for the elements that remain in the flow.

The `fo:static-content` element contains information to be placed on each page. For instance, it may place the title of the book in the header of each page. Static content can be adjusted depending on the master page. For instance, the part title may be placed on left-hand pages, and the chapter title on right-hand pages. The `fo:static-content` element can also be used for items like page numbers that have to be calculated from page to page when the same calculation is repeated. In other words, what is static is not the text, but the calculation that produces the text.

Sequence Specifications

The `fo:sequence-specification` element lists the order in which particular master pages will be instantiated using one or more of these three child elements:

- ♦ `fo:sequence-specifier-single`
- ♦ `fo:sequence-specifier-alternating`
- ♦ `fo:sequence-specifier-repeating`

Each of these child elements has attributes that determine which master pages are used when. The simplest is `fo:sequence-specifier-single` whose `page-master-name` attribute identifies the master page to be instantiated. For example, this `fo:sequence-specification` element says that all content must be placed on a single instance of the master page named `letter`:

```
<fo:sequence-specification>
  <fo:sequence-specifier-single page-master-name="letter"/>
</fo:sequence-specification>
```

If there's more content than will fit on a single page, then the extra content is either truncated or scrolled, depending on the values of the `clip` and `overflow` attributes of the various regions where the content is placed. However, no more than one page will be created. Now consider this sequence specification:

```
<fo:sequence-specification>
  <fo:sequence-specifier-single page-master-name="letter"/>
  <fo:sequence-specifier-single page-master-name="letter"/>
</fo:sequence-specification>
```

This provides for up to pages, each based on the letter page master. If the first page fills up, a second will be created. If that page fills up, then content will be truncated or scrolled.

The same technique can be used to apply different master pages. For example, this sequence specification bases the first page on the master page named `letter1` and the second on the master page named `letter2`:

```
<fo:sequence-specification>
  <fo:sequence-specifier-single page-master-name="letter1"/>
  <fo:sequence-specifier-single page-master-name="letter2"/>
</fo:sequence-specification>
```

Of course, most of the time you don't know in advance exactly how many pages there will be. The `fo:sequence-specifier-alternating` and `fo:sequence-specifier-repeating` elements let you specify that as many pages as necessary will be used to hold the content. The `fo:sequence-specifier-repeating` element specifies one master page for the first page and a second master page for all subsequent pages. The `fo:sequence-specifier-alternating` element specifies up to six different master pages for the first page, even pages with content, odd pages with content, blank even pages, last even pages, and last odd pages.

For example, this sequence specifier says that the first page output should use the master page named `letter` first, but that all subsequent pages should use the master page named `letter`:

```
<fo:sequence-specification>
  <fo:sequence-specifier-repeating
    page-master-first="letter_first"
    page-master-repeating="letter"
  />
</fo:sequence-specification>
```

If the total content overflows the first page, it will be placed on a second page. If it overflows the second page, a third page will be created. As many pages as needed to hold all the content will be constructed.



Tip

At the time of this writing, it has not yet been decided whether or not `page-master-first` and `page-master-repeating` are both required. However, if you only have a single master page, you can certainly reuse it as the value for both `page-master-first` and `page-master-repeating` like this:

```
<fo:sequence-specification>
  <fo:sequence-specifier-repeating
    page-master-first="letter"
    page-master-repeating="letter"
  />
</fo:sequence-specification>
```

The `fo:sequence-specifier-alternating` element is designed more for a chapter of a printed book in which the first and last pages, as well as the even and odd pages, traditionally have different margins, headers, and footers. This element has attributes that allow you to specify master pages for all these different pages. For example:

```
<fo:sequence-specification>
  <fo:sequence-specifier-repeating
    page-master-first="chapter_first"
    page-master-even="chapter_even"
    page-master-blank-even="chapter_blank"
    page-master-odd="chapter_odd"
    page-master-last-even="chapter_last_even"
    page-master-last-odd="chapter_last_odd"
    page-master-repeating="letter"
  />
</fo:sequence-specification>
```



Note

If the above attributes seem a little asymmetrical — for instance, there's no `page-master-blank-odd` attribute — that's because traditional publishing is asymmetrical. If you look carefully at the pages of this book, and indeed at almost any other book you own, you'll notice that the odd-numbered pages are always on the right, the even-numbered pages on the left, and that chapters always begin on a right-hand page. Chapters can end on either right-hand (odd) or left-hand (even) pages, but if they do end on an odd page, then a blank even page is inserted so the next chapter begins on an odd page.

Flows

The `fo:flow` object holds the actual content which will be placed on the instances of the master pages specified by the sequence specification. This content is composed of a sequence of `fo:block`, `fo:display-graphic`, `fo:display-link`, `fo:display-rule`, and other block-level elements. In this section, we'll stick to basic `fo:block` elements, which are roughly equivalent to HTML's `DIV` elements. Later in this chapter, we'll see a lot more block-level elements a flow can contain.

For example, here is a basic flow containing the names of several atoms, each in its own block:

```
<fo:flow name="xsl-body">
  <fo:block>Actinium</fo:block>
  <fo:block>Aluminum</fo:block>
  <fo:block>Americium</fo:block>
</fo:flow>
```

The name attribute of the `fo:flow`, here with the value `xsl-body`, specifies which of the five regions of the page this flow's content will be placed in. The allowed values are:

- ♦ `xsl-body`
- ♦ `xsl-after`
- ♦ `xsl-before`
- ♦ `xsl-start`
- ♦ `xsl-end`

For example, a flow for the header (in left-to-right, top-to-bottom English text) has a `flow-name` value of `xsl-before`. Here is a flow for a footer:

```
<fo:flow id="q2" flow-name="xsl-after">
  <fo:block>
    The XML Bible
    Chapter 15: XSL Formatting Objects
  </fo:block>
</fo:flow>
```

Static Content

Whereas each piece of the content of a `fo:flow` element appears on one page, each piece of the content of a `fo:static-content` element appears on every page; a header or a footer for example. You do not have to use `fo:static-content` elements, but if you do use them, they must appear before all the `fo:flow` elements in the page sequence.

`fo:static-content` elements have the same attributes and contents as a `fo:flow`. However, because a `fo:static-content` cannot break its contents across multiple pages, if necessary, it will generally have less content than a `fo:flow`. For example, here is a `fo:static-content` for a header:

```
<fo:static-content id="sc2" flow-name="xsl-before">
  <fo:block>
    The XML Bible
    Chapter 15: XSL Formatting Objects
  </fo:block>
</fo:static-content>
```

Page Numbering

Besides the usual `id` attribute that any formatting object element can have, `fo:page-sequence` element has six optional attributes that define page numbering for the sequence. These are:

- ♦ `initial-page-number`
- ♦ `format`
- ♦ `letter-value`
- ♦ `digit-group-sep`
- ♦ `n-digits-per-group`
- ♦ `sequence-src`

The `initial-page-number` attribute defines the number of the first page in this sequence. The most likely value for this attribute is 1, but it could be a larger number if the previous pages are in a different file. The remaining five attributes have exactly the same syntax and meaning as when used as attributes of the `xsl:number` element from the XSL transformation language.



The `xsl:number` element and the `format`, `letter-value`, `digit-group-sep`, `n-digits-per-group`, `sequence-src` attributes are discussed in the “Number to String Conversion” section in Chapter 14, *XSL Transformations*.

The `fo:page-number` formatting object is an empty inline element that inserts the number of the current page. The formatter is responsible for determining what that number is. This element has only a single attribute, `id`. Otherwise, you wrap `fo:page-number` in a `fo:inline-sequence`, `fo:block`, or similar element to apply font properties and the like to it. For example, this footer uses `fo:static-content` and `fo:page-number` to put the page number at the bottom of every page:

```
<fo:static-content id="sc2" flow-name="xsl-after">
  <fo:block>
    <fo:page-number/>
  </fo:block>
</fo:static-content>
```

This page sequence specifies that the page number uses small Roman numerals and begins counting from ten.

```
<fo:page-sequence initial-page-number="10" format="i">
  <!-- sequence specification -->
  <fo:static-content flow-name="xsl-after">
    <fo:block text-align-last="centered" font-size="10pt">
      <fo:page-number/>
    </fo:block>
  </fo:static-content>
```

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

Content

The content (as opposed to markup) of an XSL formatting objects document is mostly text. Additionally, external images can be linked to in a fashion similar to the `IMG` element of HTML. This content is stored in several kinds of elements including:

- ♦ Block-level formatting objects
- ♦ Inline formatting objects
- ♦ Table formatting objects
- ♦ Out-of-line formatting objects

All of these different kinds of elements will be descendants of either a `fo:flow` or a `fo:static-content` element. They are never placed directly on page masters or page sequences.

Block-level Formatting Objects

A block-level formatting object is drawn as a rectangular area separated by a line break and possibly extra whitespace from any content that precedes or follows it. Blocks may contain other blocks, in which case the contained blocks also are separated from the containing block by a line break and perhaps extra whitespace. Block-level formatting objects include:

- ♦ `fo:block`
- ♦ `fo:display-graphic`
- ♦ `fo:display-rule`
- ♦ `fo:display-included-container`
- ♦ `fo:display-sequence`
- ♦ `fo:list`
- ♦ `fo:list-item`

The `fo:block` element is the XSL equivalent of `display: block` in CSS or `DIV` in HTML. Blocks may be contained in `fo:flow` elements, other `fo:block` elements, and `fo:static-content` elements. `fo:block` elements may contain other `fo:block` elements, other block-level elements such as `fo:display-graphic` and `fo:display-rule`, and inline elements such as `fo:inline-sequence` and `fo:page-number`. They may also contain raw text. For example:

```

<fo:block>
  <fo:inline-sequence font-style="italic">
    The XML Bible
  </fo:inline-sequence>
  Page <fo:page-number/>
  <fo:inline-sequence>
    Chapter 15: XSL Formatting Objects
  </fo:inline-sequence>
</fo:block>

```

The `fo:block` elements generally have attributes for both area properties and text formatting properties. The text formatting properties are inherited by any child elements of the block unless overridden. Allowed properties include:

- ♦ **alignment properties:** `text-align` and `text-align-last`
- ♦ **aural properties:** `azimuth`, `cue`, `cue-after`, `cue-before`, `elevation`, `pause`, `pause-after`, `pause-before`, `pitch`, `pitch-range`, `play-during`, `richness`, `speak`, `speak-header`, `speak-numeral`, `speak-punctuation`, `speech-rate`, `stress`, `voice-family`, and `volume`
- ♦ **background properties:** `background`, `background-attachment`, `background-color`, `background-image`, `background-position`, and `background-repeat`
- ♦ **border properties:** `border-before-color`, `border-before-style`, `border-before-width`, `border-after-color`, `border-after-style`, `border-after-width`, `border-start-color`, `border-start-style`, `border-start-width`, `border-end-color`, `border-end-style`, `border-end-width`, `border-top-color`, `border-top-style`, `border-top-width`, `border-bottom-color`, `border-bottom-style`, `border-bottom-width`, `border-left-color`, `border-left-style`, `border-left-width`, `border-right-color`, `border-right-style`, `border-right-width`, `border`, `border-top`, `border-bottom`, `border-left`, `border-right`, `border-color`, `border-style`, and `border-width`
- ♦ **break properties:** `page-break-inside`, `widows`, `orphans`, and `wrap-option`
- ♦ **color properties:** `color`
- ♦ **column properties:** `span`
- ♦ **font properties:** `font-family`, `system-font`, `font-size`, `font-size-adjust`, `font-stretch`, `font-style`, `font-variant`, `font-weight`, and `font`
- ♦ **hyphenation properties:** `country`, `hyphenate`, `hyphenation-char`, `hyphenation-push-char-count`, `hyphenation-remain-char-count`, `language`, `script`, `hyphenation-keep`, and `hyphenation-ladder-count`
- ♦ **indentation properties:** `text-indent` and `last-line-end-indent`
- ♦ **layering property:** `z-index`

- ♦ **line-height properties:** `line-height`, `line-height-shift-adjustment` and, `line-stacking-strategy`
- ♦ **margin properties:** `margin-bottom`, `margin-left`, `margin-right`, `margin-top`, `margin`, `margin, space-before`, `space-after`, `start-indent`, and `end-indent`
- ♦ **padding properties:** `padding-top`, `padding-bottom`, `padding-left`, `padding-right`, `padding-before`, `padding-after`, `padding-start`, and `padding-end`
- ♦ **position properties:** `position`, `top`, `bottom`, `right`, and `left`
- ♦ **text direction properties:** `writing-mode`
- ♦ **visibility property:** `visibility`
- ♦ **whitespace properties:** `white-space-treatment`

Most of these are familiar from CSS. The rest will be discussed below. The other block-level elements have very similar property lists.

Inline Formatting Objects

An inline formatting object is drawn as a rectangular area that may contain text or other inline areas. Inline areas are most commonly arranged in lines running from left to right. When a line fills up, a new line is started below the previous one. However, the exact order in which inline elements are placed depends on the writing mode. For example, when working in Hebrew or Arabic, it makes sense to first place inline elements on the left and then fill to the right. Inline formatting objects include:

- ♦ `fo:bidi-override`
- ♦ `fo:character`
- ♦ `fo:first-line-marker`
- ♦ `fo:inline-graphic`
- ♦ `fo:inline-included-container`
- ♦ `fo:inline-rule`
- ♦ `fo:inline-sequence`
- ♦ `fo:list-item-body`
- ♦ `fo:list-item-label`
- ♦ `fo:page-number`
- ♦ `fo:page-number-citation`

Table-formatting Objects

The table formatting objects designed are the XSL equivalents of CSS2 table properties. However, tables do work somewhat more naturally in XSL than in CSS. For the most part, an individual table is a block-level object, while the parts of the table aren't really either inline or block level. However, an entire table can be turned into an inline object by wrapping it in a `fo:inline-included-container`.

There are nine XSL table-formatting objects:

- ♦ `fo:table-and-caption`
- ♦ `fo:table`
- ♦ `fo:table-caption`
- ♦ `fo:table-column`
- ♦ `fo:table-header`
- ♦ `fo:table-footer`
- ♦ `fo:table-body`
- ♦ `fo:table-row`
- ♦ `fo:table-cell`

The root of a table is not a `fo:table`, but rather a `fo:table-and-caption` which contains a `fo:table` and a `fo:caption`. The `fo:table` contains a `fo:table-header`, `fo:table-body`, and `fo:table-footer`. The table body contains `fo:table-row` elements which are divided up into `fo:table-cell` elements.

Out-of-line Formatting Objects

There are three out-of-line formatting objects:

- ♦ `fo:float`
- ♦ `fo:footnote`
- ♦ `fo:footnote-citation`

Out-of-line formatting objects “borrow” space from existing inline or block objects. On the page, they do not necessarily appear between the same elements they appeared between in the input formatting object XML tree.

Rules

A rule is a horizontal line inserted into text. XSL has two kinds of horizontal lines. The `fo:display-rule` formatting object is a block-level element that creates a horizontal line such as that produced by HTML's `<HR>` tag. The `fo:inline-rule` formatting object element is similar to the `fo:display-rule` element. However, as the name suggests, `fo:inline-rule` is an inline element instead of a block-level element. Thus, it may appear in the middle of a line of text and does not imply a line break. For example, this is a display rule:

However, this _____ is an inline rule.

Both the `fo:inline-rule` and `fo:display-rule` elements have six primary attributes that describe them:

1. **length**: the length of the line, such as 12pc or 5in
2. **rule-orientation**: escapement, horizontal, line-progression, or vertical
3. **rule-style**: exact values remain to be determined at the time of this writing
4. **rule-thickness**: the thickness of the line, such as 1px or 0.1cm
5. **vertical-align**: baseline, bottom, middle, sub, super, text-bottom, text-top, top, or a length or percentage of the line height
6. **color**: the color of the line, such as pink or #FFCCCC

For example, this is a green block-level rule that's 7.5 inches long and 2 points thick:

```
<fo:display-rule length="7.5in"
  line-thickness="2pt" color="#00FF00"/>
```

Additionally, the `fo:display-rule` can have most of the usual attributes of a block-level element like those describing margins and padding, and a `fo:inline-rule` can have the usual attributes of an inline element like `line-height`. The exceptions are those attributes that are directly related to text, like `font-family`. Obviously, these attributes make no sense for a rule.

Graphics

XSL provides two means of embedding pictures in a rendered document. The `fo:display-graphic` element inserts a block-level graphic. The `fo:inline-graphic` element inserts an inline graphic. These two elements provide the equivalent of an HTML `IMG` tag. Six attributes describe the picture:

1. `href`: the URI of the image file
2. `min-height`: the minimum vertical height of the image
3. `min-width`: the minimum horizontal width of the image
4. `max-height`: the maximum vertical height of the image
5. `max-width`: the maximum horizontal width of the image
6. `scale`: with a value `max`, expand the graphic to the size of `max-height` and `max-width`; with the value `max-uniform`, expand the graphic by the same amount in the vertical and horizontal directions to either the `max-height` or `max-width`, whichever comes first; with the value a single real number, multiply both height and width by that number; with the value two real numbers, multiply width by the first and height by the second

For example, consider this standard HTML `IMG` element:

```
<IMG SRC="logo.gif" WIDTH="100" HEIGHT="100"
      ALIGN="right" ALT="alt text" BORDER="0">
```

The `fo:display-graphic` element equivalent looks like this:

```
<fo:display-graphic image="logo.gif"
                    height="100px" width="100px" />
```

Links

For online presentations only, XSL provides the `fo:simple-link` element. Assuming a Web browser-style user interface, clicking anywhere on the contents of a link element jumps to the link target. This element can act as either a block-level or inline link depending on what it contains. The link behavior is controlled by these six attributes:

- ♦ `external-destination`
- ♦ `internal-destination`
- ♦ `indicate-destination`

- ♦ `show-destination`
- ♦ `space-above-destination-block`
- ♦ `space-above-destination-start`

A link to a remote document target specifies the URI through the value of the `external-destination` attribute. The document at this URI should be loaded when the link is activated. In GUI environments, the link most likely is activated by clicking on the link contents. For example:

```
<fo:block> Be sure to visit the
  <fo:simple-link
    external-destination="http://metalab.unc.edu/xml/">
    Cafe con Leche Web site!
  </fo:simple-link>
</fo:block>
```

You can also link to another node in the same document by using the `internal-destination` attribute. The value of this attribute is not a URI, but rather the ID of the element you're linking to. You should not specify both the internal and external destination for one link.

The other four attributes affect the appearance and behavior of the link. The `indicate-destination` attribute has a Boolean value (`true` or `false`, `false` by default) that specifies whether, when the linked item is loaded it should somehow be distinguished from non-linked parts of the same document. For example, if you follow a link to one `ATOM` element in a table of 100 atoms, the specific atom you were connecting to might be in bold face while the other atoms would be in normal type. The exact details are system dependent.

The `show-destination` attribute has two possible values, `replace` (the default) and `new`. With a value of `replace`, when a link is followed it replaces the existing document in the same window. With a value of `new`, when a link is followed, the targeted document is opened in a new window.

When a browser follows an HTML link into the middle of a document, generally the specific linked element is positioned at the tippy-top of the window. The `space-above-destination-start` and `space-above-destination-block` attributes let you specify that the browser should position the linked element further down in the window by leaving a certain amount of space (not empty space, it will generally contain the content preceding the linked element) above the linked item.

In addition, the link may have an usual property such as color that will be inherited by the link's contents. This allows you to format content that's in a link differently from content that's not; for example, by underlining all links. However, XSL formatting objects do not provide a means to distinguish between visited, unvisited, and active links, unlike CSS and HTML which do.

Lists

The `fo:list-block` formatting object element describes a block-level list element. (There are no inline lists.) A list may or may not be bulleted, numbered, indented, or otherwise formatted. Each `fo:list-block` element contains either a series of `fo:list-item` elements or `fo:list-item-label` `fo:list-item-body` pairs. (It cannot contain both.) A `fo:list-item` must contain a `fo:list-item-label` and a `fo:list-item-body`. The `fo:list-item-label` contains the bullet, number, or other label for the list item. The `fo:list-item-body` contains the actual content of the list item. To summarize, a `fo:list-block` contains `fo:list-item` elements. Each `fo:list-item` contains a `fo:list-item-label` and `fo:list-item-body`. However, the `fo:list-item` elements can be omitted. For example:

```
<fo:list-block>
  <fo:list-item>
    <fo:list-item-label>*</fo:list-item-label>
    <fo:list-item-body>Actinium</fo:list-item-body>
  </fo:list-item>
  <fo:list-item>
    <fo:list-item-label>*</fo:list-item-label>
    <fo:list-item-body>Aluminum</fo:list-item-body>
  </fo:list-item>
</fo:list-block>
```

Or, with the `fo:list-item` tags removed:

```
<fo:list-block>
  <fo:list-item-label>*</fo:list-item-label>
  <fo:list-item-body>Actinium</fo:list-item-body>
  <fo:list-item-label>*</fo:list-item-label>
  <fo:list-item-body>Aluminum</fo:list-item-body>
</fo:list-block>
```

The `fo:list-block` element has three special attributes:

1. `provisional-label-separation`: the distance between the list item label and the list item body, given as a triplet of maximum;minimum;optimum, such as 2cm;0.5cm;1cm.
2. `provisional-distance-between-starts`: the distance between the start edge of the list item label and the start edge of the list item body.
3. `space-between-list-rows`: vertical distance between successive list items, given as a triplet of maximum;minimum;optimum, such as 36pt;4pt;12pt.

The `fo:list-item` element has the standard block-level properties for backgrounds, position, aural rendering, borders, padding, margins, line and page breaking.

Tables

The fundamental table element in XSL is a `fo:table-and-caption`. This is a block-level object. However, it can be turned into an inline object by wrapping it in a `fo:inline-included-container` or an out-of-line object by wrapping it in a `fo:float`. The table model is quite close to HTML's. Table 15-1 shows the equivalence between HTML 4.0 table elements and XSL formatting objects:

Table 15-1 HTML Tables versus XSL Formatting Object Tables	
<i>HTML Element</i>	<i>XSL FO Element</i>
TABLE	<code>fo:table-and-caption</code>
no equivalent	<code>fo:table</code>
CAPTION	<code>fo:table-caption</code>
COL	<code>fo:table-column</code>
COLGROUP	no equivalent
THEAD	<code>fo:table-header</code>
TBODY	<code>fo:table-body</code>
TFOOT	<code>fo:table-footer</code>
TD	<code>fo:table-cell</code>
TR	<code>fo:table-row</code>

The `fo:table-and-caption` contains an optional `fo:caption` element and one `fo:table` element. The caption can contain any block-level elements you care to place in the caption. By default, captions are placed before the table, but this can be adjusted by setting the `caption-side` property of the `table-and-caption` element to one of these eight values:

- ♦ before
- ♦ after
- ♦ start
- ♦ end
- ♦ top
- ♦ bottom
- ♦ left
- ♦ right

For example, here's a table with a caption on the bottom:

```
<fo:table-and-caption caption-side="bottom">
  <fo:table-caption>
    <fo:block font-weight="bold"
              font-family="Helvetica, Arial, sans"
              font-size="12pt">
      Table 15-1: HTML Tables vs. XSL Formatting Object Tables
    </fo:block>
  </fo:table-caption>
  <fo:table>
    <!-- table contents go here -->
  </fo:table>
</fo:table-and-caption>
```

The `fo:table` element contains an optional `fo:table-column`, `fo:table-header`, an optional `fo:table-footer`, and one or more `fo:table-body` elements. The `fo:table-body` is divided into `fo:table-row` elements. Each `fo:table-row` is divided into `fo:table-cell` elements. The `fo:table-header` and `fo:table-footer` can either be divided into `fo:table-cell` or `fo:table-row` elements. For example, here's a simple table that matches the first three rows of Table 15-1:

```
<fo:table>
  <fo:table-header>
    <fo:table-cell>
      <fo:block font-family="Helvetica, Arial, sans"
                font-size="11pt" font-weight="bold">
        HTML Element
      </fo:block>
    </fo:table-cell>
    <fo:table-cell>
      <fo:block font-family="Helvetica, Arial, sans"
                font-size="11pt" font-weight="bold">
        XSL FO Element
      </fo:block>
    </fo:table-cell>
  </fo:table-header>
  <fo:table-body>
    <fo:table-row>
      <fo:table-cell>
        <fo:block font-family="Courier, monospace">
          TABLE
        </fo:block>
      </fo:table-cell>
      <fo:table-cell>
        <fo:block font-family="Courier, monospace">
          fo:table-and-caption
        </fo:block>
      </fo:table-cell>
    </fo:table-row>
    <fo:table-row>
      <fo:table-cell>
        <fo:block>no equivalent</fo:block>
```

```

</fo:table-cell>
<fo:table-cell>
  <fo:block font-family="Courier, monospace">
    fo:table
  </fo:block>
</fo:table-cell>
</fo:table-row>
</fo:table-body>
</fo:table>

```

Table cells can span multiple rows and columns by setting the `n-columns-spanned` and/or `n-rows-spanned` attributes to an integer giving the number of rows or columns to span. The optional `column-number` attribute can change which column the spanning begins in. The default is the current column.

Borders can be drawn around table parts using the normal border properties which we'll discuss later. The `empty-cells` attribute has the value `show` or `hide`, `show` if borders are to be drawn around cells with no content, `hide` if not. The default is `show`.

Most table parts do not use the standard width and height properties. Instead, they have equivalent attributes. Any or all of these may be omitted, in which case the formatter will simply size everything as it sees fit:

- ♦ `table: table-width, table-height`
- ♦ `table-caption: caption-width, height determined automatically by the formatter`
- ♦ `table-row: row-height, width determined by contents`
- ♦ `table-cell: cell-height, column-number, column-width, n-columns-spanned, n-rows-spanned`

The `fo:table-row` element has optional `may-break-after-row` and `may-break-before-row` attributes with the values `yes` or `no` that determine whether a page break is allowed before and after the row. The defaults are both `yes`.

When a long table extends across multiple pages, the header and footer are sometimes repeated on each page. You can specify this behavior with the `table-omit-middle-header` and `table-omit-middle-footer` attributes of the `fo:table` element. The value `yes` indicates that the header or footer is to be repeated from page to page. The value `no` indicates that it is not. The default is `no`.

The optional `fo:table-column` element is an empty element that specifies values for all cells in a particular column. The cells it applies to are identified by the `column-number` attribute. `fo:table-column` does not actually contain any cells. A `fo:table-column` can apply properties to more than one consecutive column by setting the `n-columns-spanned` property to an integer greater than one. The most common property to set in a `fo:table-column` is `column-width` (a signed length) but the standard border, padding, and background properties (discussed below) can also be set.

Characters

The `fo:character` formatting object replaces a particular character or string of characters in the input with a different character in the output. You might use this to translate between the American decimal point and the French decimal comma, for example. The `character` attribute specifies what replacement character to use. For example, this template rule substitutes `*` for the characters in a `PASSWORD` element:

```
<xsl:template match="PASSWORD">
  <fo:character character="*">
    <xsl:value-of select="."/>
  </fo:character>
</xsl:template>
```

However, this use is rare. The main purpose of the `fo:character` element is so that formatting engines can treat each character and glyph as its own element. If you're not writing a formatting engine, you probably can ignore this element.

Sequences

Sequences have no particular effect on the layout of either inline or block-level boxes. They're simply elements on which you can hang formatting attributes such as `font-style` or `text-indent` for application to the sequence's children.

The `fo:display-sequence` formatting object element is a container that groups block-level objects together. In fact, it can only hold block-level elements such as `fo:display-graphic` and `fo:block`. It cannot contain inline elements or raw text.

The `fo:inline-sequence` formatting object element is a container that groups inline objects together. It cannot contain block-level elements. For example, you can use inline-sequence elements to add style to various parts of the footer, like this:

```
<fo:flow id="q2" flow-name="xsl-after">
  <fo:block font-style="bold" font-size="10pt"
    font-family="Arial, Helvetica, sans">
    <fo:inline-sequence font-style="italic"
      text-align="start">
      The XML Bible
    </fo:inline-sequence>
    <fo:inline-sequence text-align="centered">
      Page <fo:page-number/>
    </fo:inline-sequence>
    <fo:inline-sequence text-align="right">
      Chapter 15: XSL Formatting Objects
    </fo:inline-sequence>
  </fo:block>
</fo:flow>
```

Footnotes

The `fo:footnote` element represents a footnote. The author places the `fo:footnote` element in the flow exactly where the footnote reference like ¹ or * will occur. The `fo:footnote` element contains both a `fo:footnote-reference` and a block-level element containing the text of the footnote. However, only the footnote reference is inserted inline. The formatter places the note text in the after region (generally the footer) of the page.

For example, this footnote uses an asterisk as a footnote marker and refers to “*JavaBeans*, Elliotte Rusty Harold (IDG Books, Foster City, 1998), p. 147”. Standard XSL properties like `font-size` and `vertical-align` are used to format both the note marker and the text in the customary fashion.

```
<fo:footnote>
  <fo:footnote-reference
    font-size="smaller" vertical-align="super">
    *
  </fo:footnote-reference>
  <fo:block font-size="smaller">
    <fo:inline-sequence
      font-size="smaller" vertical-align="super">
      *
    </fo:inline-sequence>
    <fo:inline-sequence
      font-style="italic">JavaBeans</fo:inline-sequence>,
      Elliotte Rusty Harold
      (IDG Books, Foster City, 1998), p. 147
    </fo:block>
  </fo:footnote>
```


Tip

The formatting objects vocabulary doesn't provide any means of automatically numbering and citing footnotes, but this can be done by judicious use of `xsl:number` in the transformation stylesheet. XSL transformations also make end notes easy as well.

Floats

A `fo:float` produces a floating box anchored to the top of the region where it occurs. A `fo:float` is most commonly used for graphics, charts, tables, or other out-of-line content that needs to appear somewhere on the page, but exactly where it appears in not particularly important. For example, here is the code for a floating graphic with a caption embedded in the middle of a paragraph:

```
<fo:block>
  Although PDF files are themselves ASCII text,
  this isn't a book about PostScript, so there's
  nothing to be gained by showing you the exact
```

```
output of the above command. If you're curious,
open the PDF file in any text editor.
Instead, Figure 15-1
<fo:float>
  <fo:display-graphic
    image="3236-7fg1501.jpg"
    height="485px" width="623px" />
  <fo:block font-family="Helvetica, sans">
    <fo:inline-sequence font-weight="bold">
      Figure 15-1:
    </fo:inline-sequence>
    The PDF file displayed in Netscape Navigator
  </fo:block>
</fo:float>
shows the rendered file displayed in
Netscape Navigator using the Acrobat plug-in.
</fo:block>
```

The formatter makes a best effort to place the graphic somewhere on the same page where the content surrounding the `fo:float` appears, though this is not always possible, in which case it moves the object to the subsequent page. Within these limits, it's free to place it anywhere on that page.

XSL Formatting Properties

By themselves, formatting objects say relatively little about how content is formatted. They merely put content in abstract boxes, which are placed in particular parts of a page. Attributes on the various formatting objects determine how the content in those boxes is styled.

As already mentioned, there are about 200 separate formatting properties. Not all properties can be attached to all elements. For instance, there isn't much point to specifying the `font-style` of a `fo:display-graphic`. Most properties, however, can be applied to more than one kind of formatting object element. (The few that can't, such as `href` and `provisional-label-separation`, are discussed above with the formatting objects they apply to.) When a property is common to multiple formatting objects, it shares the same syntax and meaning across the objects. For example, you use identical code to format the `fo:list-label` in 14-point Times bold as you do to format a `fo:block` in 14-point Times bold.

Many of the XSL properties are similar to CSS properties. The value of a CSS `font-family` property is the same as the value of an XSL `font-family` attribute. If you've read about CSS in Chapters 12 and 13, you're already more than half finished learning XSL properties.

Units and Data Types

The value of an XSL formatting property may be a keyword such as *auto*, *italic*, or *transparent*; or it may be a literal value such as *true*, *5px*, *-5.0cm*, or <http://www.w3.org/index.html>. Literal values in XSL are given as one of 24 data types, which are listed in Table 15-2.

Table 15-2
Formatting Property Data Types

<i>Data Type</i>	<i>Definition</i>	<i>Examples</i>
Name	An XML name token.	q1 copyright
ID	A unique XML name token.	q1 copyright
IDREF	A name token that matches the ID of an element in the document.	q1 copyright
Boolean	Either the string “true” or the string “false”.	true false
Char	A single, non-whitespace Unicode character.	A —
Signed Integer	A sequence of digits, optionally prefixed by a plus or minus sign.	0 -28 +1000000000
Unsigned Integer	A sequence of digits.	0 28 1000000000
Positive Integer	A sequence of digits that includes at least one nonzero digit.	28 1000000000
Signed Real	A floating point number in the format sign-digits-period-digits. Exponential notation is not supported. The + is optional for positive numbers.	+0.879 -31.14 2.71828
Unsigned Real	A non-negative floating point number in the format digits-period-digits. Exponential notation is not supported.	0.0 31.14 2.71828
Positive Real	A positive floating point number in the format digits-period-digits. Exponential notation is not supported.	0.01 31.14 2.71828

Continued

Table 15-2 (continued)

Data Type	Definition	Examples
Signed Length	A signed integer or signed real followed by a unit.	5px -0.5in
Unsigned Length	An unsigned integer or unsigned real number followed by a unit.	10px 0.5cm
Positive Length	A positive integer or positive real number followed by a unit.	10px 1pc
Percent	A signed real number that must be divided by 100 to get its actual value.	100.0 -43.2 0.0
Space Specifier	Minimum length semicolon maximum length semicolon optimal length semicolon precedence semicolon conditionality.	0px;72px;12px; force;discard
Limit Specifier	Minimum length semicolon maximum length.	0px;72px
Color	A named color or a hexadecimal triple in the form #RRGGBB.	white #FFFFFF
URI	A Uniform Resource Identifier; in practice, a URL.	http://www.w3 .org/index.html /index.html / ../index.html
Language	An ISO 639 language code.	en la
Font Name	The name of a font, either actual or symbolic.	Times New Roman serif
Font List	Font names separated by commas and possibly whitespace.	Times New Roman, Times, serif
Enumeration	An XML enumeration.	(airplane train car horse)
String	Any sequence of characters.	Fred Lucy and Ethel Castles don't have phones.

Informational Properties

There are two informational properties, which can be applied to any formatting object. However, neither has a direct affect on the formatting. In essence, these are non-formatting properties.

The id Property

The first such property is `id`. This is an XML ID-type attribute. The value of this property must, therefore, be an XML name that's unique within the style sheet and within the output formatting object document. The last requirement is a little tricky since it's possible that one template rule in the stylesheet may generate several hundred elements in the output document. The `generate-id()` function of XSL transformations can be useful here.

The language Property

The second such property is `language`. This specifies the language of the content contained in this element. Generally, the value of this property is an ISO 639 language code such as `en` (English) or `la` (Latin). It may also be the keyword `none` or `use-document`. The latter means to simply use the language of the input as specified by the `xml:lang` attribute. For example, consider the first verse of Caesar's *Gallie Wars*:

```
<fo:block id="verse1.1.1" language="la">  
  Gallia est omnis divisa in partes tres,  
  quarum unam incolunt Belgae, aliam Aquitani,  
  tertiam qui ipsorum lingua Celtae, nostra Galli appellantur  
</fo:block>
```

Although the `language` property has no direct effect on formatting, it may have an indirect effect if the formatter selects layout algorithms depending on the language. For instance, the formatter may use different default writing modes for Arabic and English text. This carries over into determination of the start and end regions, and the inline progression direction.

Paragraph Properties

Paragraph properties are styles that normally are thought of as applying to an entire block of text in a traditional word processor, though perhaps block-level text properties are more appropriate here. For example, indentation is a paragraph property, because you can indent a paragraph but you can't indent a single word separate from its enclosing paragraph.

Break Properties

The break properties specify where page breaks are and are not allowed. There are five loosely related break properties:

- ♦ `keep-with-next`
- ♦ `keep-with-previous`
- ♦ `break-before`
- ♦ `break-after`
- ♦ `inhibit-line-breaks`

The `keep-with-next` and `keep-with-previous` properties are Booleans that specify whether the formatting object should remain in the same parent-formatting object as the next and previous formatting objects, respectively. This has the effect of keeping two formatting objects on the same page, but it's more strict than that.

The `break-before` property inserts a break before the formatting object starts. Possible things to break include `column`, `page`, `odd-page` and `even-page`. The value may also be `none` or `auto-page`. The `break-after` property inserts a break after the formatting object finishes. The same values are used as for `break-before`. For example, this template rule ensures that each SONNET of sufficiently small size prints on a page of its own:

```
<xsl:template match="SONNET">
  <fo:block break-before="page" break-after="page">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Finally, the `inhibit-line-breaks` property is a Boolean that can be set to `true` to indicate that not even a line break is allowed, much less a page break.

Hyphenation Properties

The hyphenation properties determine whether hyphenation is allowed and how it should be used. This applies only to soft or “optional” hyphens such as the ones sometimes used to break long words at the end of a line. It does not apply to hard hyphens such as the ones in the word *mother-in-law*, though these hyphens may affect where soft hyphens are allowed. There are six hyphenation properties. They are:

- ♦ `hyphenate`: automatic hyphenation is allowed only if this Boolean property has the value `true`
- ♦ `hyphenation-char`: the Unicode character used to hyphenate words, such as - in English
- ♦ `hyphenation-keep`: one of the four keywords (`column`, `none`, `page`, `spread`) that specify whether hyphenation is allowed at the end of a facing page pair or column

- ♦ `hyphenation-ladder-count`: an unsigned integer that specifies the maximum number of hyphenated lines that may appear in a row
- ♦ `hyphenation-push-char-count`: an unsigned integer that specifies the minimum number of characters that must follow an automatically inserted hyphen. (Short syllables look bad in isolation.)
- ♦ `hyphenation-remain-char-count`: an unsigned integer specifying the minimum number of characters that must precede an automatically inserted hyphen

Hyphenation also depends on the language and script in use. Thus, the following three properties have particular impact here:

- ♦ `country`
- ♦ `language`
- ♦ `script`

For example:

```
<fo:block hyphenate=true
          hyphenation-char="-"
          hyphenation-keep="none"
          hyphenation-ladder-count="2"
          hyphenation-push-char-count="4"
          hyphenation-remain-char-count="4" >
  some content...
</fo:block>
```

XSL does not specify a syllable-breaking algorithm to determine where a soft hyphen may be applied. Even with these properties allowing hyphenation, it's still completely up to the formatter to figure out how to hyphenate individual words.

The vertical-align Property

The `vertical-align` property determines the vertical position of a formatting object on its line. It is identical in behavior to the CSS2 property of the same name. There are eight possible keyword values for this property:

1. `baseline`: align the baseline of the box with the baseline of the line box
2. `sub`: align the baseline of the box with the baseline of subscripts inside the line box
3. `super`: raise the baseline of the box to the baseline of superscripts in the line box
4. `top`: align the top of box with the top of the line box

5. `middle`: align the midpoint of the box with the baseline of the line box, plus half the x-height of the line box
6. `bottom`: align the bottom of the box with the bottom of the line box
7. `text-top`: align the top of the box with the top of the font
8. `text-bottom`: align the bottom of the box with the bottom of the font

You can also set `vertical-align` to a signed length that raises or lowers the box by the specified distance from the baseline.

Indentation Properties

The four indent properties `start-indent`, `end-indent`, `text-indent` and `last-line-end-indent` specify how far lines are indented from the edge of the text. The `start-indent` property offsets all lines from the start edge (left edge in English). The `end-indent` property offsets all lines from the end edge (right edge in English). The `text-indent` property offsets only the first line from the start edge. The `last-line-end-indent` property offsets only the last line from the start edge. Values are given as a signed length. Using a positive value for `start-indent` and a negative value for `text-indent` creates hanging indents. For example, a standard paragraph with 0.5-inch, first-line indent might be formatted this way:

```
<fo:block text-indent="0.5in">
  The first line of this paragraph is indented
</fo:block>
```

A block quote with a 1-inch indent on all lines on both sides is formatted like this:

```
<fo:block start-indent="1.0in" end-indent="1.0in">
  This text is offset one inch from both edges.
</fo:block>
```

Character Properties

Character properties describe the qualities of individual characters, although they can apply to elements that contain characters such as `fo:block` and `fo:list-item-body` elements. These include color, font, style, weight, and similar properties.

The color Property

The `color` property sets the foreground color of the contents using the same syntax as the CSS `color` property. For example, this colors the text “Lions and tigers and bears, oh my!” pink:

```
<fo:inline-sequence color="#FFCCCC">
  Lions and tigers and bears, oh my!
</fo:inline-sequence>
```

Font Properties

Any formatting object that holds text can have a wide range of font properties. Most of these are familiar from CSS, including:

- ♦ **font-family**: a list of font names in order of preference
- ♦ **font-size**: a signed length
- ♦ **font-size-adjust**: the preferred ratio between the x-height and size of a font, specified as an unsigned real number or none
- ♦ **font-stretch**: the “width” of a font, given as one of the keywords *condensed*, *expanded*, *extra-condensed*, *extra-expanded*, *narrower*, *normal*, *semi-condensed*, *semi-expanded*, *ultra-condensed*, *ultra-expanded*, or *wider*
- ♦ **font-style**: the style of font specified as one of the keywords *italic*, *normal*, *oblique*, *reverse-normal*, or *reverse-oblique*
- ♦ **font-variant**: either *normal* or *small-caps*
- ♦ **font-weight**: the thickness of the strokes that draw the font, given as one of the keywords *100*, *200*, *300*, *400*, *500*, *600*, *700*, *800*, *900*, *bold*, *bolder*, *lighter*, or *normal*

The text-transform Property

The `text-transform` property defines how text is capitalized, and is identical to the CSS property of the same name. The four possible values are:

- ♦ **none**: don’t; change the case (the default)
- ♦ **capitalize**: make the first letter of each word uppercase and all subsequent letters lowercase
- ♦ **uppercase**: make all characters uppercase
- ♦ **lowercase**: make all characters lowercase

This property is somewhat language specific. (Chinese, for example, doesn’t have separate upper and lower cases.) Formatters are free to ignore the case recommendations when they’re applied to non-Latin-1 text.

The text-shadow Property

The `text-shadow` property applies a shadow to text. This is similar to a background color, but differs in that the shadow attaches to the text itself rather than the box containing the text. The value of `text-shadow` can be the keyword *none* or a named or RGB color. For example:

```
<fo:inline-sequence text-shadow="FFFF66">  
  This sentence is yellow.  
</fo:inline-sequence>
```

The text-decoration Property

The `text-decoration` property is identical to the CSS2 `text-decoration` property. It has these five possible values:

- ♦ none
- ♦ underline
- ♦ overline
- ♦ line-through
- ♦ blink

The default is none.

The score-space Property

Scoring is a catchall word for underlining, ~~line-through~~, double strike-through, and so forth. The `score-space` property determines whether whitespace is scored. For example, if `score-spaces` is true, an underlined sentence looks like this. If `score-spaces` is false, an underlined sentence looks like this.

Sentence Properties

Sentence properties apply to groups of characters, that is, a property that makes sense only for more than one letter at a time, such as the space between letters or words.

Letter Spacing Properties

Kerning of text is a slippery measure of how much space separates two characters. It's not an absolute number. Most formatters adjust the space between letters based on local necessity, especially in justified text. Furthermore, high-quality fonts use different amounts of space between different glyphs. However, you can make text looser or tighter overall.

The `letter-spacing` property adds additional space between each pair of glyphs, beyond that provided by the kerning. It's given as a signed length specifying the desired amount of extra space to add. For example:

```
<fo:block letter-spacing="1.5px">
  This is fairly loose text
</fo:block>
```

You can make the length negative to tighten up the text. Formatters, however, will generally impose limits on how much extra space they allow to be added to, or removed from, the space between letters.

Word Spacing Properties

The word-spacing property adjusts the amount of space between words. Otherwise, it behaves much like the letter spacing properties. The value is a signed length giving the amount of extra space to add between two words. For example:

```
<fo:block word-spacing="0.3cm">
  This is pretty loose text.
</fo:block>
```

Line Spacing Properties

An XSL formatting engine divides block areas into line areas. You cannot create line areas directly from XSL. However, with these five properties you can affect how they're vertically spaced:

- ♦ **line-height**: the minimum height of a line
- ♦ **line-height-shift-adjustment**: consider-shifts if subscripts and superscripts should expand the height of a line, disregard-shifts if they shouldn't
- ♦ **line-stacking-strategy**: line-height (the CSS model and the default); font-height (make the line as tall as the font height after addition of font-height-override-before and font-height-override-after); or max-height (distance between the maximum ascender height and maximum descender depth)
- ♦ **font-height-override-after**: a signed length specifying additional vertical space added after each line; can also be the keyword `use-font-metrics` (the default) to indicate that this depends on the font
- ♦ **font-height-override-before**: a signed length specifying the minimum additional vertical space added before each line; can also be the keyword `use-font-metrics` (the default) to indicate that this depends on the font

The line height also depends to a large extent on the size of the font in which the line is drawn. Larger font sizes will naturally have taller lines. For example, the following opening paragraph from Mary Wollstonecraft's *Of the Rights of Woman* is effectively double-spaced:

```
<fo:block font-size="12pt" line-height="24pt">
  In the present state of society it appears necessary to go
  back to first principles in search of the most simple truths,
  and to dispute with some prevailing prejudice every inch of
  ground. To clear my way, I must be allowed to ask some plain
  questions, and the answers will probably appear as
  unequivocal as the axioms on which reasoning is built;
  though, when entangled with various motives of action, they
  are formally contradicted, either by the words or conduct
  of men.
</fo:block>
```

Text Alignment Properties

The `text-align` and `text-align-last` properties specify how the inline content is horizontally aligned within its box. The six possible values are:

1. `start`: left aligned in right-to-left scripts
2. `centered`: centered
3. `end`: right aligned in right-to-left scripts
4. `justify`: expanded with extra space as necessary to fill out the line
5. `page-inside`: align with the inside edge of the page, that is, the right edge on the left page of two facing pages or the left edge on the right page of two facing pages
6. `page-outside`: align with the outside edge of the page, that is, the left edge on the left page of two facing pages or the right edge on the right page of two facing pages

The `text-align-last` property enables you to specify a different value for the last line in a block. This is especially important for justified text, where the last line often doesn't have enough words to be attractively justified. The possible values are `start`, `end`, `justified`, and `relative`. The `relative` value uses the same value as the `text-align` property unless `text-align` is `justified`, in which case the last line will align with the start edge instead.

Whitespace Property

The `whitespace-treatment` property specifies what the formatting engine should do with whitespace that's still present after the original source document is transformed into formatting objects. There are three possible values:

1. `preserve`: leave the whitespace as it is
2. `collapse`: collapse all whitespace to a single space
3. `ignore`: throw away leading and trailing whitespace

My preference is to preserve all whitespace that's still left after transformation. If it's insignificant, it's easy for the transformation process to throw it away using `xsl:strip-space`.

The wrap-option Property

The `wrap-option` property determines how text that's too long to fit on a line is handled. This property has two possible keyword values:

1. `wrap`: soft wrap the text to the next line
2. `no-wrap`: do not wrap the text

Area Properties

Area properties are applied to boxes. These may be either block-level or inline boxes. Each of these boxes has:

- ♦ a background
- ♦ margins
- ♦ borders
- ♦ padding
- ♦ a size

Background Properties

The background properties are basically identical to the CSS1 background properties. There are five:

- ♦ The `background-color` property specifies the color of the box's background. Its value is either a color or the keyword `transparent`.
- ♦ The `background-image` property gives the URI of an image to be used as a background. The value can be the keyword `none`.
- ♦ The `background-attachment` property specifies whether the background image is attached to the window or the document. Its value is one of the two keywords `fixed` or `scroll`.
- ♦ The `background-position` property specifies how a background image is placed in the box. Possible values include `center`, `left`, `right`, `bottom`, `middle`, `top`, or a coordinate.
- ♦ The `background-repeat` property specifies how and whether a background image is tiled if it is smaller than its box. Possible values include `repeat`, `no-repeat`, `repeat-x`, and `repeat-y`.

The following block shows the use of the `background-image`, `background-position`, `background-repeat`, and `background-color` properties:

```
<fo:block background-image="/bg/paper.gif"
           background-position="0,0"
           background-repeat="repeat"
           background-color="white">
  Two strings walk into a bar...
</fo:block>
```

Border Properties

The border properties describe the appearance of a border around the box. They are mostly the same as the CSS border properties. However, as well as `border-XXX-bottom`, `border-XXX-top`, `border-XXX-left`, and `border-XXX-right` properties, the XSL versions also have `border-XXX-before`, `border-XXX-after`, `border-XXX-start`, and `border-XXX-end` versions. There are 31 border properties in all. These are:

- ♦ **color:** `border-color`, `border-before-color`, `border-after-color`, `border-start-color`, `border-end-color`, `border-top-color`, `border-bottom-color`, `border-left-color`, `border-right-color`. The default border color is black.
- ♦ **width:** `border-width`, `border-before-width`, `border-after-width`, `border-start-width`, `border-end-width`, `border-top-width`, `border-bottom-width`, `border-left-width`, `border-right-width`.
- ♦ **style:** `border-style`, `border-before-style`, `border-after-style`, `border-start-style`, `border-end-style`, `border-top-style`, `border-bottom-style`, `border-left-style`, `border-right-style`
- ♦ **shorthand properties:** `border`, `border-top`, `border-bottom`, `border-left`, `border-right`, `border-color`, `border-style`, `border-width`

For example, this draws a 2-pixel-wide blue box around a block:

```
<fo:block border-before-color="blue" border-before-width="2px"
           border-after-color="blue" border-after-width="2px"
           border-start-color="blue" border-start-width="2px"
           border-end-color="blue" border-end-width="2px">
  Two strings walk into a bar...
</fo:block>
```

Padding Properties

The padding properties specify the amount of space between the border of the box and the contents of the box. The border of the box, if shown, falls between the margin and the padding. The padding properties are mostly the same as the CSS padding properties. However, as well as `padding-bottom`, `padding-top`, `padding-left`, and `padding-right` properties the XSL versions also have `padding-before`, `padding-after`, `padding-start`, and `padding-end` versions. Thus, in total there are eight padding properties, each of which has a signed length for a value. These are:

- ♦ `padding-after`
- ♦ `padding-before`
- ♦ `padding-bottom`

- ♦ padding-end
- ♦ padding-left
- ♦ padding-start
- ♦ padding-right
- ♦ padding-top

For example, this block has 0.5 centimeters of padding on all sides:

```
<fo:block padding-before="0.5cm" padding-after="0.5cm"
           padding-start="0.5cm" padding-end="0.5cm">
  Two strings walk into a bar...
</fo:block>
```

Margin Properties for Blocks

There are five margin properties, each of whose value is given as an unsigned length. These are:

- ♦ margin-top
- ♦ margin-bottom
- ♦ margin-left
- ♦ margin-right
- ♦ margin

However, these properties are only here for compatibility with CSS. In general it's recommended that you use the following properties instead, which fit better into the XSL formatting model:

- ♦ space-before
- ♦ space-after
- ♦ start-indent
- ♦ end-indent

The **space-before** and **space-after** properties are exactly equivalent to the **margin-top** and **margin-bottom** properties respectively. The **start-indent** property is equivalent to the sum of **padding-left**, **border-left-width**, and **margin-left**. The **end-indent** property is equivalent to the sum of **padding-right**, **border-right-width**, and **margin-right**. Figure 15-3 should make this clearer.

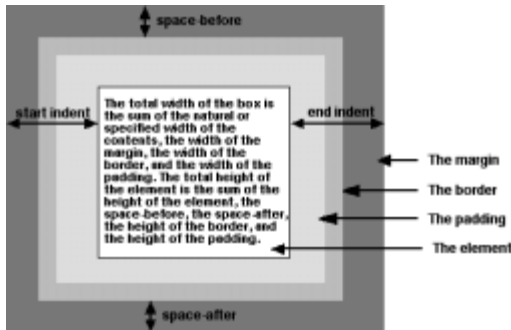


Figure 15-3: Padding, indents, borders, and space before and after for an XSL box

For example, this block has 0.5 centimeters of margin at its start and end sides:

```
<fo:block start-indent="0.5cm" end-indent="0.5cm">
  Two strings walk into a bar...
</fo:block>
```

Margin Properties for Inline Boxes

There are two margin properties that apply only to inline elements. These are:

- ♦ space-end
- ♦ space-start

Their values are space specifiers that give a range of extra space to be added before and after the element. The actual spaces may be smaller or larger. Because the space is not part of the box itself, one box's end space can be part of the next box's start space.

A space specifier provides a range of values including a minimum, maximum, and optimum value. The formatter is free to pick from within this range to fit the constraints of the page. Furthermore, a space specifier includes values for precedence and conditionality. All five of these are separated by semicolons.

The precedence can either be an integer or the keyword *force*. The precedence determines what happens when the *space-end* of one inline area conflicts with the *space-start* of the next. The area with higher precedence wins. The default precedence is 0.

The conditionality is one of two keywords: *discard* or *retain*. These keywords determine what happens to extra space at the end of a line. The default is to discard it.

Contents Height and Width Properties

There are four properties that specify the height and width of the content area of a box as unsigned lengths. These are:

- ♦ `height`
- ♦ `width`
- ♦ `max-height`
- ♦ `max-width`

These do not specify the total width and height of the box, which also include the margins, padding, and borders. This is only the width and height of the content area. As well as an unsigned length, the `height` and `width` properties may be set to the keyword `auto`, which chooses the height and width based on the amount of content in the box. However, in no case are the height and width larger than the values specified by the `max-height` and `max-width` properties. For example:

```
<fo:block height="2in" width="2in">  
  Two strings walk into a bar...  
</fo:block>
```

The overflow Properties

The `overflow` property determines what happens if there's too much content to fit within a box of a specified size. This may be an explicit specification using the size properties or an implicit specification based on page size or other constraints. There are four possibilities, each represented by a keyword:

1. `auto`: use scrollbars if there is overflow; don't use them if there isn't
2. `hidden`: don't show any content that runs outside the box
3. `scroll`: attach scrollbars to the box so the reader can scroll to the additional content
4. `visible`: the complete contents are shown, if necessary, by overriding the size constraints on the box

The `clip` property specifies the shape of the clipping region if the `overflow` property does not have the value `visible`. The default clipping region is simply the box itself. However, you can change this by specifying a particular rectangle like this:

```
clip=rect(top_offset right_offset bottom_offset left_offset)
```

Here `top_offset`, `right_offset`, `bottom_offset`, and `left_offset` are signed lengths giving the offsets of the clipping region from the top, right, bottom, and left sides of the box. This allows you to make the clipping region smaller than the box itself.

The reference-orientation Property

The `reference-orientation` property enables you to specify that the content of a box is rotated in 90@@dg increments relative to its normal orientation. The only valid values are 90@@dg increments, which are measured counterclockwise, that is 0, 90, 180, and 270. You can also specify -90, -180, and -270. For example, here's a 90@@dg rotation:

```
<fo:block reference-orientation="90">
  Bottom to Top
</fo:block>
```

Writing Mode Properties

The writing mode specifies the direction of text in the box. This has important implications for the ordering of formatting objects in the box. Most of the time, speakers of English and other Western languages assume a left-to-right, top-to-bottom writing mode, such as this:

```
A B C D E F G
H I J K L M N
O P Q R S T U
V W X Y Z
```

However, in the Hebrew- and Arabic-speaking worlds, a right-to-left, top-to-bottom ordering such as this one seems more natural:

```
G F E D C B A
N M L K J I H
U T S R Q P O
Z Y X W V
```

In Taiwan, a top-to-bottom, right-to-left order is most comfortable:

```
A E I M Q U Y
B F J N R V Z
C G K O S W
D H L P T X
```

In the XSL formatting language, the writing mode doesn't just affect text. It also affects how objects in a flow or sequence are laid out, how wrapping is performed, and more. You've already noticed that many properties are organized in start, end, before, and after variations instead of left, right, top, and bottom. Specifying style rules in terms of start, end, before, and after, instead of left, right, top, and bottom, produces more robust, localizable style sheets.

The `writing-mode` property specifies the writing mode for an area. This property can have one of the same 14 keyword values. These are:

1. `bt-lr`: bottom-to-top, left-to-right

2. `bt-rl`: bottom-to-top, right-to-left
3. `lr-alternating-rl-bt`: left-to-right lines alternating with right-to-left lines, bottom-to-top
4. `lr-alternating-rl-tb`: left-to-right lines alternating with right-to-left lines, top-to-bottom
5. `lr-bt`: left-to-right, bottom-to-top
6. `lr-inverting-rl-bt`: left to right, then move up to the next line and go right to left (that is, snake up the page like a backward S)
7. `lr-inverting-rl-tb`: left to right, then move down to the next line and go right to left (that is, snake down the page like a backward S)
8. `lr-tb`: left to right, top to bottom
9. `rl-bt`: right to left, bottom to top
10. `rl-tb`: right to left, top to bottom
11. `tb-rl`: top to bottom, left to right
12. `tb-rl`: top to bottom, right to left
13. `tb-rl-in-rl-pairs`: top to bottom, right to left
14. `use-page-writing-mode`: whatever writing mode the page on which this object appears uses; the default

Orphans and Widows

To a typesetter, an orphan is a single line of a paragraph at the bottom of a page. A widow is a single line of a paragraph at the top of a page. Good typesetters move an extra line from the previous page or to the next page as necessary to avoid orphans and widows. You can adjust the number of lines considered to be an orphan by setting the `orphans` property to an unsigned integer. You can adjust the number of lines considered to be a widow by setting the `widows` property to an unsigned integer. For instance, if you want to make sure that every partial paragraph at the end of a page has at least three lines, set the `orphans` property to 3. For example:

```
<fo:simple-page-master page-master-name="even"
  orphans="3" page-height="8.5in" page-width="11in"
/>
```

Aural Properties

XSL supports the full collection of CSS2 aural stylesheet properties including:

- ♦ `azimuth`
- ♦ `cue`

- ♦ cue-after
- ♦ cue-before
- ♦ elevation
- ♦ pause
- ♦ pause-after
- ♦ pause-before
- ♦ pitch
- ♦ pitch-range
- ♦ play-during
- ♦ richness
- ♦ speak
- ♦ speak-header
- ♦ speak-numeral
- ♦ speak-punctuation
- ♦ speech-rate
- ♦ stress
- ♦ voice-family
- ♦ volume



The aural style sheet properties are discussed in the last section of Chapter 13, *Cascading Style Sheets Level 2*. They have exactly the same semantics and syntax in XSL formatting objects as they do in CSS2.

Summary

In this chapter, you learned about the XSL formatting language in detail. In particular, you learned:

- ♦ An XSL transformation is performed to turn an XML source document into a new XML document marked up in the XSL formatting object vocabulary.
- ♦ Most XSL formatting objects generate one or more rectangular areas. Page areas contain block areas. Block areas contain block areas and line areas. Line areas contain inline areas. Inline areas contain other inline areas and character areas.
- ♦ The root element of a formatting object document is `fo:root`. This contains `fo:layout-master-set` elements and `fo:page-sequence` elements.

- ♦ The `fo:layout-master-set` elements contain one or more `fo:simple-page-master` elements, each of which defines the layout of a particular kind of page by dividing it into five regions (before, after, start, end, and body), and assigning properties to each one.
- ♦ The `fo:page-sequence` elements contains one `fo:sequence-specifier` element, zero or more `fo:static-content` elements, and one `fo:flow` element. The contents of the `fo:flow` are copied onto instances of the master pages in the order specified by the `fo:sequence-specifier` element. The contents of the `fo:static-content` elements are copied onto every page that's created.
- ♦ The `fo:display-rule` element produces a block-level horizontal line. The `fo:inline-rule` element produces an inline horizontal line.
- ♦ The `fo:display-graphic` element loads an image from a URL and displays it in a block. The `fo:inline-graphic` element loads an image from a URL and displays it inline.
- ♦ The `fo:simple-link` element creates a hypertext link to a URL that's displayed in a block.
- ♦ A list is a block-level element created by a `fo:list-block` element. It contains block-level `fo:list-item` elements. Each `fo:list-item` contains a `fo:list-item-label` and `fo:list-item-body`.
- ♦ The `fo:page-number` element inserts the current page number.
- ♦ The `fo:character` element replaces a particular character or string of characters in the input with a different character in the output.
- ♦ The `fo:display-sequence` and `fo:inline-sequence` elements are containers used to attach properties to the text and areas they contain.
- ♦ The `fo:footnote` element inserts an out-of-line footnote and an inline footnote reference into the page.
- ♦ The `fo:float` element inserts an out-of-line block-level element like a figure or a pullquote onto the page.
- ♦ There are more than 200 separate XSL formatting properties, many of which are identical to CSS properties of the same name. These are attached to XSL formatting object elements as attributes.

The next chapter introduces XLinks, a more powerful linking syntax than the standard HTML A element hyperlinks or XSL's `fo:display-link` and `fo:inline-link`.



