Print and Page Layout Community Group

XSLT Extensions

The Print and Page Layout Community Group @ W3C (www.w3.org/community/ppl/) is open to all aspects of page layout theory and practice. It is free to join, and all are welcome at all levels of expertise.

The open-source XSLT extension functions (www.w3.org/community/ppl/wiki/XSLTExtensions) allow you to run your XSL-FO formatter within your XSLT transform to get an area tree, and to do it as often as you like, so the XSLT can make decisions based on formatted areas to do things like:

- Adjust the start-indent of a `fo:list-block` based on the length of the longest `fo:list-item-label` in the list; or
- Size this text to be 51.65771484375pt so it fits this box.

XSLT and XSL-FO Processors

The open-source extension is available for Java and DotNet and uses either the Apache FOP XSL formatter or Antenna House AHF formatter to produce the area trees.

A single Java jar file covers four combinations of XSLT processor and XSL-FO formatter:

- Saxon 9.5 and FOP
- Saxon 9.5 and Antenna House
- Xalan and FOP
- Xalan and Antenna House

The DotNet version supports:

- DotNet 4.0 and FOP
- DotNet 4.0 and Antenna House

API

The PPL CG provides `ppl:area-tree()` for running the formatter and getting the area tree plus a selection of convenience functions to help hide both the details of the area tree and the differences between the area trees of different XSL-FO formatters.

`ppl:area-tree($fo-tree as node()) as document-node()`
: Runs the XSL-FO formatter on $fo-tree to get an area tree.

`ppl:block-by-id($area-tree as document-node(), $id as string) as element()?`
: Returns the block area with ID $id.

`ppl:block-bpd($block as element()) as xs:double`
: Returns the block-progression-dimension of $block in points.

`ppl:block-bpd($block as element()) as xs:double`
: Returns the inline-progression-dimension of $block in points.

`ppl:block-available-ipd($block as element()) as xs:double`
: Returns the difference, in points, between the inline-progression-dimension of $block and the inline-progression-dimension of its ancestor reference area.

`ppl:is-first($block as element()) as xs:boolean`
: Returns the value of the `is-first` trait.

`ppl:is-last($block as element()) as xs:boolean`
: Returns the value of the `is-last` trait.

`ppl:sum-lengths-to-inches($lengths as xs:string*) as xs:double`
: Returns the length, in inches, of the sum of a sequence of lengths represented as strings, e.g., “6pt”, etc.

`ppl:sum-lengths-to-pt($lengths as xs:string*) as xs:double`
: Returns the length, in points (1/72 of an inch), of the sum of a sequence of lengths represented as strings, e.g., “6pt”, etc.

`bogus`
: Not a function, just an illustration of how, by using the extension functions to find the formatted size of the function definition, the `fo:list-item-body` moves down only for long function definitions. `troff` can do it, so why shouldn’t we?

XML

```
<example name="Balisage 2014">
<header>Print and Page Layout  <svg href="ppl-logo.svg" id="logo"/>
Community Group<lb/>XSLT Extensions</header>

<box id="box001" width="2240pt" height="520pt">
<paragraph id="para001">The Print and Page Layout Community Group @ W3C
(<url>www.w3.org/community/ppl/</url>) is open to all aspects of page
layout theory and practice.  It is free to join, and all are welcome at
all levels of expertise.</paragraph>
<paragraph>The open-source XSLT extension functions (<url>www.w3.org/
community/ppl/wiki/XSLTExtensions</url>) allow you to run your XSL-FO
formatter within your XSLT transform to get an area tree, and to do it as
often as you like, so the XSLT can make decisions based on formatted areas
to do things like:</paragraph>
<list id="list009">
  <item label="&#x2013;">Adjust the start-indent of a <code>fo:list-block</
code> based on the length of the longest <code>fo:list-item-label</code>
in the list; or</item>
  <item label="&#x2014;">Size this text to be <font-size/> so it fits this
box.</item>
</list>
</box>
<box id="box002" width="2240pt" height="690pt" columns="2" border="none">
<title>XSLT and XSL-FO Processors</title>
<paragraph id="para002">The open-source extension is available for Java
and DotNet and uses either the Apache FOP XSL formatter or Antenna House
AHF formatter to produce the area trees.</paragraph>
<paragraph>A single Java jar file covers four combinations of XSLT
processor and XSL-FO formatter:</paragraph>
<list id="list001">
  <item label="&#x2013;">Saxon 9.5 and FOP</item>
  <item label="&#x2013;">Saxon 9.5 and Antenna House</item>
  <item label="&#x2013;">Xalan and FOP</item>
  <item label="&#x2013;">Xalan and Antenna House</item>
</list>
<paragraph>The DotNet version supports:</paragraph>

<list id="list002">
  <item label="&#x2013;">DotNet 4.0 and FOP</item>
  <item label="&#x2014;">DotNet 4.0 and Antenna House</item>
</list>
<title>API</title>
<paragraph>The PPL CG provides <code>ppl:area-tree()</code> for running
the formatter and getting the area tree plus a selection of convenience
functions to help hide both the details of the area tree and the
differences between the area trees of different XSL-FO formatters.</
paragraph>
<api-list id="list003">
  <item>
    <function>ppl:area-tree($fo-tree as node()) as document-node()</
function>
    <def>Runs the XSL-FO formatter on $fo-tree to get an area tree.</def>
  </item>
  <item>
    <function>ppl:block-by-id($area-tree as document-node(), $id as
string) as element()?</function>
<def>Returns the block area with ID $id.</def>
  </item>
  <item>
    <function>ppl:block-bpd($block as element()) as xs:double</function>
    <def>Returns the block-progression-dimension of $block in points.</def>
  </item>
  <item>
    <function>ppl:block-bpd($block as element()) as xs:double</function>
    <def>Returns the inline-progression-dimension of $block in points.</
def>
  </item>
  <item>
    <function>ppl:block-available-ipd($block as element()) as xs:double</
function>
    <def>Returns the difference, in points, between the inline-progression-
dimension of $block and the inline-progression-dimension of its ancestor
reference area.</def>
  </item>
  <item>
    <function>ppl:is-first($block as element()) as xs:boolean</function>
    <def>Returns the value of the <code>is-first</code> trait.</def>
  </item>
  <item>
    <function>ppl:is-last($block as element()) as xs:boolean</function>
    <def>Returns the value of the <code>is-last</code> trait.</def>
  </item>
  <item>
    <function>ppl:sum-lengths-to-inches($lengths as xs:string*) as
xs:double</function>
    <def>Returns the length, in inches, of the sum of a sequence of
lengths represented as strings, e.g., &#x201C;<code>6pt</code>&#x201D;,
etc.</def>
  </item>
  <item>
    <function>ppl:sum-lengths-to-pt($lengths as xs:string*) as xs:double</
function>
    <def>Returns the length, in points (1/72 of an inch), of the sum of a
sequence of lengths represented as strings, e.g., &#x201C;<code>6pt</
code>&#x201D;, etc.</def>
  </item>
  <item>
    <function>bogus</function>
    <def>Not a function, just an illustration of how, by using the
extension functions to find the formatted size of the function definition,
the <code>fo:list-item-body</code> moves down only for long function
definitions.  <code>troff</code> can do it, so why shouldn&#x2019;t we?</
def>
  </item>
</api-list>
</box>
</example>
```

XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ============================================================= -->
<!-- PPL CG Balisage 2014 Poster                                   -->
<!--                                                               -->
<!-- See http://www.w3.org/community/ppl/wiki/XSLTExtensions       -->
<!--                                                               -->
<!-- Requires Saxon 9.5 or later and AHF 6.1 or later              -->
<!--                                                               -->
<!-- Produced by the Print and Page Layout Community Group @ W3C   -->
<!-- http://www.w3.org/community/ppl/                              -->
<!--                                                               -->
<!-- License: PUBLIC DOMAIN                                        -->
<!-- ============================================================= -->
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="2.0"
    xmlns:ppl="http://www.w3.org/community/ppl/ns/"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:axf="http://www.antennahouse.com/names/XSL/Extensions"
    exclude-result-prefixes="axf ppl xs">

<!-- ============================================================= -->
<!-- IMPORTS                                                       -->
<!-- ============================================================= -->

<!-- Print and Page Layout Community Group extensions. -->
<xsl:import href="ppl-extensions.xsl" />

<xsl:strip-space elements="example box" />

<!-- ============================================================= -->
<!-- KEYS                                                          -->
<!-- ============================================================= -->

<!-- 'override' elements used to override default property values. -->
<xsl:key name="overrides" match="override" use="@id" />

<!-- All 'box' elements in source. -->
<xsl:key name="boxes" match="box" use="@id"/>


<!-- ============================================================= -->
<!-- STYLESHEET PARAMETERS                                         -->
<!-- ============================================================= -->

<!-- Initial font size. -->
<xsl:param name="font-size" select="15" as="xs:double" />

<!-- Allowed difference in height between outer box and formatted
     paragraph text to be able to say paragraph fits within box and
     stop further iterations. -->
<xsl:param name="tolerance" select="1" as="xs:double" />

<!-- Difference between font-size.minimum and font-size.maximum below
     which it's not worth continuing. -->
<xsl:param name="font-size-tolerance" select="0.1" as="xs:double" />

<!-- Maximum number of iterations. -->
<xsl:param name="iteration-max" select="40" as="xs:integer"/>

<!-- Whether to use 'pygmentize' to highlight inserted markup. -->
<xsl:param name="pygmentize" select="'no'" as="xs:string" />


<!-- ============================================================= -->
<!-- STYLESHEET VARIABLES                                          -->
<!-- ============================================================= -->

<!-- Overrides of default property values. -->
<xsl:variable name="overrides">
  <no-overrides/>
</xsl:variable>

<!-- A0, portrait, page size. -->
<xsl:variable name="page-height" select="'1189mm'" as="xs:string" />
<xsl:variable name="page-width" select="'841mm'" as="xs:string" />

<!-- ============================================================= -->
<!-- ATTRIBUTE SETS                                                -->
<!-- ============================================================= -->

<xsl:attribute-set name="title">
  <xsl:attribute name="font-weight" select="'bold'" />
  <xsl:attribute name="font-size" select="'12em'" />
  <xsl:attribute name="keep-with-next" select="'always'" />
  <xsl:attribute name="space-before" select="'0.5em'" />
  <xsl:attribute name="space-after" select="'0.25em'" />
</xsl:attribute-set>

<xsl:attribute-set name="code">
  <xsl:attribute name="font-family" select="'monospace'" />
</xsl:attribute-set>

<xsl:attribute-set name="code-block" use-attribute-sets="code">
  <xsl:attribute name="padding" select="'6pt'" />
  <xsl:attribute name="space-before.conditionality" select="'retain'" />
  <xsl:attribute name="linefeed-treatment" select="'preserve'" />
  <xsl:attribute name="white-space-collapse" select="'false'" />
  <xsl:attribute name="white-space-treatment" select="'preserve'" />
  <xsl:attribute name="axf:line-number" select="'show'" />
  <xsl:attribute name="axf:line-number-color" select="'silver'" />
  <xsl:attribute name="axf:line-number-font-size" select="'1em'" />
  <xsl:attribute name="axf:line-number-font-family"
       select="'Calluna Sans, Verdana, sans-serif'" />
</xsl:attribute-set>

<!-- ============================================================= -->
<!-- INITIAL TEMPLATE                                              -->
<!-- ============================================================= -->

<xsl:template name="main">
  <xsl:message select="concat('tolerance: ', $tolerance)" />
  <xsl:message select="concat('font-size-tolerance: ', $font-size-
tolerance)" />

  <xsl:apply-templates select="/" mode="ppl">
    <xsl:with-param name="font-size"
                    select="$font-size" as="xs:double" tunnel="yes" />
    <xsl:with-param name="source" as="element()*">
      <fo:block-container
          font-size="9pt"
          axf:column-count="5"
          column-gap="48pt"
          space-before="30pt">
        <fo:block xsl:use-attribute-sets="title">XML</fo:block>
        <fo:block xsl:use-attribute-sets="code-block"
                  background-color="#f0f3f3">
          <xsl:sequence
              select="if ($pygmentize ne 'no')
                      then doc(concat(tokenize(base-uri(), '/')[last()],
'.fo'))
                      else unparsed-text(base-uri())" />
        </fo:block>
        <fo:block xsl:use-attribute-sets="title">XSLT</fo:block>
        <fo:block height="100%" xsl:use-attribute-sets="code-block"
                  background-color="#f0f3f3" display-align="justify">
          <xsl:sequence
              select="if ($pygmentize ne 'no')
                      then doc(concat(tokenize(base-uri(doc('')), '/')
[last()], '.fo'))
                      else unparsed-text('')" />
        </fo:block>
      </fo:block-container>
    </xsl:with-param>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="*" mode="ppl">
  <xsl:apply-templates select="." mode="#default" />
</xsl:template>

<xsl:template match="/*" mode="ppl">
  <xsl:apply-templates mode="#current" />
</xsl:template>

<xsl:template match="box" mode="ppl">
  <!-- 'do-box' templates recursively calls itself until: the
       formatted area fits within the box within $tolerance; the
       change in $font-size between successive runs is less than
       $font-size-tolerance; or $iteration-max iterations is
       reached. -->
  <xsl:call-template name="do-box">
    <xsl:with-param name="font-size" select="$font-size" as="xs:double"
tunnel="yes" />
    <xsl:with-param name="font-size.minimum"
                    select="$font-size" as="xs:double" tunnel="yes" />
    <xsl:with-param name="font-size.maximum"
                    select="$font-size * 6" as="xs:double" tunnel="yes" />
    <xsl:with-param name="iteration" select="1" as="xs:integer" />
    <xsl:with-param name="iteration-max"
                    select="$iteration-max" as="xs:integer" tunnel="yes" />
    <xsl:with-param name="tolerance"
                    select="$tolerance" as="xs:double" tunnel="yes" />
    <xsl:with-param name="font-size-tolerance"
                    select="$font-size-tolerance" as="xs:double"
tunnel="yes" />
  </xsl:call-template>
</xsl:template>

<xsl:template name="do-box">
  <xsl:param name="font-size" as="xs:double" tunnel="yes" />
  <xsl:param name="font-size.minimum"
             as="xs:double" tunnel="yes" />
  <xsl:param name="font-size.maximum"
             as="xs:double" tunnel="yes" />
  <xsl:param name="iteration"
             select="1" as="xs:integer" />
  <xsl:param name="iteration-max"
             select="5" as="xs:integer" tunnel="yes" />
  <xsl:param name="tolerance"
             select="$tolerance" as="xs:double" tunnel="yes" />
  <xsl:param name="font-size-tolerance"
             select="$font-size-tolerance"
             as="xs:double" tunnel="yes" />

  <xsl:message select="concat('iteration = ', $iteration)" />
  <xsl:message select="concat('font-size = ', $font-size)" />
  <xsl:message select="concat('font-size.minimum = ', $font-
size.minimum)" />
  <xsl:message select="concat('font-size.maximum = ', $font-
size.maximum)" />

  <xsl:variable name="overrides">
    <overrides>
      <!-- Set the font size. -->
      <xsl:for-each select="key('boxes', @id)">
        <xsl:variable name="id" select="@id" as="xs:string" />
        <override id="{$id}" font-size="{$font-size}" />
      </xsl:for-each>
    </overrides>
  </xsl:variable>

  <!-- Get just this box. -->
  <xsl:variable name="this-box">
    <xsl:sequence select="." />
  </xsl:variable>

  <!-- Save the FO tree for this box in a variable. -->
  <xsl:variable name="fo_tree">
    <xsl:apply-templates select="$this-box" mode="#default">
      <xsl:with-param name="overrides"
                      select="$overrides"
                      as="document-node()" tunnel="yes" />
    </xsl:apply-templates>
  </xsl:variable>

  <xsl:variable
      name="area-tree"
      select="ppl:area-tree($fo_tree)"
      as="document-node()?" />

  <xsl:variable name="block" select="ppl:block-by-id($area-tree, @id)"
        as="element()" />

  <xsl:variable name="bpd" select="ppl:block-bpd($block)" as="xs:double" />

  <xsl:variable name="target-height" select="ppl:length-to-pt(@height)"
        as="xs:double" />

  <xsl:message select="concat('bpd: ', $bpd)" />
  <xsl:message select="concat('target-height: ', $target-height)" />
  <xsl:choose>
    <xsl:when test="$target-height - $bpd > 0 and
                    $target-height - $bpd &lt; $tolerance and
                    ppl:is-last($block)">
      <xsl:message select="concat('It fits.  Using ', $font-size, '.')" />
      <xsl:apply-templates select="." mode="#default">
        <xsl:with-param
            name="overrides"
            select="$overrides"
            as="document-node()"
            tunnel="yes" />
      </xsl:apply-templates>
    </xsl:when>
    <xsl:when test="$font-size.maximum - $font-size.minimum &lt; $font-
size-tolerance and
                    $target-height - $bpd > 0 and ppl:is-last($block)">
      <xsl:message>Font size difference less than $font-size-tolerance.
Using <xsl:value-of select="$font-size" />.</xsl:message>
      <xsl:apply-templates select="." mode="#default">
        <xsl:with-param
            name="overrides" select="$overrides"
            as="document-node()" tunnel="yes" />
      </xsl:apply-templates>
    </xsl:when>
    <xsl:when test="$iteration eq $iteration-max">
      <xsl:message select="concat('>Maximum iterations.  Using ', $font-
size, '.')" />
      <xsl:apply-templates select="." mode="#default">
        <xsl:with-param
            name="overrides"
            select="$overrides"
            as="document-node()"
            tunnel="yes" />
      </xsl:apply-templates>
    </xsl:when>
    <xsl:when test="$bpd > $target-height or
                    not(ppl:is-last($block))">
      <xsl:message>$bpd gt $target-height or not last</xsl:message>
      <xsl:call-template name="do-box">
        <xsl:with-param
            name="font-size"
            select="($font-size + $font-size.minimum) div 2"
            as="xs:double"
            tunnel="yes" />
        <xsl:with-param
            name="font-size.maximum"
            select="$font-size"
            as="xs:double"
            tunnel="yes" />
        <xsl:with-param name="iteration"
                        select="$iteration + 1" as="xs:integer" />
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:message>$bpd lt $target-height</xsl:message>
      <xsl:call-template name="do-box">
        <xsl:with-param
            name="font-size"
            select="($font-size + $font-size.maximum) div 2"
            as="xs:double"
            tunnel="yes" />
        <xsl:with-param
            name="font-size.minimum"
            select="$font-size"
            as="xs:double"
            tunnel="yes" />
        <xsl:with-param name="iteration"
                        select="$iteration + 1" as="xs:integer" />
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="/" mode="#all">
  <xsl:param name="source" select="()" as="element()*" />

  <fo:root font-family="Calluna Sans, verdana, sans-serif" font-
size="18pt">
    <fo:layout-master-set>
      <fo:simple-page-master master-name="test-page">
        <xsl:if test="exists($page-height)">
          <xsl:attribute name="page-height" select="$page-height"/>
        </xsl:if>
        <xsl:if test="exists($page-width)">
          <xsl:attribute name="page-width" select="$page-width"/>
        </xsl:if>
        <fo:region-body margin="1in"/>
      </fo:simple-page-master>
    </fo:layout-master-set>

    <fo:page-sequence master-reference="test-page"
                      axf:line-number-start="5"
                      axf:line-number-interval="5"
                      axf:line-number-offset="0.375em + 6pt">
      <fo:flow flow-name="xsl-region-body">
        <xsl:apply-templates mode="#current" />
        <xsl:sequence select="$source" />
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>

<xsl:template match="header" mode="ppl">
  <xsl:variable name="header-font-size" select="20" as="xs:double" />

  <!-- Get a document containing just this element. -->
  <xsl:variable name="tmp-doc">
    <xsl:sequence select="." />
  </xsl:variable>

  <!-- Save the FO tree for this element in a variable. -->
  <xsl:variable name="fo_tree">
    <xsl:apply-templates select="$tmp-doc" mode="#default">
      <xsl:with-param name="overrides" as="document-node()" tunnel="yes">
        <xsl:document>
          <overrides>
            <override id="header" font-size="{$header-font-size}pt" />
          </overrides>
        </xsl:document>
      </xsl:with-param>
    </xsl:apply-templates>
  </xsl:variable>

  <xsl:variable name="area-tree" select="ppl:area-tree($fo_tree)"
                as="document-node()?" />

  <xsl:variable name="block" select="ppl:block-by-id($area-tree, 'header')"
                as="element()" />

  <xsl:variable name="ipd" select="ppl:block-ipd($block)" as="xs:double" />

  <xsl:variable name="available-ipd" select="ppl:block-available-
ipd($block)"
                as="xs:double" />

  <xsl:apply-templates select="." mode="#default">
    <xsl:with-param name="overrides" tunnel="yes">
      <xsl:document>
        <overrides>
          <override id="header"
                    font-size="{$header-font-size * $available-ipd div
$ipd}pt" />
        </overrides>
      </xsl:document>
    </xsl:with-param>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="header">
  <xsl:param name="overrides" select="$overrides"
             as="document-node()" tunnel="yes"/>
  <fo:block-container
      font-size="{$font-size}" font-size.minimum="{$font-size.minimum}"
      font-size.maximum="{$font-size.maximum}"
      font-size-tolerance="{$font-size-tolerance}"
      tolerance="{$tolerance}" iteration="{$iteration}"
      iteration-max="{$iteration-max}"
      font-weight="900" text-align="center" id="header">
    <fo:block
        line-height="1.2em"
        line-height-shift-adjustment="disregard-shifts">
      <xsl:apply-templates />
    </fo:block>
  </fo:block-container>
</xsl:template>

<xsl:template match="lb">
  <fo:block />
</xsl:template>

<xsl:template match="box">
  <xsl:param name="overrides" select="$overrides"
             as="document-node()" tunnel="yes"/>

  <fo:block-container
      role="{local-name()}" border="{(@border, 'medium solid black')[1]}"
      width="{@width}" height="{@height}" padding="12pt"
      id="{@id}" axf:column-count="{(@columns, 1)[1]}"
      column-gap="48pt" space-before="12pt">
    <xsl:if test="key('overrides', @id, $overrides)/@font-size">
      <xsl:attribute name="font-size"
                     select="concat(key('overrides', @id, $overrides)/
@font-size, 'pt')" />
    </xsl:if>
    <xsl:apply-templates/>
  </fo:block-container>
</xsl:template>

<xsl:template match="title">
  <fo:block xsl:use-attribute-sets="title">
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="paragraph">
  <fo:block id="{@id}" text-align="justify" space-before="3pt"
        text-indent="{if (preceding-sibling::*[1][self::paragraph])
                      then '1em' else '0pt'}">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="list">
  <xsl:param name="font-size" as="xs:double" tunnel="yes" />

  <!-- Get a document containing just this element. -->
  <xsl:variable name="tmp-doc">
    <box id="labels" font-size="{$font-size}">
      <xsl:for-each select="item/@label">
        <paragraph>
        <xsl:value-of select="." />
        </paragraph>
      </xsl:for-each>
    </box>
  </xsl:variable>

  <!-- Save the FO tree for the labels in a variable. -->
  <xsl:variable name="fo_tree">
    <xsl:apply-templates select="$tmp-doc" mode="#default">
      <xsl:with-param name="overrides" as="document-node()" tunnel="yes">
        <xsl:document>
          <overrides>
            <override id="labels" font-size="{$font-size}" />
          </overrides>
        </xsl:document>
      </xsl:with-param>
    </xsl:apply-templates>
  </xsl:variable>

  <xsl:variable name="area-tree" select="ppl:area-tree($fo_tree)"
                as="document-node()?" />

  <xsl:variable name="block" select="ppl:block-by-id($area-tree, 'labels')"
                as="element()" />

  <xsl:variable name="ipd" select="ppl:block-ipd($block)" as="xs:double" />

  <xsl:message select="concat('list labels ipd: ', $ipd)"/>

  <fo:list-block
      provisional-distance-between-starts="{$ipd}pt + 0.5em"
      provisional-label-separation="1em"
      space-before="0.25em"
      space-after="0.25em"
      id="{@id}">
    <xsl:apply-templates/>
  </fo:list-block>
</xsl:template>

<xsl:template match="item">
  <fo:list-item space-before="3pt" relative-align="baseline">
    <fo:list-item-label end-indent="label-end()">
      <fo:block color="red">
    <xsl:value-of select="@label"/>
      </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <fo:block>
        <xsl:apply-templates/>
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>

<xsl:template match="api-list">
  <fo:list-block
      provisional-distance-between-starts="35mm"
      provisional-label-separation="4pt"
      space-before="0.5em"
      id="{@id}">
    <xsl:apply-templates/>
  </fo:list-block>
</xsl:template>

<xsl:template match="api-list/item">
  <xsl:param name="font-size" as="xs:double" tunnel="yes" />

  <fo:list-item space-before="6pt" keep-together.within-column="always">
    <fo:list-item-label end-indent="0">
      <xsl:apply-templates select="function"/>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <!-- Get a document containing just the label. -->
      <xsl:variable name="tmp-doc">
        <box id="function" font-size="{$font-size}">
          <paragraph>
            <code>
              <xsl:value-of select="function" />
            </code>
          </paragraph>
        </box>
      </xsl:variable>

      <!-- Save the FO tree for the function in a variable. -->
      <xsl:variable name="fo_tree">
        <xsl:apply-templates select="$tmp-doc" mode="#default">
          <xsl:with-param name="overrides" as="document-node()"
tunnel="yes">
            <xsl:document>
              <overrides>
                <override id="function" font-size="{$font-size}" />
              </overrides>
            </xsl:document>
          </xsl:with-param>
        </xsl:apply-templates>
      </xsl:variable>

      <xsl:variable name="area-tree" select="ppl:area-tree($fo_tree)"
                    as="document-node()?" />

      <xsl:variable name="block" as="element()"
                    select="ppl:block-by-id($area-tree, 'function')" />

      <xsl:variable name="ipd" select="ppl:block-ipd($block)"
                    as="xs:double" />

      <xsl:message select="concat('function: ', function, '; ipd: ',
$ipd)"/>

      <xsl:if test="$ipd > ppl:length-to-pt('35mm')">
        <fo:block />
      </xsl:if>
      <fo:block space-before="1.2em" space-before.conditionality="retain">
        <xsl:apply-templates select="def" />
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>

<xsl:template match="item/function">
  <fo:block xsl:use-attribute-sets="code">
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="svg">
  <fo:instream-foreign-object content-height="1.8em"
                              alignment-adjust="-0.25em">
    <xsl:copy-of select="document(@href)"/>
  </fo:instream-foreign-object>
</xsl:template>

<xsl:template match="code | url">
  <fo:inline font-size="0.9em" xsl:use-attribute-sets="code">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>

<xsl:template match="font-size">
  <xsl:param name="font-size" as="xs:double" tunnel="yes" />

  <xsl:value-of select="$font-size"/>
  <xsl:text>pt</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

What can you do with an area tree?

Size text to fill available width

The text is formatted at a known size, then formatted “for real” with the font-size multiplied by the ratio of the available width to the formatted width

Print and Page Layout ppl Community Group XSLT Extensions

The Print and Page Layout Community Group @ W3C (`www.w3.org/community/ppl/`) is open to all aspects of page layout theory and practice. It is free to join, and all are welcome at all levels of expertise.

The XSLT extension functions (`www.w3.org/community/ppl/wiki/XSLTExtensions`) allow you to run your XSL-FO formatter within your XSLT transform to get an area tree, and to do it as often as you like, so the XSLT can make decisions based on formatted areas to do things like:

– Adjust the start-indent of a `fo:list-block` based on the length of the longest `fo:list-item-label` in the list; or

— Size this text to be 52.06970214843 75pt so it fits this box.

XSLT and XSL-FO Processors

The extension is available for Java and DotNet and uses either the Apache FOP XSL formatter or Antenna House AHF formatter to produce the area trees.

A single Java jar file covers four combinations of XSLT processor and XSL-FO formatter:

* Saxon 9.5 and FOP
† Saxon 9.5 and Antenna House
‡ Xalan and FOP
§ Xalan and Antenna House

The DotNet version supports:

– DotNet 4.0 and FOP
— DotNet 4.0 and Antenna House

API

The PPL CG provides `ppl:area-tree()` for running the formatter and getting the area tree plus a selection of convenience functions to help hide both the details of the area tree and the differences between the area trees of different XSL-FO formatters.

`ppl:area-tree($fo-tree as node()) as document-node()`
Runs the XSL-FO formatter on $fo-tree to get an area tree.

`ppl:block-by-id($area-tree as document-node(), $id as string) as element()?`
Returns the block area with ID $id.

`ppl:block-bpd($block as element()) as xs:double`
Returns the block-progression-dimension of $block in points.

`ppl:block-bpd($block as element()) as xs:double`
Returns the inline-progression-dimension of $block in points.

`ppl:block-available-ipd($block as element()) as xs:double`
Returns the difference, in points, between the inline-progression-dimension of $block and the inline-progression-dimension of its ancestor reference area.

`ppl:is-first($block as element()) as xs:boolean`
Returns the value of the `is-first` trait.

`ppl:is-last($block as element()) as xs:boolean`
Returns the value of the `is-last` trait.

`ppl:sum-lengths-to-inches($lengths as xs:string*) as xs:double`
Returns the length, in inches, of the sum of a sequence of lengths represented as strings, e.g., "`6pt`", etc.

`ppl:sum-lengths-to-pt($lengths as xs:string*) as xs:double`
Returns the length, in points (1/72 of an inch), of the sum of a sequence of lengths represented as strings, e.g., "`6pt`", etc.

`bogus` Not a function, just an illustration of how, by using the extension functions to find the formatted size of the function definition, the `fo:list-item-body` moves down only for long function definitions. `troff` can do it, so why shouldn't we?

XML

XSLT

What can you do with an area tree?

Size text to fill available height

‘do-box’ template recursively calls itself until: the formatted area fits within the box within $tolerance; the change in $font-size between successive runs is less than $font-size-tolerance; or $iteration-max iterations is reached.

Print and Page Layout ppl Community Group XSLT Extensions

The Print and Page Layout Community Group @ W3C (`www.w3.org/community/ppl/`) is open to all aspects of page layout theory and practice. It is free to join, and all are welcome at all levels of expertise.

The XSLT extension functions (`www.w3.org/community/ppl/wiki/XSLTExtensions`) allow you to run your XSL-FO formatter within your XSLT transform to get an area tree, and to do it as often as you like, so the XSLT can make decisions based on formatted areas to do things like:

- Adjust the start-indent of a `fo:list-block` based on the length of the longest `fo:list-item-label` in the list; or
- Size this text to be 52.0697021484375pt so it fits this box.

XSLT and XSL-FO Processors

The extension is available for Java and DotNet and uses either the Apache FOP XSL formatter or Antenna House AHF formatter to produce the area trees.

A single Java jar file covers four combinations of XSLT processor and XSL-FO formatter:

* Saxon 9.5 and FOP
† Saxon 9.5 and Antenna House
‡ Xalan and FOP
‖ Xalan and Antenna House

The DotNet version supports:

– DotNet 4.0 and FOP
— DotNet 4.0 and Antenna House

API

The PPL CG provides `ppl:area-tree()` for running the formatter and getting the area tree plus a selection of convenience functions to help hide both the details of the area tree and the differences between the area trees of different XSL-FO formatters.

`ppl:area-tree($fo-tree as node()) as document-node()`
Runs the XSL-FO formatter on $fo-tree to get an area tree.

`ppl:block-by-id($area-tree as document-node(), $id as string) as element()?`
Returns the block area with ID $id.

`ppl:block-bpd($block as element()) as xs:double`
Returns the block-progression-dimension of $block in points.

`ppl:block-bpd($block as element()) as xs:double`
Returns the inline-progression-dimension of $block in points.

`ppl:block-available-ipd($block as element()) as xs:double`
Returns the difference, in points, between the inline-progression-dimension of $block and the inline-progression-dimension of its ancestor reference area.

`ppl:is-first($block as element()) as xs:boolean`
Returns the value of the `is-first` trait.

`ppl:is-last($block as element()) as xs:boolean`
Returns the value of the `is-last` trait.

`ppl:sum-lengths-to-inches($lengths as xs:string*) as xs:double`
Returns the length, in inches, of the sum of a sequence of lengths represented as strings, e.g., "6pt", etc.

`ppl:sum-lengths-to-pt($lengths as xs:string*) as xs:double`
Returns the length, in points (1/72 of an inch), of the sum of a sequence of lengths represented as strings, e.g., "6pt", etc.

`bogus` Not a function, just an illustration of how, by using the extension functions to find the formatted size of the function definition, the `fo:list-item-body` moves down only for long function definitions. `troff` can do it, so why shouldn't we?

What can you do with an area tree?

Adjust fo:list-block start-indent

Just the list item labels are formatted, the longest length is found, and the provisional-distance-between-starts is set on the fo:list-block

Print and Page Layout ppl Community Group XSLT Extensions

The Print and Page Layout Community Group @ W3C (`www.w3.org/community/ppl/`) is open to all aspects of page layout theory and practice. It is free to join, and all are welcome at all levels of expertise.

The XSLT extension functions (`www.w3.org/community/ppl/wiki/XSLTExtensions`) allow you to run your XSL-FO formatter within your XSLT transform to get an area tree, and to do it as often as you like, so the XSLT can make decisions based on formatted areas to do things like:

- Adjust the start-indent of a `fo:list-block` based on the length of the longest `fo:list-item-label` in the list; or
- Size this text to be 52.0697021484375pt so it fits this box.

XSLT and XSL-FO Processors

The extension is available for Java and DotNet and uses either the Apache FOP XSL formatter or Antenna House AHF formatter to produce the area trees.

A single Java jar file covers four combinations of XSLT processor and XSL-FO formatter:

* Saxon 9.5 and FOP
† Saxon 9.5 and Antenna House
‡ Xalan and FOP
§ Xalan and Antenna House

The DotNet version supports:

– DotNet 4.0 and FOP
— DotNet 4.0 and Antenna House

API

The PPL CG provides `ppl:area-tree()` for running the formatter and getting the area tree plus a selection of convenience functions to help hide both the details of the area tree and the differences between the area trees of different XSL-FO formatters.

`ppl:area-tree($fo-tree as node()) as document-node()`
Runs the XSL-FO formatter on $fo-tree to get an area tree.

`ppl:block-by-id($area-tree as document-node(), $id as string) as element()?`
Returns the block area with ID $id.

`ppl:block-bpd($block as element()) as xs:double`
Returns the block-progression-dimension of $block in points.

`ppl:block-bpd($block as element()) as xs:double`
Returns the inline-progression-dimension of $block in points.

`ppl:block-available-ipd($block as element()) as xs:double`
Returns the difference, in points, between the inline-progression-dimension of $block and the inline-progression-dimension of its ancestor reference area.

`ppl:is-first($block as element()) as xs:boolean`
Returns the value of the `is-first` trait.

`ppl:is-last($block as element()) as xs:boolean`
Returns the value of the `is-last` trait.

`ppl:sum-lengths-to-inches($lengths as xs:string*) as xs:double`
Returns the length, in inches, of the sum of a sequence of lengths represented as strings, e.g., "6pt", etc.

`ppl:sum-lengths-to-pt($lengths as xs:string*) as xs:double`
Returns the length, in points (1/72 of an inch), of the sum of a sequence of lengths represented as strings, e.g., "6pt", etc.

`bogus` Not a function, just an illustration of how, by using the extension functions to find the formatted size of the function definition, the `fo:list-item-body` moves down only for long function definitions. `troff` can do it, so why shouldn't we?

What can you do with an area tree?

Make list item body avoid a long label

Each fo:list-item-body formats its fo:list-item-label and, if the fo:list-item-label is too long, adds space above itself to get out of the way

Print and Page Layout ppl Community Group XSLT Extensions

The Print and Page Layout Community Group @ W3C (`www.w3.org/community/ppl/`) is open to all aspects of page layout theory and practice. It is free to join, and all are welcome at all levels of expertise.

The XSLT extension functions (`www.w3.org/community/ppl/wiki/XSLTExtensions`) allow you to run your XSL-FO formatter within your XSLT transform to get an area tree, and to do it as often as you like, so the XSLT can make decisions based on formatted areas to do things like:

- Adjust the start-indent of a `fo:list-block` based on the length of the longest `fo:list-item-label` in the list; or
- Size this text to be 52.0697021484375pt so it fits this box.

XSLT and XSL-FO Processors

The extension is available for Java and DotNet and uses either the Apache FOP XSL formatter or Antenna House AHF formatter to produce the area trees.

A single Java jar file covers four combinations of XSLT processor and XSL-FO formatter:

* Saxon 9.5 and FOP
† Saxon 9.5 and Antenna House
‡ Xalan and FOP
§ Xalan and Antenna House

The DotNet version supports:

– DotNet 4.0 and FOP
— DotNet 4.0 and Antenna House

API

The PPL CG provides `ppl:area-tree()` for running the formatter and getting the area tree plus a selection of convenience functions to help hide both the details of the area tree and the differences between the area trees of different XSL-FO formatters.

`ppl:area-tree($fo-tree as node()) as document-node()`
Runs the XSL-FO formatter on $fo-tree to get an area tree.

`ppl:block-by-id($area-tree as document-node(), $id as string) as element()?`
Returns the block area with ID $id.

`ppl:block-bpd($block as element()) as xs:double`
Returns the block-progression-dimension of $block in points.

`ppl:block-bpd($block as element()) as xs:double`
Returns the inline-progression-dimension of $block in points.

`ppl:block-available-ipd($block as element()) as xs:double`
Returns the difference, in points, between the inline-progression-dimension of $block and the inline-progression-dimension of its ancestor reference area.

`ppl:is-first($block as element()) as xs:boolean`
Returns the value of the `is-first` trait.

`ppl:is-last($block as element()) as xs:boolean`
Returns the value of the `is-last` trait.

`ppl:sum-lengths-to-inches($lengths as xs:string*) as xs:double`
Returns the length, in inches, of the sum of a sequence of lengths represented as strings, e.g., "6pt", etc.

`ppl:sum-lengths-to-pt($lengths as xs:string*) as xs:double`
Returns the length, in points (1/72 of an inch), of the sum of a sequence of lengths represented as strings, e.g., "6pt", etc.

`bogus` Not a function, just an illustration of how, by using the extension functions to find the formatted size of the function definition, the `fo:list-item-body` moves down only for long function definitions. `troff` can do it, so why shouldn't we?