

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/pp1/) 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/pp1/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
 - Xalan 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 `pp1: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.

`pp1:area-tree($fo-tree as node()) as document-node()`
Runs the XSL-FO formatter on `$fo-tree` to get an area tree.

`pp1:block-by-id($area-tree as document-node(), $id as string) as element()?`
Returns the block area with ID `$id`.

`pp1:block-bpd($block as element()) as xs:double`
Returns the block-progression-dimension of `$block` in points.

`pp1:block-bpd($block as element()) as xs:double`
Returns the inline-progression-dimension of `$block` in points.

`pp1: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.

`pp1:is-first($block as element()) as xs:boolean`
Returns the value of the `is-first` trait.

`pp1:is-last($block as element()) as xs:boolean`
Returns the value of the `is-last` trait.

`pp1: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.

`pp1: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. `tr:off` 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/pp1/) 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/pp1/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 XSLT 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 pp1: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.

pp1:area-tree(\$fo-tree as node()) as document-node()
Runs the XSL-FO formatter on \$fo-tree to get an area tree.

pp1:block-by-id(\$area-tree as document-node(), \$id as string) as element()?
Returns the block area with ID \$id.

pp1:block-bpd(\$block as element()) as xs:double
Returns the block-progression-dimension of \$block in points.

pp1:block-bpd(\$block as element()) as xs:double
Returns the inline-progression-dimension of \$block in points.

pp1:block-available-ipt(\$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.

pp1:is-first(\$block as element()) as xs:boolean
Returns the value of the is-first trait.

pp1:is-last(\$block as element()) as xs:boolean
Returns the value of the is-last trait.

pp1: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.

pp1: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. trueff 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 Community Group XSLT Extensions

The Print and Page Layout Community Group @ W3C (www.w3.org/community/pp1/) 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/pp1/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 `pp1: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.

```
pp1:area-tree($fo-tree as nodes()) as document-nodes()
  Runs the XSL-FO formatter on $fo-tree to get an area tree.
```

```
pp1:block-by-id($area-tree as document-node(), $id as string) as element()?
  Returns the block area with ID $id.
```

```
pp1:block-bpd($block as element()) as xs:double
  Returns the block-progression-dimension of $block in points.
```

```
pp1:block-bpd($block as element()) as xs:double
  Returns the inline-progression-dimension of $block in points.
```

```
pp1:block-available-ipt($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.
```

```
pp1:is-first($block as element()) as xs:boolean
  Returns the value of the is-first trait.
```

```
pp1:is-last($block as element()) as xs:boolean
  Returns the value of the is-last trait.
```

```
pp1: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.
```

```
pp1: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. `error` can do it, so why shouldn't we?

