For instance, the `border-top` property provides a width, style, and color for the top border. The `border-right`, `border-bottom`, and `border-left` properties are similar. Omitted properties are set to the value of the parent element. For example. Figure 12-30 shows a two-pixel solid blue border (a horizontal rule if you will) below each act. To achieve this, you would use this rule:

```
ACT { border-bottom: 2px solid blue }
```

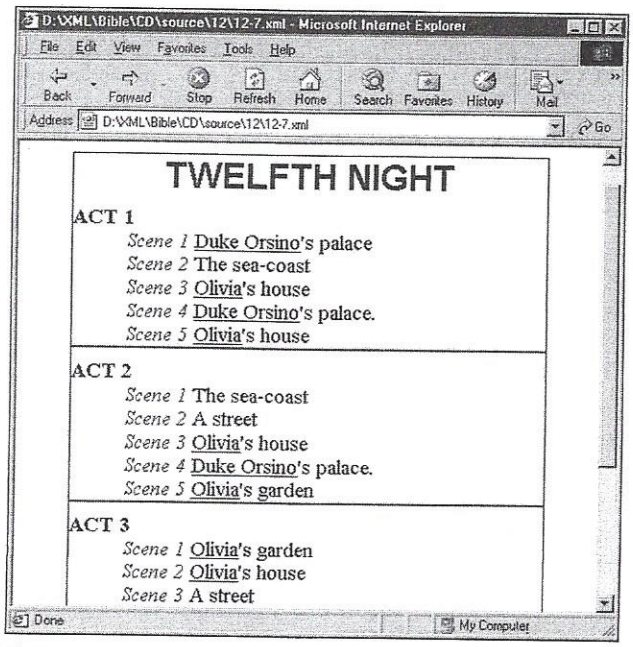


Figure 12-30: A two-pixel, solid bottom border is similar to HTML's HR element.

The `border` property sets all four sides to the specified width, style, and height. For example, this rule draws a three-pixel wide, solid, red border around a `CHART` element.

```
CHART { border: 3pt solid red }
```

Padding Properties

The padding properties specify the amount of space on the *inside* of the border of the box. The border of the box, if shown, falls between the margin and the padding. Padding may be set separately for the top, bottom, right and left padding using the

padding-top, padding-bottom, padding-right, and padding-left properties. Each padding may be given as an absolute length or be a percentage of the size of the parent element's width. For example, you can set off the SYNOPSIS from its border by setting its padding properties as shown in this rule.

```
SYNOPSIS { padding-bottom: 1em;
           padding-top: 1em;
           padding-right: 1em;
           padding-left: 1em }
```

You can also set all four at once using the shorthand padding property. For example, the following rule is the same as the previous one:

```
SYNOPSIS { padding: 1em 1em 1em 1em }
```

In fact, this is the same as using a single value for the padding property, which CSS interprets as applying to all four sides.

```
SYNOPSIS { padding: 1em }
```

Given two padding values, the first applies to top and bottom, the second to right and left. Given three padding values, the first applies to the top, the second to the right and left, and the third to the bottom. It's probably easier to use the separate padding-top, padding-bottom, padding-right, and padding-left properties.

The blue borders below the acts in the synopsis seem a little too close, so let's add an ex of padding between the end of the act and the border with the padding-bottom property, as shown in the following rule. Figure 12-31 shows the result. Generally, it's a good idea to use a little padding around borders to make the text easier to read.

```
ACT { padding-bottom: 1ex }
```

Size Properties

A box can be forced to a given size using the width and height properties. The contents of the box will be scaled as necessary to fit. Although you can use this with text boxes, it's more common and useful with replaced elements like images and applets. The width and the height may be given as an absolute length, as a percentage of the parent element's height and width, or as the keyword auto (the default) to indicate that the browser should use the real size. For example, this rule tries to fit the entire SYNOPSIS element in a 3-inch by 3-inch square.

```
SYNOPSIS { padding: 1em; width: 3in; height: 3in }
```

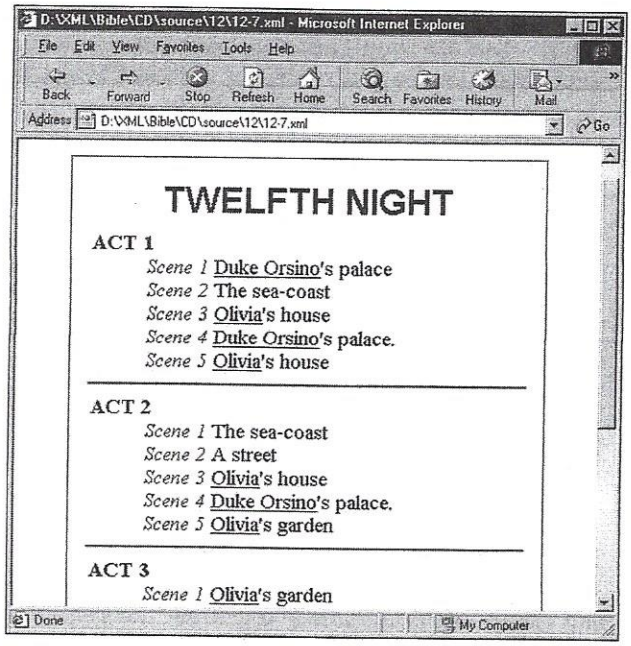


Figure 12-31: Padding makes borders easier on the eye

Figure 12-32 shows the result in Internet Explorer 5.0. When faced with an element that's simply bigger than its box allows, the Internet Explorer constrains the width but expands the height. Mozilla lets the text flow outside the box, possibly overlapping elements below. Browsers deal inconsistently and unpredictably with content that won't fit in a precisely sized box. Therefore, exact sizing is to be eschewed in cross-browser Web design.

If the `width` is set to an absolute or relative unit, and the `height` is set to `auto`, then the height will be adjusted proportionally to the width.

Positioning Properties

By default, block-level elements nested inside the same parent element follow each other on the page. They do not line up side by side or wrap around each other. You can change this with judicious use of the `float` and `clear` properties.

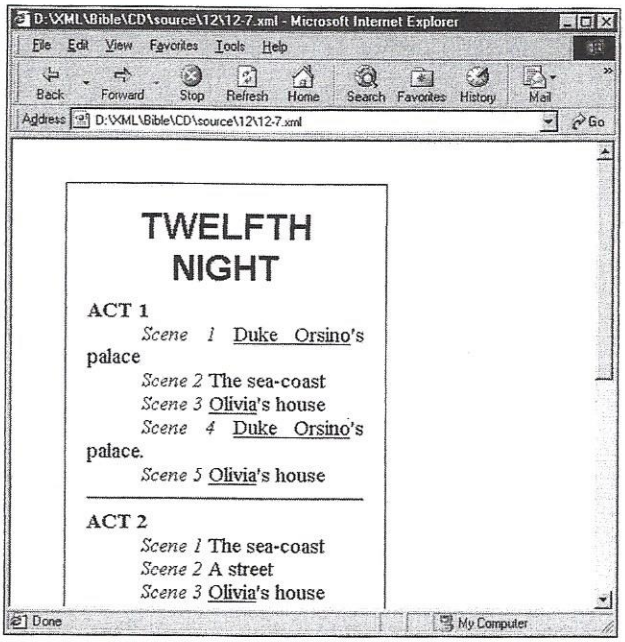


Figure 12-32: A three-inch high by three-inch wide synopsis as viewed in Mozilla

The float Property

The `float` property, whose value is `none` by default, can be set to `left` or `right`. If the value is `left`, then the element is moved to the left side of the page and the text flows around it on the right. In HTML, this is how an `IMG` with `ALIGN="LEFT"` behaves. If the value is `right`, then the element is moved to the right side of the page and the text flows around it on the left. In HTML, this is how an `IMG` with `ALIGN="RIGHT"` behaves.

There's no standard way to embed images in XML files, so for this example we'll fake it with a background image and some judicious use of CSS properties. Listing 12-16 is a slightly revised party invitation with an empty `IMAGE` element. Listing 12-17 is a style sheet that sets the party.gif file as the background for `IMAGE`. It also sets the width and height properties of `IMAGE`. Finally, it sets `float` to `left`. Figure 12-33 shows the result.

Listing 12-16: A party invitation with an empty IMAGE element

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="12-17.css"?>
<INVITATION>
  <IMAGE />
  <TEXT>
  You're invited to a party on December 31, 1999 to celebrate
  the new millennium! You're invited to a party on December 31,
  1999 to celebrate the new millennium! You're invited to a
  party on December 31, 1999 to celebrate the new millennium!
  You're invited to a party on December 31, 1999 to celebrate
  the new millennium! You're invited to a party on December 31,
  1999 to celebrate the new millennium! You're invited to a
  party on December 31, 1999 to celebrate the new millennium!
  You're invited to a party on December 31, 1999 to celebrate
  the new millennium! You're invited to a party on December 31,
  1999 to celebrate the new millennium! You're invited to a
  party on December 31, 1999 to celebrate the new millennium!
  You're invited to a party on December 31, 1999 to celebrate
  the new millennium! You're invited to a party on December 31,
  1999 to celebrate the new millennium!
  </TEXT>
</INVITATION>
```

Listing 12-17: A style sheet that loads an IMAGE

```
INVITATION { display:block; }
IMAGE { background: url(party.gif) no-repeat center center;
        width: 134px;
        height: 196px;
        float: left; }
TEXT { display: block }
```

The clear Property

The `clear` property specifies whether an element can have floating elements on its sides. If it cannot, the element will be moved below any floating elements that precede it. It's related to the HTML `<BR CLEAR="ALL">` element. The possible values are:

`none`	`right`
`left`	`both`

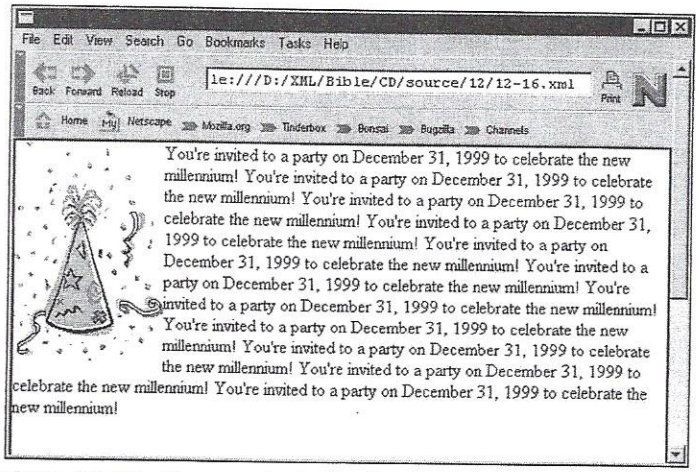


Figure 12-33: The party invitation image floating on the left

The default value, `none`, causes floating elements to appear on both sides of the element. The value `left` bans floating elements on the left-hand side of the element. The value `right` bans floating elements on the right-hand side of the element. The value `both` bans floating elements on the both sides of the element. For example, suppose you add this rule to the style sheet of Listing 12-17:

```
TEXT { clear: left }
```

Now, although the `IMAGE` element wants to float on the left of `TEXT`, `TEXT` doesn't allow that as is shown in Figure 12-34. `IMAGE` is still on the left, but now `TEXT` is pushed down below the image.

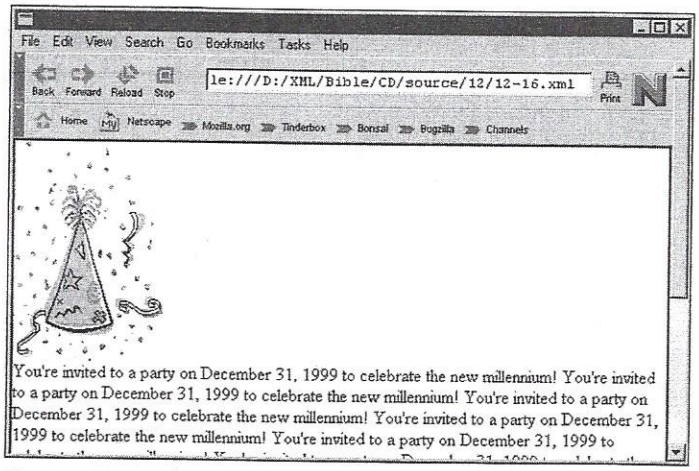


Figure 12-34: The party invitation image with the clear property set to left

Summary

In this chapter, you learned:

- ✦ CSS is a straightforward language for applying styles to the contents of elements that works well with HTML and even better with XML.
- ✦ Selectors are a comma-separated list of the elements a rule applies to.
- ✦ CSS can apply rules to elements of a given type or elements with particular `CLASS` or `ID` attributes.
- ✦ Many (though not all) CSS properties are inherited by the children of the elements they apply to.
- ✦ If multiple rules apply to a single element, then the formatting properties cascade in a sensible way.
- ✦ You can include C-like `/* */` comments in a CSS style sheet.
- ✦ Lengths can be specified in relative or absolute units. Relative units are preferred.
- ✦ The `display` property determines whether an element is block, inline, or a list item.
- ✦ Font properties determine the font face, style, size, and weight of text.
- ✦ Color of elements is given in a 24-bit RGB space in either decimal, hexadecimal, or as percentages.
- ✦ Background properties include color, image, image position, and image tiling.
- ✦ Text properties let you adjust line height, word spacing, letter spacing, vertical and horizontal alignment, decoration, and capitalization.
- ✦ Box properties let you adjust the relative positions and spacing of elements on the page, as well as wrapping borders around elements.

There are some limits to what CSS Level 1 can achieve. First, CSS1 can only attach styles to content that already appears in the document. It cannot add content to the document, even simple content like punctuation marks. Furthermore, it cannot transform the content in any way such as sorting or reordering it. These needs are addressed by XSL, the Extensible Style Language. Even from the perspective of merely formatting content, CSS1 offers less than what you want. Most glaringly, there's no support for tables. And there are other, less-obvious deficiencies. CSS1 cannot handle right-to-left text like Hebrew and Arabic or vertical text such as traditional Chinese. In the next chapter, we'll delve into CSS Level 2, which addresses these and other limitations of CSS1.

✦ ✦ ✦

CHAPTER 15

XSL Formatting Objects

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

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.

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.

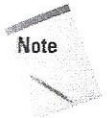
Formatting Objects and Their Properties

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:

- bidi-override
- block
- character
- display-graphic
- display-included-container
- display-rule
- display-sequence
- first-line-marker
- float
- flow
- footnote
- footnote-citation
- inline-graphic
- inline-included-container
- inline-rule
- inline-sequence
- layout-master-set
- list-block
- list-item
- list-item-body
- list-item-label
- multi-case
- multi-properties
- multi-property-set

- 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.



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:

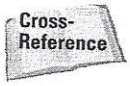
```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns:fo="http://www.w3.org/XSL/Format/1.0"
  result-ns="fo">
```

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:

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

- ✦ 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.

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 The `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.

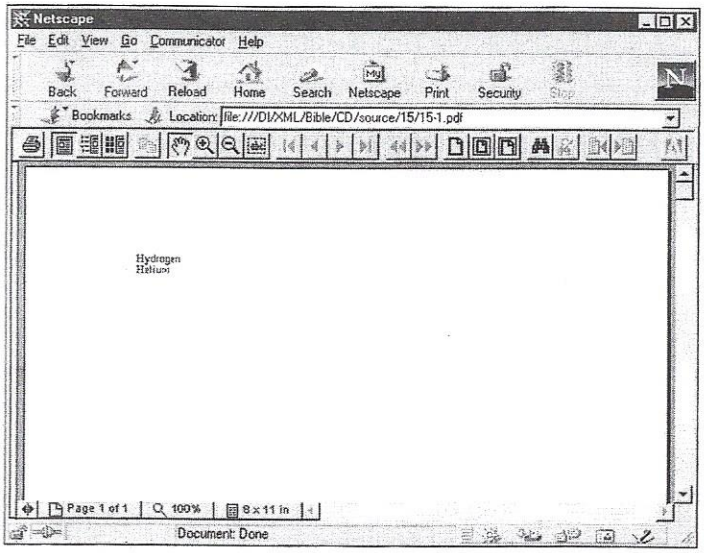


Figure 15-1: The PDF file displayed in Netscape Navigator

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.

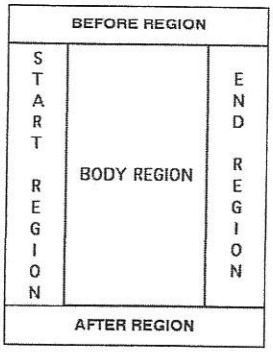
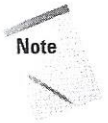


Figure 15-2: The layout of the parts of a simple page of English text



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 isthe 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.

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>
```

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

HTML Element	*XSL FO Element*
TABLE	`fo:table-and-caption`
no equivalent	`fo:table`
CAPTION	`fo:table-caption`
COL	`fo:table-column`
COLGROUP	no equivalent
THEAD	`fo:table-header`
TBODY	`fo:table-body`
TFOOT	`fo:table-footer`
TD	`fo:table-cell`
TR	`fo:table-row`

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>
```

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 a 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

Data Type	*Definition*	*Examples*
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