Making XML Look Good in Print

XSL-FO

O'REILLY®

Dave Pawson

XSL-FO

Dave Pawson

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

CHAPTER 6
Inline Elements

In this chapter, we will cover what is perhaps the simplest area of XSL-FO: styling the inline content. This is analogous to the word processor's application of bold or italics to particular words.

Inline content can be defined as content that, *when formatted*, does not extend beyond the formatted line extent, i.e., it does not wrap into a new line. Typical source content that may need marking for `fo:inline` might include content that needs to be emphasized for a specific purpose, such as emphasis, computer commands, instructions, and cross-references. The formatted output might be italicized, underlined, boldface, or hyperlinked. Other visual forms of emphasis include font changes and nontext output, such as inline graphics, horizontal lines, or dot leaders. These are all possible within `fo:inline`. It's sometimes difficult to decide between using `fo:block` and `fo:inline`. In such cases, if the content in question falls into a typeset line of content, use the `fo:inline` tag, otherwise, use `fo:block`.

A simpler view of an inline element is as a wrapper to apply style to phrases or individual words. A word of advice: if you use a style change, make a note of it and stick to it. If one specific font is used to represent a certain type of content, stick to it. Try the options out on sample input and find a scheme that is identifiable by the schema in use, and produces output that looks cohesive. A good example of this is provided in Donald E. Knuth's *The TEXbook* (Addison Wesley). Throughout the book, two symbols are used. The first symbol is similar to a bend roadsign; the other has two such symbols, referring to a dangerous bend! This simple scheme is used regularly and produces a nice visual reminder.

Content

The content model for `fo:inline` is rather loose, permitting both other inline elements and block elements. It makes sense to restrict content to `#PCDATA` plus other inlines elements, in most cases. I'll leave it to you to experiment. If you start out with this principle, you will break it, though usually for good reason, and emerge with greater understanding. I only ask that you consider why you are breaking it.

Inline Styling

Starting with the familiar, the stylesheet snippet of bullet 2 in Examples 6-1 and 6-2 demonstrates basic `fo:inline` usage.

Example 6-1. Inline example, XML source

```
<para>Some base content, containing an inline warning,
  <emphasis role="warning">Do not touch blue paper</emphasis>,
  a fairly straightforward piece requiring emphasis
  <emphasis>TEXT</emphasis>, and some instructions which
  require presenting in a different way, such as
  <instruction>Now light the blue paper</instruction>.
</para>
```

Example 6-2. Inline example, stylesheet snippet

```
     <xsl:template match="para">
❶     <fo:block>
          <xsl:apply-templates/>
        </fo:block>
      </xsl:template>

      <xsl:template match="emphasis[@role='warning']">
❷   <fo:inline background-color="red">Warning:</fo:inline>
          <xsl:apply-templates/>
        </xsl:template>

        <xsl:template match="emphasis[not(@role) or @role='']">
❸    <fo:inline font-weight="bold">
          <xsl:apply-templates/></fo:inline>
        </xsl:template>


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

❶ This is the containing block.

❷ The warning generates literal content using an inline.

❸ The simple `emphasis` tag generates bold content.

❹ The `instruction` tag generates italics.

This provides simple inline usage, probably the most common requirement, with the output as shown in Figure 6-1.

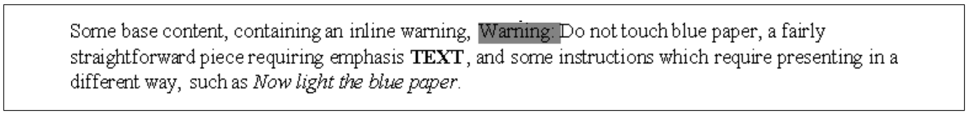

Some base content, containing an inline warning, Warning: Do not touch blue paper, a fairly straightforward piece requiring emphasis **TEXT**, and some instructions which require presenting in a different way, such as *Now light the blue paper*.

Figure 6-1. Inline output example

Other straightforward styles that may be applied in this manner are the `font-style` attributes of `normal`, `italic`, `oblique`, and `backslant`; the `font-weight` attributes, which split into relative values, are `normal`, `bold`, `bolder`, `lighter`, as well as the absolute values of 100, 200, 300, 400, 500, 600, 700, 800, and 900. Decoration is applied in the same way, using the `text-decoration` attribute of the `fo:inline` element.

The values for `text-decoration` are `underline`, `overline`, and `line-through`, which are the affirmative requests to the formatter, requesting a line under, above, or through the marked content, respectively. Each is effectively removed by the use of the `no-` prefix, so that an inline with mixed underline can be produced, as in Example 6-3. This nests inline content to (potentially) reduce the level of markup needed. Figure 6-2 shows the text decoration being switched on and off.

Example 6-3. Text decoration

```
<fo:block text-decoration="underline">Underline on for all
   but one <fo:inline text-decoration="no-underline">word</fo:inline>
   of the sentence.
</fo:block>
```

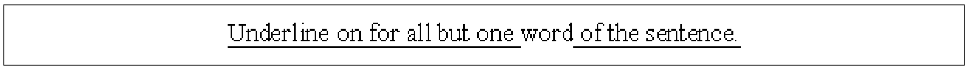

Underline on for all but one word of the sentence.

Figure 6-2. Text decoration being switched on and off

In Figure 6-2, the containing block has the `underline` property set; the contained inline turns it off for the single word.

The ability to select either the affirmative requirement (`underline`) or its inverse (`no-underline`) is more readily appreciated when transforming from XML. The utility of the specification becomes apparent only when you need it.

Other values for `text-decoration` are `overline` and `line-through`. This is very useful when marking up content for insertion and deletion in a manuscript. New content could be shown with either the background or the content colored and with content set for deletion shown as strike-through.

When decorating content, each case should be judged on its merits and future use. If you're designing stylesheets for a general purpose schema, it might be wise to allow for both cases, such that either may be applied. Which one you choose should be determined more by the case than by any rules. This is where your understanding of blocks and line layout will be tested.

text-shadow is available as an extended compliance option for text decoration; it is applicable to all elements, though it's most appropriate on inline content. It takes two length specifications and a color attribute. It is not widely implemented. The two lengths specify the horizontal and vertical offset, and the color specifies the color to be used for the shadow, as in Example 6-4. The first length is the horizontal offset from the text, the second, the vertical. Negative values indicate an offset left and up, positive values, down and right

Example 6-4. Text shadow effect

```
<fo:block>
          <fo:inline
             text-shadow="red 1mm 1mm">
          Text with a red shadow down and to the right by 1mm
          .</fo:inline>
</fo:block>
```

Inapplicable Properties

Certain properties appear (to me) to be largely inapplicable to inlines. I hate to make general rules, because it's all too likely that they will be broken, but it is a reasonable starting point. Borders are of less use inline than they are in blocks. Padding provides rather ugly whitespace around an inline and is best suited to blocks. Equally, breaks are best left to the blocks. This is not the way to terminate a page or column. Finally, whitespace preservation properties do not apply to inlines.

Inline Containers

fo:inline-container is available as an inline wrapper for content with a different writing mode to that of the bulk of the content. This matches the provision of a block with a change in writing mode, this time for inline content. A simple example is shown in Example 6-5. Note that the content of this element is a block.

Example 6-5. The use of fo:inline-container

```
<fo:block>
  <fo:inline-container writing-mode="rl-tb">
    <fo:block>
     Some text with writing mode st to rl-tb.</fo:block>
  </fo:inline-container>

</fo:block>
```

Inline Graphics

Sometimes the formatter may not provide what you want. A classic response to this has been the use of graphics as a replacement. For example, it has been common practice to insert mathematical expressions into HTML as graphics. Another formatter might not have a glyph in the font you wish to use. XSL-FO provides a means of including external graphics with `fo:external-graphic`, within the `fo:inline` element. This can be used to provide graphic content that has the appearance of normal inline content. A classic aspect of PDF that produces inaccessible content is the use of graphics to replace the first letter of a word. When exported to provide plain text, of course, the first letter is are omitted. Example 6-6 shows how you would obtain the graphic.

Example 6-6. An inline graphic

```
<fo:inline id="ls1">The main
      <fo:external-graphic src="url(images/image.png)"/> is ....
</fo:inline>
```

This form needs due care and attention because various facets tend to conspire against it. First, the resolution of the surrounding content is likely to be higher than that of the graphic. The graphic itself will need scaling and cropping to match the surrounding text. XSL-FO has a property that aids with this aspect, permitting use of two properties that size the graphic with respect to the font in use, using something such as Example 6-7.

Example 6-7. Scaled graphic

```
<fo:external graphic
content-height="1em"
content-width="1em"
src="url(images/image.png)"
/>
```

Other uses of graphics are discussed further in Chapter 7.

The `fo:instream-foreign-object` has developed quite well to permit the use of vector graphics, due to the efforts and needs of the FOP group, who embedded Scalable Vector Graphics (SVG) as a namespace-identified inclusion. This permits high-quality graphics, particularly line graphics, to be included as an integral element of a high-quality print document. You can read more about SVG on the W3C site or in *SVG Essentials* by David Eisenberg (O'Reilly).

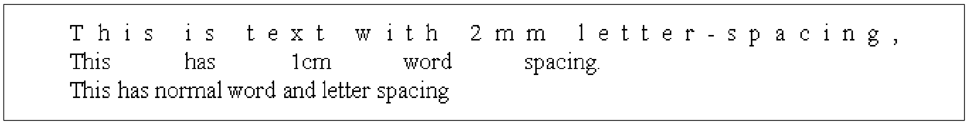
Word and Letter Spacing

Both word and letter spacing are tasks for the formatter. XSL-FO provides both. The `letter-spacing` property specifies spacing behavior between text characters. When a length is specified, the value indicates inter-character space, in addition to the default space between characters. Similarly, the `word-spacing` property indicates inter-word space, in addition to the default space between words. See Example 6-8.

Example 6-8. Letter spacing

```
<fo:inline
 letter-spacing="2mm">This is text with 2mm letter-spacing,</fo:inline>
<fo:inline word-spacing="1cm">  this has 1cm word spacing.</fo:inline>
```

Figure 6-3 is the result.



```
T h i s   i s   t e x t   w i t h   2 m m   l e t t e r - s p a c i n g ,
This      has      1cm      word      spacing.
This has normal word and letter spacing
```

Figure 6-3. Character spacing

Other Styling Properties

The `line-height` property can emphasize certain content within a large surrounding block of text, without other styling. This use of whitespace clearly outlines the content without otherwise distinguishing it. Note that if it is used within a block of content, the entire line will be laid out with the additional spacing. This is useful for content that does not stretch over a line boundary, after which point it simply looks strange. I find the percentage value most useful in this application, because it will adjust to any changes in surrounding font size, being a percentage of the font size itself.

The Horizontal Rule and Its Variants

The way a line is drawn in HTML is commonly known as the horizontal rule. XSL-FO provides a more subtle way of producing the same effect. The uses of the `fo:leader` element are generally decorative and might include breaks between sections of a book, signature lines, dot leaders in a table of contents, or text spacing. `fo:leader` is not allowed as a top-level element; it must be used within a block.

The basic, full-length line is shown in Example 6-9.

Example 6-9. Leaders for lines

```
    <fo:block>
❶   <fo:leader
❷     leader-length="100%"
❸     leader-pattern="rule"
❹     rule-style="solid"
❺     rule-thickness="0.1mm"  color="black"/>
    </fo:block>
```

❶ Wrapper element

❷ Length of the leader

❸ Its pattern
❹ Its style
❺ How thick and what color the result should be

The length of the resulting line is a standard length specification. I've chosen percentage here. Any form of length range can be used.

The pattern is specified using the `leader-pattern` attribute, which accepts the following options: `space`, `rule`, `dots`, `use-content`, and `inherit`. The pattern used in the example is the `rule` option. The style of the rule (`rule-style`) can be one of the following: `none`, `dotted`, `solid`, `double`, `groove`, `ridge`, or `inherit`. Each provides a variant decoration.

The thickness of the leader is specified using a length specification on the `rule-thickness` attribute. The example uses millimeters.

Note that the default for `leader-length` is 100% when a situation occurs where the width of the content area is determined by something other than the content itself. For example, when `text-align-last` is `justify`, the default `rule-style` is `solid`. In these circumstances, use the following to get a full-width rule:

```
<fo:block text-align-last="justify">
  <fo:leader leader-pattern="rule"/>
</fo:block>
```

A variation on this is to use two leader lines with a decorative character or graphic centered within the line. Figure 6-4 shows such an example, using a character from the Zapf family and two leader lines on either side, each half slightly less than 50% wide.

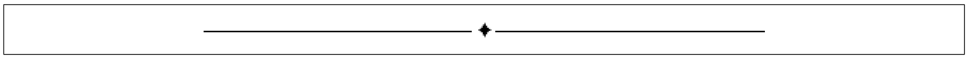


Figure 6-4. Decorative rule

The various options for decorative characters are nearly self-explantory. The pattern to be used is one from the selection `space`, `rule`, `dots`, or `use-content`. `space` uses the space character; `rule` uses a plain line; `dots` produces the dot leaders often used in lists; `use-content` produces a series of characters that are specified as actual content of the `fo:leader` element. Example 6-10 shows an example of this, using the character o.

Example 6-10. Leader pattern example

```
<fo:leader  leader-pattern="use-content" leader-length="60%">o</fo:leader>
```

The style of the rule is one from a selection list: `none`, `dotted`, `solid`, `double`, `dashed`, `groove`, or `ridge`. Each selects a particular style of line. Figure 6-5 shows a variety of these.

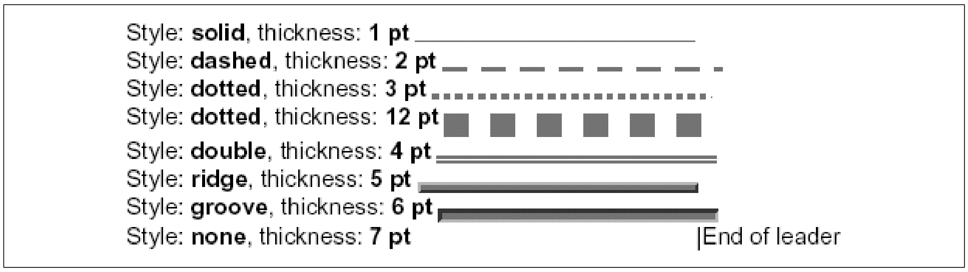

Style: solid, thickness: 1 pt
Style: dashed, thickness: 2 pt
Style: dotted, thickness: 3 pt
Style: dotted, thickness: 12 pt
Style: double, thickness: 4 pt
Style: ridge, thickness: 5 pt
Style: groove, thickness: 6 pt
Style: none, thickness: 7 pt
|End of leader

Figure 6-5. Line styles

The final use is the table of contents example shown in Example 6-11 and Figure 6-6.

Example 6-11. Table of contents usage

```
<fo:block
 text-align-last="justify">Chapter 10
   <fo:leader leader-pattern="dots" />Page 25</fo:block>
```

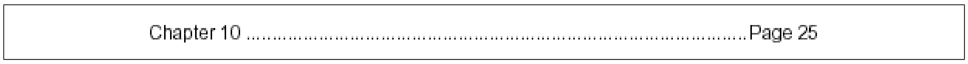

Chapter 10 .... Page 25

Figure 6-6. Table of contents usage

This demonstrates that inline styling may also be applied to blocks. This styling appears as a line of content, which is why it's in this chapter. Note the use of the `text-align-last` property, which ensures that the content is expanded to fill the available block width.

Another use of `fo:leader` is to produce a blank space on a form for someone to fill in. This makes use of the `leader-pattern="space"` and `style="none"` attribute settings, with an appropriate length specification, and produces a blank space in which the respondent can write. Note that if the font size is small, the `line-height` should be adjusted to ensure sufficient space is left to hand-write a response. Another option might be to use `leader-pattern="dots"` to provide a line on which to write, if the line is surrounded by whitespace.

Line Layout

Within any individual line, there may be layout issues requiring resolution. Some of these relate to block layout, such as those specific to first and last line layout; while others are specific to inline content. Other issues may be a case of appropriate selection. Typical of this last group are cross-references, footnotes and their references, and keeps and breaks.

Although covered in Chapter 9, I'll mention briefly the use of page numbers as typical inline content. The two aspects of this element, `fo:page-number` and `fo:page-number-citation`, are not obvious. `fo:page-number-citation` creates a page number reference. So when you want to talk about some material on another page, it becomes a case of referencing the remote content and/or the page number on which that content appears. The task of creating the page number is part of the formatting stage; the task of creating the reference to the section is part of the transformation stage. As in other such contexts, the use of ID values is a great help here. If you wish to reference Chapter 6 on page 34, or the title of that chapter, then if its ID value is known, it can be done easily in XSLT, using XSLT constructs and the `fo:page-number-citation` element from the XSL-FO syntax. The XSLT function `id(string)` takes a string value as a parameter, which is the ID needed, and returns the node-set that contains that ID value. This can then be used to obtain, for example, the title of the chapter using an XPath expression. The notable difference about `fo:page-number-citation` is that the ID value is not an ID value in the source document, but one in the intermediate document made up from the transform, such that the formatter has this information to determinine the page number on which this particular content is laid out. The implications of this are that for any content having an ID value in the source file, it is worthwhile generating an ID value on the derived block or inline. This is done easily with a simple, named template that adds the ID value to the block or inline only if the context node actually has an ID value. Such a template, and its call, are shown in Example 6-12.

Example 6-12. ID generation

```
<xsl:template match="p">
    <fo:block>
      <xsl:call-template name="gen-id">
        <xsl:with-param name="id-val" select="@id"/>
      </xsl:call-template>
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>

 <xsl:template name="gen-id">
    <xsl:param name="id-val" select="@id"/>
    <xsl:if test="$id-val">
      <xsl:attribute name="id"><xsl:value-of
         select="$id-val"/></xsl:attribute>
    </xsl:if>
  </xsl:template>
```

As an example of ID generation, consider the snippet of XML in Example 6-13.

Example 6-13. Cross-reference generation

```
<section id="sect1">
   <head>Introduction</head>
   <p>A plain paragraph</p>

   <p id="referenced-para">A paragraph which is referenced,
  hence has the id value.</p>
```

The section element is identifiable, and a particular paragraph is identified. Later on in the document, we may see something similar to Example 6-14.

Example 6-14. Source XML example

```
<p>See  <link target="sect1"/>
on page <pgref target="referenced-para"/>.</p>
```

Although single media, this shows the principles in use. The transformation requirement is to generate the `head` element contents in place of the `link` element and to replace the `pgref` element with the page number on which that particular paragraph is placed. The transform to execute this is shown in Example 6-15.

Example 6-15. Link usage

```
<xsl:template match='section'>
<fo:block id='{@id}'>
  <xsl:apply-templates/>
</fo:block>
</xsl:template>

<xsl:template match='para'>
  <fo:block id='{generate-id( )}'>
   <xsl:apply-templates/>
  </fo:block>
</xsl:template>

 <xsl:template match="link">
    <xsl:value-of select="id(@target)/head"/>
  </xsl:template>

 <xsl:template match="pgref">
   <fo:page-number-citation ref-id="{@target}"/>
  </xsl:template>
```

Note that because the ID value is copied across from the source XML document to the transformed document, we can use it as the target of `page-number-citation`.

Keeping Line Content Together

Some applications require you to keep a block of content within one line. The line wrapping options in XSL-FO will let you know about this, but it's up to the stylesheet author to manage it. The `keep` property `keep-together.within-line` attribute can be set to `always` to control wrapping. Note that if content cannot be fitted to the line, the `overflow` property can be used. If it is set to `error-if-overflow`, the formatter will report the error.

Other Uses

Some uses of inlines may not be obvious. For example, lists may be built using leaders with a fixed length to provide the spacing between the list marker and the list item content. Example 6-16 shows an example of this. There may be occasions when you simply need the line format that a list provides, but you don't want the block layout of a list.

Example 6-16. Leaders for lists

```
<fo:inline>
  <fo:character
  character="&#x2022;" font-family="ZapfDingbats"/> <fo:leader
        leader-pattern="space"
        leader-length="1.5cm"/> The list item contents</fo:inline>
```

Here, I have used the bullet character, U+2022, a fixed-length leader, and then the element content. Note that this is viable only for lists with short content because wrapping will not occur as in a proper list.

Page Headers

One of the requirements of a page header is often to contain two or three items. These might be the page number, the running header, and perhaps the book title. A number of options are available to do this, although the specification does not address this need directly. Unfortunately, one of these options involves implementation dependency. The end result of this is a loss of portability. The layout you test with one implementation may not work in another.

Use the `text-align-last` property of a block to stretch the block over the full line. The specification does not say where this stretching should take place (assuming you have content that can be stretched). To quote the recommendation, "The algorithm for resolving the adjusted values between word-spacing and letter-spacing is User Agent dependent." Thus, one implementation may stretch the space between words, and another may stretch the character spacing. Additionally, this is fraught with danger if the actual content is unknown and results in unequal content. Given these caveats, I have provided examples that can provide this layout. In Example 6-17, I have used leaders to provide the stretchable spaces. Try them out with your formatter, and be careful.

Example 6-17. Stretchable spaces for three area headers

```
<fo:block text-align-last="justify">
   <fo:inline> start1 </fo:inline>
   <fo:inline> center </fo:inline>
   <fo:inline> end </fo:inline>
</fo:block>
```

Example 6-17. Stretchable spaces for three area headers (continued)

```
<fo:block text-align-last="justify">
    <fo:inline letter-spacing="0pt" word-spacing="0pt"> start2 </fo:inline>
    <fo:inline letter-spacing="0pt" word-spacing="0pt"> center </fo:inline>
    <fo:inline letter-spacing="0pt" word-spacing="0pt"> end </fo:inline>
</fo:block>

<fo:block text-align-last="justify">
    <fo:inline> start3 </fo:inline>
    <fo:leader />
    <fo:inline> center </fo:inline>
    <fo:leader />
    <fo:inline> end </fo:inline>
</fo:block>

<fo:block text-align-last="justify">
    <fo:inline> start4 longer </fo:inline>
    <fo:leader />
    <fo:inline> center </fo:inline>
    <fo:leader />
    <fo:inline> end </fo:inline>
</fo:block>



<fo:list-block>
  <fo:list-item>
    <fo:list-item-label>
      <fo:block id="A" text-align="left">start5</fo:block>
    </fo:list-item-label>
    <fo:list-item-body>
      <fo:list-block>
        <fo:list-item>
          <fo:list-item-label>
            <fo:block id="B" text-align="center">Center</fo:block>
          </fo:list-item-label>
          <fo:list-item-body>
            <fo:block id="C" text-align="right">Right</fo:block>
          </fo:list-item-body>
        </fo:list-item>
      </fo:list-block>
    </fo:list-item-body>
  </fo:list-item>
</fo:list-block>
```

The first part of the example uses whitespace as the stretchable item; the second part uses word spacing—this guarantees that only spaces between inlines are expanded (inlines with explicitly specified `letter-spacing` and `word-spacing` are not subject to justification). The third part of the example uses a leader; the fourth part abuses the list structure. Note the impact of the longer start direction in the third part of the example on the centered area.

Stretching the block can also be achieved within an inline using two properties of inlines and leaders. First, the balanced spacing is achieved using a leader with its pattern set to space, and the inline element uses the text-align-last attribute set to justify. This spreads the three elements out over the inline giving the desired effect. Using fo:block to replace the fo:inline will provide the full page width, as might be used for a header or footer with content at the left, right, and center of the header. Example 6-18 shows how this spreading is accomplished.

Example 6-18. Header justification

```
<fo:inline text-align-last="justify">
    Left-hand text
    <fo:leader leader-pattern="space" />
     Centre Text using inlines
    <fo:leader leader-pattern="space" />
      Right-hand text
  </fo:inline>
```

This produces output shown in Figure 6-7. The width depends on the content.

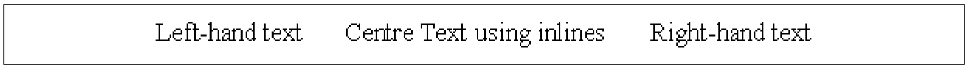


Figure 6-7. Header justification

An alternative, when the content of each header area is unequal, is to misuse the list. Misuse is probably too strong a term. One XSL Working Group member suggested a better title might be side-by-side formatting objects and provided Example 6-19, which shows a static-content example that enables unbalanced content to be nicely formatted in three areas, with all three correctly placed. This is written to be used as a callable template, with a parameter (listed here as a variable $header-width) for the actual header width.

Example 6-19. Lists in headers

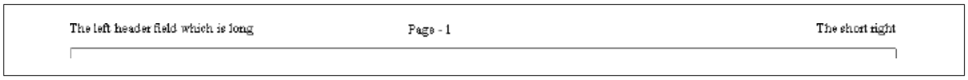
```
<xsl:template name='head1'>
<xsl:param name='header-width'>

<fo:static-content flow-name="xsl-region-before">
    <!-- header-width is the width of the full header in picas -->
    <xsl:variable name="header-width" select="36"/>
    <xsl:variable name="header-field-width">
    <xsl:value-of select="$header-width * 0.3333"/><xsl:text>pc</xsl:text>
    </xsl:variable>
    <fo:list-block font-size="8pt" provisional-label-separation="0pt">
        <xsl:attribute name="provisional-distance-between-starts">
            <xsl:value-of select="$header-field-width"/>
        </xsl:attribute>
```

Example 6-19. Lists in headers (continued)

```
        <fo:list-item>
            <fo:list-item-label end-indent="label-end( )">
                <fo:block text-align="left">
                    <xsl:text>The left header field which is long </xsl:text>
                </fo:block>
            </fo:list-item-label>
            <fo:list-item-body start-indent="body-start( )">
                <fo:list-block provisional-label-separation="0pt">
                    <xsl:attribute name="provisional-distance-between-starts">
                        <xsl:value-of select="$header-field-width"/>
                    </xsl:attribute>
                    <fo:list-item>
                        <fo:list-item-label end-indent="label-end( )">
                            <fo:block text-align="center">
                                Page - <fo:page-number/>
                            </fo:block>
                        </fo:list-item-label>
                        <fo:list-item-body start-indent="body-start( )">
                            <fo:block text-align="right">
                                <xsl:text>short right</xsl:text>
                            </fo:block>
                        </fo:list-item-body>
                    </fo:list-item>
                </fo:list-block>
            </fo:list-item-body>
        </fo:list-item>
    </fo:list-block>
</fo:static-content>
</xsl:template>
```

This produces the output shown in Figure 6-8, illustrating the balancing effect.



The left header field which is long   Page - 1   The short right

Figure 6-8. Header justification 2

Inlines are only presented here in terms of what is practical and what is likely to be needed. Many more properties are covered elsewhere in this book that can be used with inlines.

As stated earlier, the overlap between inlines and blocks is significant. Most of the properties available with inlines are also available with blocks, so use them as you need them.