fosox: Generating XSL-FO in Python

John W. Shipman
2013-09-04 11:44

Abstract

Describes a module for the Python programming language to support generation of documents as Adobe PDF (Page Description Format) files. The module does this by generating XSL-FO (Extensible Stylesheet Language, Formatting Objects), which can be translated to PDF by various means.

This publication is available in Web form\(^1\) and also as a PDF document\(^2\). Please forward any comments to john@nmt.edu.

This work is licensed under a \(^3\)Creative Commons Attribution-NonCommercial 3.0 Unported License.

Table of Contents

1. A direct route to PDF ................................................................. 3
2. XSL-FO resources ................................................................. 4
3. Online files for this project ...................................................... 4
4. Structure of the generated XSL-FO ........................................... 5
5. Design notes ............................................................................ 8
   5.1. Dashing from camelCase .................................................. 9
6. The interface ............................................................................. 10
   6.1. class FoDim: General dimension ....................................... 10
   6.2. class Box: Define a rectangle ............................................ 11
   6.3. class MarginSet: Four margin dimensions ......................... 11
   6.4. class PageDims: Page layout ............................................ 12
   6.5. pageDimsFactory(): Some pre-built PageDims instances ...... 12
   6.6. root(): Generate the root element ..................................... 12
   6.7. LayoutMasterSet(): Start the layout-master-set element ...... 13
   6.8. simpleMaster(): Start a simple-page-master ..................... 13
   6.9. regionBody(): Generate a region-body ............................. 13
   6.10. regionBefore(): The region-before element .................... 14
   6.11. regionAfter(): The region-after region ......................... 14

\(^1\)http://www.nmt.edu/~shipman/soft/fosox/
\(^2\)http://www.nmt.edu/~shipman/soft/fosox/fosox.pdf
\(^3\)http://creativecommons.org/licenses/by-nc/3.0/
6.12. pageSequenceMaster(): Generate a page-sequence-master
6.13. singlePageMasterReference() ................................................................. 15
6.14. repeatablePageMasterReference() .......................................................... 15
6.15. repeatablePageMasterAlternatives() ....................................................... 15
6.16. conditionalPageMasterReference() ......................................................... 15
6.17. pageSequence() .................................................................................... 16
6.18. staticContent() ................................................................................... 17
6.19. flow() ................................................................................................. 17
6.20. block(): The basic block element ......................................................... 18
6.21. leader() .............................................................................................. 18
6.22. font() ................................................................................................. 18
7. fosoxtest: A small test driver ................................................................. 19
7.1. fosoxtest: Prologue .............................................................................. 19
7.2. fosoxtest: Imports ............................................................................... 19
7.3. fosoxtest: Manifest constants .............................................................. 19
7.4. fosoxtest: main() ............................................................................... 20
7.5. fosoxtest: buildMasters() ................................................................... 21
7.6. fosoxtest: buildStatic() ....................................................................... 23
7.7. fosoxtest: buildFlow() ......................................................................... 24
7.8. fosoxtest: Epilogue ............................................................................... 24
8. Implementation of fosox.py ................................................................. 24
9. Imports .................................................................................................. 25
10. Manifest constants ............................................................................... 25
10.1. FO_NAMESPACE .................................................................................. 25
10.2. The UNITS_ constants ...................................................................... 25
10.3. The PAPER sizes ............................................................................... 25
11. Specification functions ....................................................................... 26
12. root() .................................................................................................. 26
13. layoutMasterSet() ............................................................................... 27
14. simpleMaster() ................................................................................... 27
15. regionBody() ....................................................................................... 27
16. regionBefore() .................................................................................... 28
17. regionAfter() ...................................................................................... 28
18. pageSequenceMaster() ....................................................................... 28
19. singlePageMasterReference() ................................................................ 29
20. repeatablePageMasterReference() ....................................................... 29
21. repeatablePageMasterAlternatives() .................................................... 29
22. conditionalPageMasterReference() ..................................................... 30
23. pageSequence() ................................................................................... 30
24. staticContent() ................................................................................... 30
25. flow() ................................................................................................. 31
26. block() ............................................................................................... 31
27. class FoDim: A linear dimension ......................................................... 31
   27.1. FoDim. __init__() ........................................................................... 32
   27.2. FoDim. __str__() ........................................................................... 32
   27.3. FoDim. __add__() ......................................................................... 33
   27.4. FoDim. __sub__() ......................................................................... 33
   27.5. FoDim. __mul__() ......................................................................... 33
   27.6. FoDim. __div__() ......................................................................... 33
   27.7. FoDim. __neg__() ......................................................................... 34
   27.8. FoDim. convert(): Convert to different units .............................. 34
   27.9. FoDim. confactor(): Find a conversion factor ............................ 34
28. class Box: Define a rectangle ........................................................................................................ 35
  28.1. Box.__init__() ......................................................................................................................... 36
  28.2. Box.__str__() ......................................................................................................................... 36
29. class MarginSet ............................................................................................................................. 36
  29.1. MarginSet.__init__() ................................................................................................................. 37
  29.2. MarginSet.dict() ....................................................................................................................... 37
  29.3. MarginSet.__str__() .................................................................................................................... 37
30. class PageDims ................................................................................................................................ 37
  30.1. PageDims.__init__() ................................................................................................................. 38
  30.2. PageDims.__str__() .................................................................................................................... 38
31. pageDimsFactory() .......................................................................................................................... 38
32. letterPage() ................................................................................................................................... 39
33. letterLandPage() ............................................................................................................................ 39
34. legalPage() .................................................................................................................................... 39
35. indexCardPage() ............................................................................................................................. 40
36. paperTypeMap .................................................................................................................................. 40
37. font() .............................................................................................................................................. 41
38. leader() ........................................................................................................................................... 41
39. dash() ............................................................................................................................................. 41
40. deCamel(): Convert from camel-case to dashed attribute names .................................................. 42
41. confactest: Test driver for FoDim unit conversions ........................................................................ 43

1. A direct route to PDF

Adobe’s PDF (Portable Document Format)\(^4\) standard is a good way to represent a document in paged form. In this work we present a module named fo\(5\)ox, written in the Python programming language, that allows you to generate a PDF document.

The output of this module, named fo\(5\)ox, uses an XML document type called XSL-FO, for Extensible Stylesheet Language, Formatting Objects. A number of applications are available that can render an XSL-FO file into PDF; refer to the Wikipedia article on XSL-FO\(^6\) for more information.

In practice, most applications that generate XSL-FO use XSLT\(^7\) (Extensible Stylesheet Language, Transforms), a pure functional language that operates an input document in some XML format. However, this route requires that the input be in XML form.

The current document is intended to provide an alternative route to PDF that does not use XSLT, because XSLT operates only on XML input files. We want to provide a route for situations where the content comes from databases or other sources, to which XSLT has no direct interface.

The fo\(5\)ox module presented here allows you to generate XSL-FO directly from a Python program. For example, you might put up a Web page that queries a database and presents the result as a PDF file. For an example of just such an application, see the author’s Christmas Bird Count history report system\(^9\).

\(^4\)http://en.wikipedia.org/wiki/PDF
\(^5\)http://www.python.org/
\(^6\)http://en.wikipedia.org/wiki/XSL-FO
\(^7\)http://en.wikipedia.org/wiki/XSLT
\(^8\)http://en.wikipedia.org/wiki/XML
\(^9\)http://www.nmt.edu/~shipman/z/cbc/cbchist/
2. XSL-FO resources

In addition to a working knowledge of the Python language, you will need to understand how XSL-FO files are structured. The XSL-FO standard is quite large. You will not need to understand every single feature to generate documents, but you will certainly need access to reference material.

- The ultimate authority is the XSL-FO standard\(^1\). This standard is promulgated by the W3C\(^1\) organization.
- The conceptual model for page makeup is the same as the one used in CSS (Cascading Style Sheets)\(^1\), the W3C standard for formatting HTML. If you already understand concepts such as the box model with its padding, borders, and margins, you have that much less to learn on your way to mastering XSL-FO.
- The best single resource for understanding the structure of XSL-FO is this book. Disclaimer: the present author has no financial interest in this book.


- The `fosox` module uses the module described in the document *sox.py: Sequential Output of XML for Python*\(^1\). This module allows the direct generation of XML content using stream output, which minimizes memory usage.

Because XSL-FO files can be quite large, the author’s preferred route for direct XML generation, the `etbuilder` module described in *Python XML processing with lxml*\(^1\), would be impractical for the dynamic generation of large database query reports when available memory is limited.

3. Online files for this project

- `fosox.py`\(^1\): The actual Python module.
- `fosoxtest`: A small test driver.
- `fosoxtest.xml`: The XSL-FO file generated by `fosoxtest`.
- `fosoxtest.pdf`: The PDF output rendered from the output of `fosoxtest`.
- `confactest`: A test driver to exercise unit conversion functions.
- `confactest.out`: Output of `confactest`.
- The DocBook source\(^1\) for the present document.

\(^1\) [http://www.w3.org/TR/xsl11/](http://www.w3.org/TR/xsl11/)
\(^1\) [http://www.w3.org/](http://www.w3.org/)
\(^1\) [http://www.nmt.edu/~shipman/soft/sox/](http://www.nmt.edu/~shipman/soft/sox/)
\(^1\) [http://www.nmt.edu/~shipman/soft/pylxml/](http://www.nmt.edu/~shipman/soft/pylxml/)
\(^1\) [http://www.nmt.edu/~shipman/soft/fosox/fosox.py](http://www.nmt.edu/~shipman/soft/fosox/fosox.py)
\(^1\) [http://www.nmt.edu/~shipman/soft/fosox/fosoxtest](http://www.nmt.edu/~shipman/soft/fosox/fosoxtest)
\(^1\) [http://www.nmt.edu/~shipman/soft/fosox/fosoxtest.xml](http://www.nmt.edu/~shipman/soft/fosox/fosoxtest.xml)
\(^1\) [http://www.nmt.edu/~shipman/soft/fosox/fosoxtest.pdf](http://www.nmt.edu/~shipman/soft/fosox/fosoxtest.pdf)
\(^1\) [http://www.nmt.edu/~shipman/soft/fosox/confactest](http://www.nmt.edu/~shipman/soft/fosox/confactest)
\(^1\) [http://www.nmt.edu/~shipman/soft/fosox/confactest.xml](http://www.nmt.edu/~shipman/soft/fosox/confactest.xml)
\(^1\) [http://www.nmt.edu/~shipman/soft/fosox/confactest.pdf](http://www.nmt.edu/~shipman/soft/fosox/confactest.pdf)
\(^1\) [http://www.nmt.edu/~shipman/soft/fosox/confactest.out](http://www.nmt.edu/~shipman/soft/fosox/confactest.out)
\(^1\) [http://www.nmt.edu/~shipman/soft/fosox/fosox.xml](http://www.nmt.edu/~shipman/soft/fosox/fosox.xml)
4. Structure of the generated XSL-FO

The full XSL specification is truly a monster. The author once read the entire standard and handwrote 57 pages of notes on green engineering pads, five lines per inch. Fortunately, Pawson’s book (see Section 2, “XSL-FO resources” (p. 4)) greatly reduces the time it takes to get started producing quality documents.

As with most large packages, you probably won’t need 80% of what is there. Much of the complexity comes in when you are supporting multiple languages in the same document. The general problem XSL-FO solves is quite large. Consider the problems of formatting a page that includes English (where lines are written left to right, and the lines progress from top to bottom) and Japanese (each line is written top to bottom and lines progress right to left). This leads to the important concepts of Inline Progression Direction (IPD), which is “lr” (left-to-right) for English and “tb” (top-to-bottom) for Japanese; and Block Progression Direction (BPD), “tb” for English and “rl” for Japanese. In the balance of this document we will assume English (lr-tb) writing order.

Keep in mind also that the purpose of XSL-FO is to format documents within specific page sizes, which is basic to the PDF format. Web rendering is a completely different problem: for example, a Web page can use any width, although most people find horizontal scrolling quite annoying, and there are lots of pretty tiny screens out there.

The author’s preferred tool for understanding the structure of XML document types like XSL-FO is the Relax NG schema language, especially the RNC form (Relax NG Compact Format). See the NM Tech Computer Center’s publication, Relax NG Compact Syntax (RNC). RenderX, maker of XEP, our locally preferred XSL-FO-to-PDF translator, has kindly made available a complete RNC schema for XSL-FO.

When you are generating an XSL-FO file, the first problem is understanding the overall structure of the file. Here, in RNC format, is a highly reduced schema for the top-level elements of the document. This schema shows only a tiny part of the elements and attributes in the full schema; it is intended strictly to familiarize you with the main pieces of the document.

The root element of the tree is named root. It has two kinds of children. The layout-master-set child defines all the different ways that pages can be formatted. This is followed by one or more page-sequence elements describing the actual content that must be formatted into those page layouts.

```
start = root
root = element root { layout-master-set, page-sequence+ }

layout-master-set = element layout-master-set
                   { simple-page-master+ & page-sequence-master? }
```

Simple page masters divide the page up into five regions:

http://www.w3.org/TR/xsl11/
http://www.nmt.edu/~shipman/soft/rnc/
http://www.renderx.com/
http://www.renderx.com/tools/validators.html
The terminology here is relative to XSL-FO's generic directions: “before” and “after” refer to the BPD, and “start” and “end” are relative to the IPD.

Typically the “before” region is used for running headers; “after”, for running footers; and the content appears in the “body” region. Unless you are using the “start” and “end” regions for marginal notes, typically they will have zero width, and the body will have the same width as the headers and footers.

Each simple-page-master has a unique name that is defined by its master-name attribute. We won’t describe the five child elements here, but basically the region-body child describes the size of the text area and the margins around it; the region-before and region-after children describe the heights of the header and footer; and the region-start and region-end elements describe the widths of the marginal note columns.

Each of these five child elements has a region-name attribute that defines its name. The static-content element (described below) will refer to this name when it provides the content for a region (other than the body region).

```
simple-page-master = element simple-page-master
    { attribute master-name{text},
        region-body, region-before?, region-after?, region-start?,
        region-end? }
```

The page-sequence-master element is a container for the rules that dictate how text is assigned to the different simple page masters.

```
page-sequence-master = element page-sequence-master
    { (single-page-master-reference | repeatable-page-master-reference | repeatable-page-master-alternatives)+ }
```

- Use a single-page-master-reference for content that consists of exactly one page, such as a title page.
- Use a repeatable-page-master-reference for multiple pages of output that all use the same simple page master.
- Use a repeatable-page-master-reference element if you want to use different simple page masters in different cases (e.g., even or odd pages, first page, etc.).

In the first two of the above cases, the element has a master-reference attribute that points to the simple page master that has that master-name.
In the general case, where you have multiple simple page masters, use a `repeatable-page-master-alternatives` element. This element in turn is a container for `conditional-page-master-reference` elements, each of which describes one situation and names the simple page master to be used in that situation.

Within a `conditional-page-master-reference`:

- The `master-reference` attribute names the simple page master to be used when these conditions are true.
- If there is a `page-position` attribute, it selects whether this page master will be used only for the first page, only for the last page, or in the rest of the pages.
- If there is an `odd-or-even` attribute, it selects whether this page master is used for odd-numbered (recto) pages, even-numbered (verso) pages, or both.
- If there is a `blank-or-not-blank` attribute, it selects whether the master applies only to pages with no content, only to pages that do have any content, or both.

Here's an example that uses three simple masters: `first` for the first page, `odd` or `even` for the remaining pages.

```
<layout-master-set>
  <simple-page-master master-name="first">...
  <simple-page-master master-name="odd">...
  <simple-page-master master-name="even">...
  <page-sequence-master master-name="N">
    <repeatable-page-master-alternatives>
      <conditional-page-master-reference master-reference="first" page-position="first" />
      <conditional-page-master-reference master-reference="odd" page-position="rest" odd-or-even="odd" />
      <conditional-page-master-reference master-reference="even" page-position="rest" odd-or-even="even" />
    </repeatable-page-master-alternatives>
  </page-sequence-master>
</layout-master-set>
```

Returning to our schema, that completes the `layout-master-set` part of the document. The rest is in the other child of the root element, the `page-sequence` element.
Use a `static-content` element to specify the content of the non-body regions of the page such as headers and footers.

```xml
static-content = element static-content
  { attribute flow-name{text},
    block+ }
```

In the `static-content` element, use the `flow-name` attribute to specify where this static content goes. The value of this attribute is the region name you specified with the `region-name` attribute in the element that defined the format of that region. Typically you will supply one `block` element that defines the content to be displayed in that region. This content can include the `page-number` element, which will display the page number.

For example, suppose you have a running head, and in the `simple-page-master` for odd-numbered pages, your `region-before` element had an attribute `region-name="odd-head"`. To specify the content of the header, use a `static-content` element with a `flow-name="odd-head"` attribute.

The last element we'll discuss in this foreshortened grammar is the `flow` element.

```xml
flow = element flow {block+}
```

This element is the container for all the principal content of the document (excluding static content).

The content “`block+`” shown above is not technically correct. You may specify page body content using any number of block-level elements, including not only the actual `block` element but other elements such as tables, lists, or graphics.

This concludes our tour of the top-level structure of an XSL-FO file. The structure of the code you write that generates the file follows exactly the tree structure of the schema:

1. First you will instantiate class `sox.Sox`, the object that will manage the generation of the XML output stream. The output file name must end with “`.fo`”. In the rest of this procedure, we'll call the `Sox` instance `s`. (For more information on the `sox` module, see Section 2, “XSL-FO resources” (p. 4).)

2. You'll call `s.start('root')` to generate the `root` element. This method returns a token that you will need later to generate the closing tag at the end of the output file.

3. Generate the `layout-master-set` element and its substructure to describe your page layouts.

4. Generate the `page-sequence` element and its children, the various `static-content` children that define header and footer content, and then the `flow` element that has the actual body of the document.

5. Call the `.end()` method on the token you got back from the `s.start('root')` call to generate the final start tag, and close the output file with `s.close()`.

6. Do whatever you have to do on your local system to convert the `.fo` file to a `.pdf`.

## 5. Design notes

Before we discuss the specific features of the `fosox` package, let's discuss some of the underlying design concepts.

The `Sox` class used to generate XML as a stream is very efficient in its use of memory (see Section 2, “XSL-FO resources” (p. 4) for links to this package). However, it does impose one important constraint: you must generate the elements in document order. If the document is a simple table, that may not be a problem. In general, though, you may have to do some marshaling of the information into data structures that are amenable to sequential generation of the content.
Most of the functions provided by the fosox module require a Sox instance to start their element. All these modules will return a token (actually an instance of class sox.Elt) that you must later use to generate the closing tag by calling the token's .end() method.

5.1. Dashing from camelCase

This interface was strongly influenced by Fredrik Lundh's article An ElementTree builder\footnote{http://effbot.org/zone/element-builder.htm}. The author likes the way that the flexibility of Python calling sequences allows one to build many small XML structures in a single line of code.

One of the handy features of Lundh's package is that XML tag names and attributes that happened to be legal Python names could be used directly in the code. For example, once you have imported Lundh's magic E factory object, this code:

```python
defCamel('Title', id='x37')
```

is all you need to build this XHTML element:

```html
<h1 id='x37'>Title</h1>
```

However, the great majority of XSL-FO element and attribute names contain hyphens, and some attributes even contain periods (.)

Hence, the fosox module provides a number of features that allow you to substitute the equivalent name in camelCase form.

- For each sequence $aB$ in your name, where $a$ is a lowercase letter and $B$ is an uppercase letter, the package will substitute the sequence $a\_b$.
- For each underbar ("_") in your name, the package will substitute ".".

Here are some examples.

<table>
<thead>
<tr>
<th>Your name</th>
<th>XSL-FO name</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionBefore</td>
<td>region-before</td>
</tr>
<tr>
<td>simplePageMaster</td>
<td>simple-page-master</td>
</tr>
<tr>
<td>spaceBefore_minimum</td>
<td>space-before.minimum</td>
</tr>
</tbody>
</table>

The fosox module provides a large number of functions that generate specific XSL-FO elements, but there is no point in trying to provide a specific Python interface for every one of the hundreds of element types. For the ones not covered here, you can create them directly through the Sox instance.

The package provides these two functions that you will use often.

```python
defCamel(s)
```

For any string $s$, returns the “de-camel-cased” equivalent.

```python
>>> defCamel('spaceBefore_minimum')
'space-before.minimum'
```

```python
dash(**kw)
```

Given a set of keyword arguments, returns a single dictionary with the same set of values, but with each key run through the defCamel() function. The resulting dictionary is typically used to apply a set of attributes to an XSL-FO element.
6. The interface

Here is a complete list of the functions and classes provided by the fosox module, except for the de-Camel() and dash() functions previously discussed in Section 5.1, “Dashing from camelCase” (p. 9).

6.1. class FoDim: General dimension

Use an instance of this class to represent any width or height or other dimension. The class is aware of the various different units that are allowed in XSL-FO dimension attributes.

The class supports a few of the common arithmetic operations that one typically needs in computing various document dimensions.

Dimensions are stored internally using values of Python's Decimal (fixed-point) type. This type's fixed-point arithmetic avoids many of the common pathologies of the float type. For documentation on this type, see the The Python Standard Library.

The constructor has this calling sequence:

```python
FoDim(number, units)
```

The `number` may be any type that is acceptable to the `decimal.Decimal()` constructor.

The `units` argument may take any of these values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNITS_IN</td>
<td>&quot;in&quot;</td>
<td>Inches.</td>
</tr>
<tr>
<td>UNITS_MM</td>
<td>&quot;mm&quot;</td>
<td>Millimeters.</td>
</tr>
<tr>
<td>UNITS_CM</td>
<td>&quot;cm&quot;</td>
<td>Centimeters.</td>
</tr>
<tr>
<td>UNITS_PC</td>
<td>&quot;pc&quot;</td>
<td>Picas (12 points).</td>
</tr>
<tr>
<td>UNITS_PT</td>
<td>&quot;pt&quot;</td>
<td>Points (1/72.27&quot;)</td>
</tr>
</tbody>
</table>

Here are the methods defined on an instance of class FoDim.

```python
def __str__(self):
    Return self as a string suitable for use as an XSL-FO dimension attribute.

def __add__(self, other):
    Assuming other is another FoDim instance, returns their sum. If the units are different, the result will take into account standard unit conversions. For example:
```

```python
>>> import fohelpers as h
>>> print h.FoDim(3, h.UNITS_MM) + h.FoDim(6, h.UNITS_CM)
63mm
```

http://docs.python.org/library/decimal.html
. __sub__ (self, other)
   For another FoDim instance, returns their difference as a FoDim instance.

. __mul__ (self, other)
   Multiplies by a constant: other is any type acceptable to the decimal.Decimal() constructor.
   The result is a FoDim.

>>> import decimal as d
>>> print h.FoDim('2.5', h.UNITS_IN) * "1.5"
3.75in

. __div__ (self, other)
   Divides by a constant; other is any type acceptable to the decimal.Decimal() constructor.

. __neg__ (self)
   Returns self with the opposite sign.

. convert (newUnits)
   Returns a FoDim instance converted to newUnits, which takes the same values as the units argument of the FoDim constructor.

6.2. class Box: Define a rectangle
An instance of this class represents the dimensions of a rectangular area. The constructor:

Box (wide, high)

The arguments are the width and the height, respectively, as instances of class FoDim. The resulting instance has two attributes, .wide and .high, representing the arguments passed to the constructor.

6.3. class MarginSet: Four margin dimensions
The purpose of this class is to represent the four margin sizes of a page master. The constructor:

MarginSet (top=T, bot=B, left=L, right=R)

Each argument defines the width or height of one margin. Each argument defaults to zero. The values supplied for each argument may be either an instance of class FoDim or a string that is acceptable as an XSL-FO dimension, e.g., "3.6pc".

Instances of MarginSet have these attributes and methods:

.top
   The top margin height.

.bot
   The bottom margin height.

.left
   The left margin width.

.right
   The right margin width.
6.4. class PageDims: Page layout

Use an instance of this class to describe the overall dimensions of a page, as in a simple-page-master. Here is the constructor:

```python
PageDims(pageBox, pageMargins=None, frameMargins=None, bodyMargins=None)
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageBox</td>
<td>An instance of class Box that defines the paper size.</td>
</tr>
<tr>
<td>pageMargins</td>
<td>An instance of class MarginSet that describes the desired margins around the entire page. The default is zero margins all around.</td>
</tr>
<tr>
<td>frameMargins</td>
<td>Another instance of class MarginSet used to describe the heights of the page header (in the .top attribute) and the page footer (in .bot). The values of the .left and .right attributes of the argument are ignored. The default value is both zero.</td>
</tr>
<tr>
<td>bodyMargins</td>
<td>An instance of class MarginSet. The height between the header and the body is taken from its .top attribute, and the height between the body and footer is taken from its .bot attribute. The default is that both are zero.</td>
</tr>
</tbody>
</table>

The arguments to the constructor are available as instance attributes .pageBox, .pageMargins, .frameMargins, and .bodyMargins.

A few prefabricated PageDims instances are available; see Section 6.5, “pageDimsFactory(): Some pre-built PageDims instances” (p. 12).

6.5. pageDimsFactory(): Some pre-built PageDims instances

Use the pageDimsFactory() function to build a new instance of class PageDims in one of a few standard sizes. Here is the calling sequence:

```python
pageDimsFactory(paperType)
```

The paperType argument may be any of these values:

<table>
<thead>
<tr>
<th>paperType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPER_LETTER</td>
<td>Letter paper in portrait orientation, 8.5”×11”.</td>
</tr>
<tr>
<td>PAPER_LETTER_LAND</td>
<td>Letter paper in landscape orientation, 11”×8.5”.</td>
</tr>
<tr>
<td>PAPER_LEGAL</td>
<td>Letter paper, 8.5”×14”.</td>
</tr>
<tr>
<td>PAPER_3X5</td>
<td>A standard file card 3”×5” in portrait orientation.</td>
</tr>
</tbody>
</table>

6.6. root(): Generate the root element

```python
root(s)
```
An instance of class `sox.Sox` that manages the output XML stream. For links to the documentation for this class, see Section 2, “XSL-FO resources” (p. 4).

This function writes the opening `<root>` tag of the root element of the output XSL-FO file. The function returns a token, a `sox.Elt` instance, that you must use later to generate the closing tag. When all the document's content has been generated, be sure to call the `.end()` method of this token, and then close the `Sox` instance by calling its `.close()` method. If you don't perform both these steps, there is no guarantee that the file will have been written completely to the output file.

### 6.7. `layoutMasterSet()`: Start the layout-master-set element

```python
layoutMasterSet(s)
```

Starts a `layout-master-set` element using `Sox` instance `s`. Returns a `sox.Elt` instance representing the open element.

The content of this element will be generated by a call to Section 6.8, “`simpleMaster()`: Start a simple-page-master” (p. 13), followed by one or more calls to Section 6.12, “`pageSequenceMaster()`: Generate a page-sequence-master” (p. 14).

### 6.8. `simpleMaster()`: Start a simple-page-master

```python
simpleMaster(s, masterName, dims)
```

- `s` The `sox.Sox` instance to which the element is written.
- `masterName` The master name being defined; this must be a valid XML name.
- `dims` An instance of Section 6.4, “class `PageDims`: Page layout” (p. 12) that describes the various dimensions of the page.

This function returns a token representing the open element. Following the call, you will use calls to these functions to define the regions of the page master:

- Section 6.9, “`regionBody()`: Generate a region-body” (p. 13).
- Section 6.11, “`regionAfter()`: The region-after region” (p. 14).

Finally, call the `.end()` method of the token return by `simpleMaster()` to generate the closing tag.

### 6.9. `regionBody()`: Generate a region-body

```python
regionBody(s, dims)
```

- `s` A `sox.Sox` instance.
- `dims` An instance of Section 6.4, “class `PageDims`: Page layout” (p. 12) that defines the margins of the body region. For example, the `dims.pageBox` attribute will be used to define the paper height...
and width; the dims.pageMargins attribute will define the overall page margins; the frameMargins.top value will size the header region; and so forth.

This function does not return a value. It writes the complete region-body element.

6.10. regionBefore(): The region-before element

```python
regionBefore(s, regName, dims)
```

s
The output sox.Sox instance.

regName
The name you wish to apply to this region within this master.

dims
An instance of Section 6.4, “class PageDims: Page layout” (p. 12); its frameMargins.top attribute defines the height of the region-before.

The function does not return a value. It writes a complete region-before element.

6.11. regionAfter(): The region-after region

```python
regionAfter(s, regName, dims)
```

s
The output sox.Sox instance.

regName
The name you wish to apply to the region-after for the containing simple-page-master.

dims
An instance of Section 6.4, “class PageDims: Page layout” (p. 12). The frameMargins.bot attribute of this instance will determine the height of the footer region.

The regionAfter() function does not return a value. It writes the complete region-after element.

6.12. pageSequenceMaster(): Generate a page-sequence-master

```python
pageSequenceMaster(s, masterName)
```

s
The sox.Sox instance to which the content is to be written.

masterName
The name of the master you are defining; it must be a valid XML name. You will use this name later in the flow element to route the flow’s content to this master.

The function returns a token representing the open element. Call one or more of the following functions to generate the element or elements that specify which page master to use in which situation, and then call the .end() method on the token.

- Section 6.15, “repeatablePageMasterAlternatives()” (p. 15).
6.13. singlePageMasterReference()

```
singlePageMasterReference(s, masterRef)
```

- **s**
  
The `sox.SoX` instance to which the content will be written.

- **masterRef**
  
The name of the `simple-page-master` to be used for laying out the content.

This function does not return a value. It writes a complete `single-page-master-reference` element.

6.14. repeatablePageMasterReference()

```
repeatablePageMasterReference(s, masterRef, **kw)
```

- **s**
  
The `sox.SoX` instance to which output is to be written.

- **masterRef**
  
The name of the `simple-page-master` to be used for formatting output.

- **kw**
  
  A dictionary of attribute names and values to be attached to the generated `repeatable-page-master-reference` element.

This function does not return a value. It writes a complete element.

For example, assuming `s` is a `sox.SoX` instance, this code:

```
repeatablePageMasterReference(s, "many", dash(maximumRepeats='10'))
```

would produce this element:

```
<repeatable-page-master-reference master-reference='many' maximum-repeats='10'/>
```

6.15. repeatablePageMasterAlternatives()

```
repeatablePageMasterAlternatives(s)
```

This function writes the start tag of a `repeatable-page-master-alternatives` element, which is the `page-sequence-master` child you will use if you have a complex page layout (for instance, different margins on even and odd pages). It returns a `sox.Elt` token.

After calling this function, call Section 6.16, “conditionalPageMasterReference()” (p. 15) once for each rule that relates a page position to a `simple-page-master`. Then call the `.end()` method of the `sox.Elt` token to write the closing tag.

6.16. conditionalPageMasterReference()

```
conditionalPageMasterReference(s, masterRef, **kw)
```
The `sox.Sox` instance to which output is written.

**masterRef**

The name of the `simple-page-master` to be used for this page position.

**pagePosition=P**

Specifies on which page or pages of a sequence this simple page master is to be used. Values of `P` may be any of:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>Only on the first page of the sequence.</td>
</tr>
<tr>
<td>last</td>
<td>Only on the last page of the sequence.</td>
</tr>
<tr>
<td>rest</td>
<td>On pages that are neither the first nor the last of the sequence.</td>
</tr>
<tr>
<td>any</td>
<td>On all the pages of the sequence.</td>
</tr>
</tbody>
</table>

**oddOrEven=OE**

Specifies when this simple page master is used with respect to whether it is a recto (odd-numbered) or verso (even-numbered) page. Possible values of `OE`:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>odd</td>
<td>Only on the recto pages.</td>
</tr>
<tr>
<td>even</td>
<td>Only on the verso pages.</td>
</tr>
<tr>
<td>any</td>
<td>On recto or verso pages.</td>
</tr>
</tbody>
</table>

**blankOrNotBlank=B**

Specifies whether this simple page master is to be used with respect to whether there is any content on the page, or whether it is blank. Values may be any of these:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>Only on blank pages.</td>
</tr>
<tr>
<td>not-blank</td>
<td>Only on nonblank pages.</td>
</tr>
<tr>
<td>any</td>
<td>On blank or nonblank pages.</td>
</tr>
</tbody>
</table>

This function does not return a value; it writes a complete `conditional-page-master-reference` element. Here is an example:

```python
conditionalPageMasterReference(s, 'odd-master', oddOrEven='even')
```

The resulting element:

```xml
<conditional-page-master-reference master-reference='even-master' odd-or-even='even'/> 
```

### 6.17. `pageSequence()`

`pageSequence(s, masterRef)`

**s**

The `sox.Sox` instance to be used for output.

**masterRef**

Name of the `page-sequence-master` to be used in formatting the material in this page sequence.
This function returns a sox.Elt token representing the open element. Following this function call, carry out these steps:

1. Use calls to Section 6.18, “staticContent()” (p. 17) to define the static content (running headers and footers) for each of the simple page masters.
2. Call Section 6.19, “flow()” (p. 17) to generate the document’s actual content.
3. After all the static content and flow content have been added, call the .end() method of this token to produce the closing tag.

### 6.18. staticContent()

```python
staticContent(s, regName)
```

- **s**: A sox.Sox instance to which output will be written.
- **regName**: The name of the region to which this static content will be written; same as the `regName` value used in the call to Section 6.10, “regionBefore()”: The region-before element” (p. 14) or Section 6.11, “regionAfter()”: The region-after region” (p. 14) used to create the header or footer.

This function writes the start tag for a static-content element, and returns a sox.Elt token representing that open element.

After calling this function, use sox functions to generate the content of this header or footer enclosed in a block element. Then call the .end() method of the token.

Here are some suggestions for setting up header and footer content.

- Use the page-number element to get a folio (page number).
- The leader element is useful for producing either a fixed horizontal space or an area that can expand to fill any leftover area. Here is an example of a fixed half-inch space, as XSL-FO:

  ```xml
  <leader leader-length='0.5in' leader-pattern='space'/>
  ```

  To produce the second case, an element that expands to fill, leave out the `leader-length` attribute:

  ```xml
  <leader leader-pattern='space'/>
  ```

- If you want to stretch the header or footer content to the full page width, attach to the block element an attribute “text-align-last='justify'”. Make sure the header or footer also contains an expandable leader as described just above.

### 6.19. flow()

```python
flow(s, flowName)
```

- **s**: The sox.Sox instance that writes the output.
- **flowName**: For the main content, use `flowName='xsl-region-body'`. 
This function writes the start tag for a flow element, and returns a sox.Elt as a token of that open element.

Following the call to this function, generate the main content of the body. Allowable children of the flow element include block, block-container, table-and-caption, table, and list-block. Be sure to call the .end() method of the token for the flow element.

6.20. block(): The basic block element

```python
def block(s, *p, **kw):
```

This function starts a block element, and returns a sox.Elt token that you must use later to write the end tag. Between these operations you will generate the content of the block.

For a discussion of how positional and keyword arguments work in this function, refer to design-notes and also the reference documentation for sox.py: Sequential Output of XML for Python. The idea is to allow a number of different kinds of arguments that give you easy ways to generate elements, attributes, and content.

- String-type positional arguments are added as text content of the block element.
- Positional arguments of type dict are passed through the dash() function (described in Section 5.1, “Dashing from camelCase” (p. 9)) to convert camel-case keys to their XSL-FO equivalents.
- For each keyword argument, the name is also de-camel-cased and becomes an XSL-FO attribute.

6.21. leader()

The purpose of this function is to produce a leader element. This may appear as empty whitespace, as a row of dots, as a solid underline, or a number of other patterns.

```python
def leader(s, length=None, pattern=None):
```

- `s` The sox.Sox instance to which output will be written.
- `length` Specifies the length of the leader to be generated. If omitted, the leader will be able to stretch to any length. However, it will not be stretched unless other attributes force it to be stretched. For an example, see Section 6.18, “staticContent()” (p. 17).
- `pattern` May have many values; refer to the standard for the possibilities. Common values include space for whitespace; dots for equally-spaced dots; and rule for a horizontal ruled line. Default is space.

This function writes a complete element and does not return a value.

6.22. font()

```python
def font(**kw):
```

This function returns a dictionary of attributes that can be applied to a block or inline element to specify the text font characteristics. Attribute names are de-camelized as described in Section 5.1, “Dashing from camelCase” (p. 9). The six attribute names in their camel-case forms are:

- font-size
- font-style
- font-weight
- font-family
- font-variant
- font-stretch

Zoological Data Processing

fosox: Generating XSL-FO in Python

http://www.nmt.edu/~shipman/soft/sox/
7. fosoxtest: A small test driver

Here is a complete script that generates a small document with a complex page layout: different page layouts for odd and even pages, for double-sided printing.

This script uses the conventions of the Cleanroom software methodology; the comments in [ square brackets ] are Cleanroom intended functions. For more on this methodology, see the author's Cleanroom page.

7.1. fosoxtest: Prologue

The first lines make this script self-executing under Unix, and also refer the reader back to this documentation.

```python
#!/usr/bin/env python
#================================================================
# fosoxtest: Test driver for fosox.py module
# Do not edit this file directly. It is mechanically extracted
# from the documentation:
#   http://www.nmt.edu/~shipman/soft/fosox/
#----------------------------------------------------------------
```

7.2. fosoxtest: Imports

Throughout this script, s will be the sox.Sox instance used to route the XSL-FO output to the standard output stream, and f will be the fosox module.

```python
# - - - - - I m p o r t s
import sox
import fosox as f
```

7.3. fosoxtest: Manifest constants

```python
# - - - - - M a n i f e s t c o n s t a n t s
```

http://www.nmt.edu/~shipman/soft/clean/
For this application we'll need two simple-page-master elements, one for odd pages and one for even pages. We'll also define the name for the page-sequence-master element.

```bash
# Names of the simple master and the repeatable master

ODD_MASTER = "odd"
EVEN_MASTER = "even"
REPEAT_MASTER = "all"
```

Next we'll define the names for the header and footer regions on those pages.

```bash
# Region names

ODD_BEFORE = "odd-before"
ODD_AFTER = "odd-after"
EVEN_BEFORE = "even-before"
EVEN_AFTER = "even-after"
```

Next we'll use Section 6.5, “pageDimsFactory(): Some pre-built PageDims instances” (p. 12) to produce two instances of the PageDims class to contain the layout parameters for the odd and even pages. Each starts with the basic letter paper layout; then we add a more generous left margin on odd pages and a more generous right margin on even pages, for the binding.

```bash
# Page dimensions

ODD_DIMS = f.pageDimsFactory(f.PAPER_LETTER)
ODD_DIMS.pageMargins.left = "1.5in"
EVEN_DIMS = f.pageDimsFactory(f.PAPER_LETTER)
EVEN_DIMS.pageMargins.right = "1.5in"
```

Next we define the main font family and then three fonts of this family for the header, footer, and body faces.

```bash
# Fonts

MAIN_FAMILY = "Palatino, Paladino, serif"
HEAD_FONT = f.font(fontFamily=MAIN_FAMILY, fontSize="10pt")
FOOT_FONT = f.font(fontFamily=MAIN_FAMILY, fontSize="10pt", fontStyle="italic")
BODY_FONT = f.font(fontFamily=MAIN_FAMILY, fontSize="12pt")
```

7.4. fosoxtest: main()

```bash
# main

def main():
```

fosox: Generating XSL-FO in Python

Zoological Data Processing
"""Main program.
"""

#-- 1
# [ s := a new sox.Sox instance that writes to sys.stdout ]
s = sox.Sox()

#-- 2
# [ s += an open root element
# root := a sox.Elt instance representing that element ]
root = f.root(s)

See Section 7.5, “fosoxtest: buildMasters()” (p. 21).

#-- 3
# [ s += a layout-master-set element named REPEAT_MASTER
# for double-sided output with odd-page master ODD_MASTER
# and even-page master EVEN_MASTER ]
buildMasters(s)

#-- 4
# [ s += an open page-sequence element for REPEAT_MASTER
# pages := a sox.Elt instance representing that element ]
pages = f.pageSequence(s, REPEAT_MASTER)

See Section 7.6, “fosoxtest: buildStatic()” (p. 23) and Section 7.7, “fosoxtest: buildFlow()” (p. 24). After the content has all been generated, close the open page-sequence and root elements and we’re done.

#-- 5
# [ s += static-content elements that set up headers in
# regions ODD_BEFORE and EVEN_BEFORE and footers in
# regions ODD_AFTER and EVEN_AFTER ]
buildStatic(s)

#-- 6
# [ s += a flow element containing some sample content ]
buildFlow(s)

#-- 7
# [ s += (end tag for pages) + (end tag for root) ]
pages.end()
root.end()
s.cleanup()

7.5. **fosoxtest: buildMasters()**

This function builds the layout-master-set child of the root element, containing the page layouts and the rules that specify which one gets used for which kind of page. The first order of business is to start the layout-master-set element.

#-- buildMasters
def buildMasters(s):
    '''Set up all page layout.'''
    #-- 1
    # [ s += a new, open layout-master-set element
    # layout := a sox.Elt instance representing that element ]
    layout = f.layoutMasterSet(s)

    The first simple-page-master is for odd-numbered pages. It contains a region-body element, a region-before element for the header, and a region-after element for the footer. The second simple-page-master has the same structure.

    #-- 2
    # [ s += a simple-page-master named ODD_MASTER with
    #   a body of size ODD_DIMS, a header region named
    #   ODD_BEFORE sized by ODD_DIMS, and a footer region
    #   named ODD_AFTER sized by ODD_DIMS ]
    oddMaster = f.simpleMaster(s, ODD_MASTER, ODD_DIMS)
    f.regionBody(s, ODD_DIMS)
    f.regionBefore(s, ODD_BEFORE, ODD_DIMS)
    f.regionAfter(s, ODD_AFTER, ODD_DIMS)
    oddMaster.end()

    #-- 3
    # [ s += a simple-page-master named EVEN_MASTER with
    #   a body of size EVEN_DIMS, a header region named
    #   EVEN_BEFORE sized by EVEN_DIMS, and a footer region
    #   named EVEN_AFTER sized by EVEN_DIMS ]
    evenMaster = f.simpleMaster(s, EVEN_MASTER, EVEN_DIMS)
    f.regionBody(s, EVEN_DIMS)
    f.regionBefore(s, EVEN_BEFORE, EVEN_DIMS)
    f.regionAfter(s, EVEN_AFTER, EVEN_DIMS)
    evenMaster.end()

    The second child of the layout-master-set is the page-sequence-master element, which contains a repeatable-page-master-alternatives element, which in turn contains two conditional-page-master-reference elements that specify the rules for odd and even page formats.

    #-- 4
    # [ s += a page-sequence-master element named REPEAT_MASTER,
    #   selecting ODD_MASTER for odd pages and EVEN_MASTER for
    #   even pages ]
    seqMaster = f.pageSequenceMaster(s, REPEAT_MASTER)
    masterAlts = f.repeatablePageMasterAlternatives(s)
    f.conditionalPageMasterReference(s, ODD_MASTER, oddOrEven="odd")
    f.conditionalPageMasterReference(s, EVEN_MASTER, oddOrEven="even")
    masterAlts.end()
    seqMaster.end()

    #-- 5
    # [ s += (end tag for layout) ]
    layout.end()
7.6. **fosoxtest**: buildStatic()

This function adds static-content elements to the page-sequence element. Each one defines the content of a header or footer in one of the simple-page-master elements.

```python
# - - -  b u i l d  S t a t i c
def buildStatic(s):
    '''Set up all static content.

    [ s is a sox.Sox instance ->
      s += static-content elements that set up headers in
      regions ODD BEFORE and EVEN BEFORE and footers in
      regions ODD AFTER and EVEN AFTER ]

    ...

On odd pages, the header is the text “Running head”, right-aligned, and the footer has the page number right-aligned.

    #-- 1
    # [ s += a static-content element with fixed text in the
    #   ODD BEFORE region ]
    oddHeader = f.staticContent(s, ODD_BEFORE)
    b = f.block(s, HEAD_FONT, f.dash(textAlign="right"), "Running head")
    b.end()  
    oddHeader.end()

    #-- 2
    # [ s += a static-content element with the page number in
    #   the ODD AFTER region ]
    oddFooter = f.staticContent(s, ODD_AFTER)
    b = f.block(s, FOOT_FONT, textAlign="right")
    s.leaf("page-number")  
    b.end()  
    oddFooter.end()

The even-page headers and footers have the same content, only left-aligned.

    #-- 3
    # [ s += a static-content element with fixed text in the
    #   EVEN BEFORE region ]
    evenHeader = f.staticContent(s, EVEN_BEFORE)
    b = f.block(s, HEAD_FONT, f.dash(textAlign="left"), "Running head")
    b.end()  
    evenHeader.end()

    #-- 4
    # [ s += a static-content element with the page number in
    #   the EVEN AFTER region ]
    evenFooter = f.staticContent(s, EVEN_AFTER)
    b = f.block(s, FOOT_FONT, textAlign="left")
    s.leaf("page-number")
```

Zoological Data Processing  fosox: Generating XSL-FO in Python
7.7. **fosoxtest: buildFlow()**

This function builds the `flow` element and fills it with some sample text.

```python
# --- buildFlow

def buildFlow(s):
    '''Generate some sample content.

    [ s is a sox.SoX instance ->
      s += a flow element containing block elements with
      sample text ]

    ...
    sampleText = ("Now is the time for all good organisms to come to "
                  "the aid of their planet! ")
    flow = f.flow(s, "xsl-region-body")
    for i in range(17):
        s.leaf("block", BODY_FONT, f.dash(spaceBefore="10pt"),
               sampleText*17)
    flow.end()
```

7.8. **fosoxtest: Epilogue**

```python
# --- Epilogue

if __name__ == "__main__":
    main()
```

8. **Implementation of fosox.py**

The balance of this document contains the actual code for the `fosox.py` module in lightweight literate programming form.

```python
'''
fosox.py: XSL Formatting Objects helpers for Python

Do not edit this file directly. It is mechanically extracted from the documentation:

http://www.nmt.edu/~shipman/soft/fosox/
'''
```

---

30 http://www.nmt.edu/~shipman/soft/litprog/
9. Imports

Arithmetic in the `FoDim` class is done using Python’s standard `Decimal` module, which implements fixed-point arithmetic. We'll limit the precision to ten decimal places, which should be adequate for print layout.

```python
from decimal import Decimal, getcontext
getchecked().prec = 10
```

All XML generation is done using the `sox` module.

```python
import sox
```

10. Manifest constants

Because Python does not have constants *per se*, we use Python variables whose names are in capital letters with “_” used as a word separator.

10.1. FO_NAMESPACE

Name of the XSL-FO namespace, to be used as the `xmlns` attribute in the generated XML file.

```python
FO_NAMESPACE = "http://www.w3.org/1999/XSL/Format"
```

10.2. The UNITS_ constants

These constants are used for the five units acceptable in XSL-FO.

```python
UNITS_MM = "mm"
UNITS_CM = "cm"
UNITS_IN = "in"
UNITS_PC = "pc"
UNITS_PT = "pt"
```

10.3. The PAPER sizes

These constants are used to specify standard paper sizes to Section 6.5, “pageDimsFactory(): Some pre-built PageDims instances” (p. 12).

```python
PAPER_LETTER = 'letter'
PAPER_LETTER_LAND = 'letter-landscape'
```
11. Specification functions

These specification functions are part of the Cleanroom methodology. They define short names for certain conditions used in peer verification of the code's correctness.

```python
# - - - - - S p e c i f i c a t i o n f u n c t i o n s
#
# de-camel(s) ==
# a copy of s with all sequences 'aB', where a is any lowercase
# letter and b is any uppercase letter, replaced by 'a-b', and
# periods replaced by "_"
#
# decimal-okay ==
# any Python value acceptable as an argument to the
# decimal.Decimal() constructor
#
# dict-de-camel(d) ==
# a new dictionary with the elements of d, but with each key K
# replaced by de-camel(K)
#
# dim-okay ==
# a dimension as either an XSL-FO string (e.g., '2.5in') or
# as a FoDim instance
#
# units-okay ==
# any of the values in \{"in", "mm", "cm", "pc", "pt"\}
```

12. root()

Most of the functions are very short, consisting of single calls to `sox.start()` or `sox.leaf()`. This one uses Section 10.1, “FO_NAMESPACE” (p. 25) to define the `xmlns` attribute of the generated file.

```python
# - - - r o o t

def root(s):
    '''Generate a root element.
[ s is a sox.Sox instance ->
    s +=: an open root element
    return that element ]
...
    return s.start("root", xmlns=FO_NAMESPACE)
```

31 http://www.nmt.edu/~shipman/soft/clean/
13. `layoutMasterSet()`

```python
# - - - l a y o u t M a s t e r S e t
def layoutMasterSet(s):
    '''Open a layout-master-set element.

    [ s is a sox.Sox instance ->
    s +=: an open layout-master-set element
    return that element ]
    ...
    return s.start("layout-master-set")
```

14. `simpleMaster()`

```python
# - - - s i m p l e M a s t e r
def simpleMaster(s, masterName, dims):
    '''Build a simple-page-master element.

    [ (s is a sox.Sox instance) and
      (masterName is a valid XML name) and
      (dims is a PageDims instance) ->
    s +=: an open simple-page-master element using dims for page size and margins
    return that element ]
    ...
    return s.start("simple-page-master",
    dash(masterName=masterName,
    pageWidth=str(dims.pageBox.wide),
    pageHeight=str(dims.pageBox.high)),
    dims.pageMargins.dict())
```

15. `regionBody()`

```python
# - - - r e g i o n B o d y
def regionBody(s, dims):
    '''Generate a region-body element.

    [ (s is a sox.Sox instance) and
      (dims a PageDims instance) ->
    s +=: a complete region-body element using dims for body margins ]
    ...
    s.leaf("region-body", dims.bodyMargins.dict())
```
16. regionBefore()

```python
# --- regionBefore

def regionBefore(s, regName, dims):
    '''Generate a region-before element.

    [ (s is a sox.Sox instance) and
      (dims a PageDims instance) ->
      s += a complete region-before element tag using region-name
      (regName) and dims.frameMargins.top for its extent ]

    ... s.leaf("region-before",
               dash(regionName=regName,
               extent=str(dims.frameMargins.top)))
```

17. regionAfter()

```python
# --- regionAfter

def regionAfter(s, regName, dims):
    '''Generate a region-after element.

    [ (s is a sox.Sox instance) and
      (dims a PageDims instance) ->
      s += a complete region-after element tag using region-name
      (regName) and dims.frameMargins.bot for its extent ]

    ... s.leaf("region-after",
               dash(regionName=regName,
               extent=str(dims.frameMargins.bot)))
```

18. pageSequenceMaster()

```python
# --- pageSequenceMaster

def pageSequenceMaster(s, masterName):
    '''Open a page-sequence-master element.

    [ (s is a sox.Sox instance) and
      (name is a valid XML name) ->
      s += an open page-sequence-master element with name
      (masterName) ]

    ... return that element ]

    return s.start("page-sequence-master",
                   dash(masterName=masterName))
```
19. **singlePageMasterReference()**

```python
# - - - s i n g l e P a g e M a s t e r R e f e r e n c e
def singlePageMasterReference(s, masterRef):
    '''Generate a single-page-master-reference element.
    
    [ (s is a sox.Sox instance) and
      (masterRef is a valid XML name) ->
      s +=: an leaf single-page-master-reference element
      referring to (masterRef)
      return that element ]
    ...
    s.leaf("single-page-master-reference",
               dash(masterReference=masterRef))
```

20. **repeatablePageMasterReference()**

```python
# - - - r e p e a t a b l e P a g e M a s t e r R e f e r e n c e
def repeatablePageMasterReference(s, masterRef, **kw):
    '''Generate a repeatable-page-master-reference element.
    
    [ (s is a sox.Sox instance) and
      (masterRef is a valid XML name) and
      (kw is a dictionary) ->
      s +=: a leaf repeatable-page-master-reference element
      referring to (masterRef) with attributes
      dict-de-camel(kw) ]
    ...
    s.leaf("repeatable-page-master-reference",
               dash(masterReference=masterRef), dash(**kw))
```

21. **repeatablePageMasterAlternatives()**

```python
# - - - r e p e a t a b l e P a g e M a s t e r A l t e r n a t i v e s
def repeatablePageMasterAlternatives(s, **kw):
    '''Generate a repeatable-page-master-alternatives element.
    
    [ s is a sox.Sox instance ->
      s +=: an open repeatable-page-master-alternatives element
      return that element ]
    ...
    return s.start("repeatable-page-master-alternatives", dash(**kw))
```
22. conditionalPageMasterReference()

```python
def conditionalPageMasterReference(s, masterRef, **kw):
    '''Generate a conditional-page-master-reference element.

    [ (s is a sox.Sox instance) and
      (masterRef is a valid XML name) and
      (kw is a dictionary with keys in {"pagePosition", "oddOrEven", "blankOrNotBlank"}) ->
      s += a leaf conditional-page-master-reference element
      referring to (masterRef) with attributes
dict-de-camel(kw) ]
    ...
    return s.leaf("conditional-page-master-reference",
                 dash(masterReference=masterRef), dash(**kw))
```

23. pageSequence()

```python
def pageSequence(s, masterRef):
    '''Open a page-sequence element.

    [ (s is a sox.Sox instance) and
      (masterRef is an XML name) ->
      s += open element for page-sequence with master-reference to (masterRef)
    return that element ]
    ...
    return s.start("page-sequence",
                   dash(masterReference=masterRef))
```

24. staticContent()

```python
def staticContent(s, regName):
    '''Generate a static-content element for region (regName).

    [ s is a sox.Sox instance ->
      s += open element for static-content
    return that element ]
    ...
    return s.start("static-content",
                   dash(flowName=regName))
```
25. flow()

```python
#--- flow

def flow(s, flowName):
    '''Open a flow element.

    [ (s is a sox.Sox instance) and
      (flowName is a valid XML name) ->
      s += an open flow element routed to (flowName)
      return that element ]

    ...
    return s.start("flow", dash(flowName=flowName))
```

26. block()

```python
#--- block

def block(s, *p, **kw):
    '''Open a block element.

    [ (s is a sox.Sox instance) and
      (p is a sequence of text strings and dictionaries) and
      (kw is a dictionary) ->
      s += an open block element with attributes made from
dictionaries in p and dict-de-camel(kw) and
      (text strings in p))
      return than element ]

    ...
    return s.start("block", *p, **dash(**kw))
```

27. class FoDim: A linear dimension

An instance of this class represents a dimension expressed in one of the units acceptable to XSL-FO.

```python
#---- class FoDim

class FoDim(object):
    '''Represents a length dimension on XSL-FO.

    Exports:
    FoDim(n, units):
        [ (n is a number in a form acceptable to the Decimal()
          constructor) and
          (units is in ['in', 'mm', 'cm', 'pc', 'pt']) ->
          return a FoDim instance representing that length
          ]

    __str__(self):
        [ return a string representing self as an XSL-FO length ]
```
27.1. **FoDim.

```python
# - - - F o D i m . __ i n i t __
def __init__(self, n, units):
    '''Constructor
    ...
    self.n = Decimal(n)
    self.units = units
```

27.2. **FoDim.

```python
# - - - F o D i m . __ s t r __
def __str__(self):
    '''Convert to a string.
    ...
    return "%s%s" % (self.n, self.units)
```
27.3. **FoDim.__add__()**

Addition of two `FoDim` instances happens in two steps. First we convert the augend to the same type as `self`, then simply add their numbers.

```python
def __add__(self, other):
    '''Add two dimensions.'''
    #-- 1 --
    # [ otherCon := other expressed in units of (self.units) ]
    otherCon = other.convert(self.units)
    #-- 2 --
    return FoDim(self.n + otherCon.n, self.units)
```

27.4. **FoDim.__sub__()**

Subtraction of two units.

```python
def __sub__(self, other):
    '''Subtract two dimensions.'''
    #-- 1 --
    negOther = -other
    #-- 2 --
    return self + negOther
```

27.5. **FoDim.__mul__()**

Here, the right-hand operator must be a constant, and the result is `self` multiplied by that constant.

```python
def __mul__(self, other):
    '''Multiply self by a constant.'''
    return FoDim(self.n * Decimal(other), self.units)
```

27.6. **FoDim.__div__()**

Division by a constant.

```python
def __div__(self, other):
    pass
```
Divide self by a constant.

```python
return FoDim( self.n / Decimal(other), self.units )
```

### 27.7. `FoDim.__neg__()`

Negate a dimension.

```python
# --- FoDim.__neg__
def __neg__( self ):
    '''Negate a dimension.'''
    return FoDim(-self.n, self.units)
```

### 27.8. `FoDim.convert()`: Convert to different units

This function uses the static method Section 27.9, “`FoDim.confactor()`: Find a conversion factor” (p. 34) to find the number that must be multiplied by the current `.n` value to express it in the new units.

```python
# --- FoDim.convert
def convert(self, newUnits):
    '''Convert to the same value in different units (approximately).'''
    #-- 1 --
    # [ factor := the number by which self.n must be multiplied
    # to convert from self.units to newUnits ]
    factor = FoDim.confactor(self.units, newUnits)
    #-- 2 --
    return FoDim(self.n * factor, newUnits)
```

### 27.9. `FoDim.confactor()`: Find a conversion factor

The purpose of this static method is to find the conversion factor to be used in converting a dimension to different units.

First, `unitList` is a sequence of the five unit codes, arranged in a sequence in which units with an exact conversion factor are adjacent (picas next to points, and millimeters next to centimeters) when possible.

The sequence `factorList` is set up so that element `factorList[i]` is the factor that must be used to convert from a dimension in terms of `unitList[i]` to a dimension in terms of `unitList[i+1]`.

Then in order to derive a net conversion factor, we find the positions of the desired old and new units in `unitList`. If the new units are past the old units, we find the product of the factors between them; if the new units are before the old units, we find the inverse of the product of the factors between them.

```python
# --- FoDim.confactor (static method)
```
unitList = (UNIT_PC, UNIT_PT, UNIT_IN, UNIT_CM, UNIT_MM)
factorList = (Decimal(12), # 1pc = 12pt
             Decimal(1)/Decimal('72.27'), # 1pt = 1/72.27in
             Decimal('2.54'), # 1in = 2.54cm
             Decimal(10)) # 1cm = 10mm

@staticmethod
def confactor(fromUnits, toUnits):
    """Find any arbitrary conversion factor.
    [ fromUnits and toUnits are XSL-FO dimensional units ->
      return the factor that must be multiplied by a
      quantity using fromUnits to express it as toUnits ]
    ""
    fromPos = FoDim.unitList.index(fromUnits)
toPos = FoDim.unitList.index(toUnits)
result = Decimal(1)

    #-- 2 --
    # [ if fromPos < toPos ->
    #     result *= elements of FoDim.factorList in positions
    #     fromPos, fromPos+1, ..., toPos-1, inclusive
    #   else if fromPos > toPos ->
    #     result /= elements of FoDim.factorList in positions
    #     fromPos-1, fromPos-2, ..., toPos, inclusive ]
    if fromPos < toPos:
        for pos in range(fromPos, toPos):
            result *= FoDim.factorList[pos]
    elif fromPos > toPos:
        for pos in range(fromPos-1, toPos-1, -1):
            result /= FoDim.factorList[pos]

    #-- 3 --
    return result

For a test script that exercises all possible conversions, see Section 41, “confactest: Test driver for FoDim unit conversions” (p. 43).

28. class Box: Define a rectangle

An instance of this class represents a rectangle, defined by its width and height.

```python
# - - - - - c l a s s   B o x

class Box(object):
    """Represents the dimensions of a rectangle."""
```
Exports:
Box(wide, high):
[ wide and high are FoDim instances ->
  return a new Box representing those dimensions ]
  .wide: [ as passed to constructor ]
  .high: [ as passed to constructor ]
  __str__(self): [ return self as a string ]

28.1. Box.__init__()

```
# -- B o x . _ _ i n i t __

def __init__(self, wide, high):
  '''Constructor.
  ...
  self.wide, self.high = wide, high
```

28.2. Box.__str__()

This function is provided for debugging purposes.
```
# -- B o x . _ _ s t r __

def __str__(self):
  return ('<Box(w={0},h={1})>'.format(
    self.wide, self.high))
```

29. class MarginSet

An instance of this class represents a set of four margin sizes.
```
# - - - - - c l a s s M a r g i n S e t

class MarginSet(object):
  '''Represents a set of four margin sizes.

  Exports:
  MarginSet(top="0.0in", bot="0.0in", left="0.0in",
               right="0.0in"): [ arguments are valid XSL-FO dimensions ->
    return a new MarginSet instance representing top
    margin (top), bottom margin (bot), left margin
    (left), and right margin (right) ]
  .top, .bot, .left, .right: [ as passed to constructor ]
  .dict(): [ return a dict whose keys are "margin-top", etc.,
    with corresponding values from self.top, etc. ]
```
29.1. MarginSet.__init__()

```python
# - - - M a r g i n S e t . __ i n i t _ _

def __init__(self, top="0.0in", bot="0.0in", left="0.0in", right="0.0in"):
    '''Constructor.'''
    self.top = top
    self.bot = bot
    self.left = left
    self.right = right
```

29.2. MarginSet.dict()

Returns a dictionary suitable for use as the attributes of a box.

```python
# - - - M a r g i n S e t . d i c t

def dict(self):
    '''Return a dictionary of margin attributes.'''
    return dash(marginTop=str(self.top), marginBottom=str(self.bot),
                marginLeft=str(self.left), marginRight=str(self.right))
```

29.3. MarginSet.__str__()

```python
# - - - M a r g i n S e t . __ s t r _ _

def __str__(self):
    return("<MarginSet(t={0},b={1},l={2},r={3})>".format(
        self.top, self.bot, self.left, self.right))
```

30. class PageDims

An instance of this class describes a number of the dimensions of a full page layout.

```python
# - - - - - - c l a s s P a g e D i m s

class PageDims(object):
    '''Represents a page layout.'''

    Exports:
```
30.1. PageDims.__init__()

```python
# --- PageDims.__init__

def __init__(self, pageBox, pageMargins=None, frameMargins=None,
             bodyMargins=None):
    '''Constructor
    ...
    self.pageBox = pageBox
    self.pageMargins = pageMargins or MarginSet()
    self.frameMargins = frameMargins or MarginSet()
    self.bodyMargins = bodyMargins or MarginSet()
```

30.2. PageDims.__str__()

```python
# --- PageDims.__str__

def __str__(self):
    return "<PageDims(box(\{0\}), page(\{1\}), frame(\{2\}), "
            "body(\{3\}))>".format(
            self.pageBox, self.pageMargins, self.frameMargins, 
            self.bodyMargins))
```

31. pageDimsFactory()

This function manufactures a new PageDims instance, given one of the specific paper size and orientation codes defined in Section 10.3, “The PAPER sizes” (p. 25). For the dictionary mapping these codes to factory functions, see Section 36, “paperTypeMap” (p. 40).
# - - - p a g e D i m s F a c t o r y

def pageDimsFactory ( paperType ):  
    '''Create a new PageDims instance for a standard sheet size.  
    '''
    return paperTypeMap[paperType]()

32. letterPage()
Returns a new PageDims instance for letter paper in portrait orientation.

# - - - l e t t e r P a g e

def letterPage():
    return PageDims(
        pageBox=Box(FoDim("8.5", "in"), FoDim("11", "in")),
        pageMargins=MarginSet(top=FoDim("1", "in"),
            bot=FoDim("1", "in"), left=FoDim("1", "in"),
            right=FoDim("1", "in")),
        frameMargins=MarginSet(top=FoDim("3", "pc"),
            bot=FoDim("3", "pc"), left=FoDim("0", "pc"),
            right=FoDim("0", "pc")),
        bodyMargins=MarginSet(top=FoDim("3", "pc"),
            bot=FoDim("3", "pc"), left=FoDim("0", "pc"),
            right=FoDim("0", "pc"))

33. letterLandPage()
Returns a PageDims for letter paper in landscape orientation.

# - - - l e t t e r L a n d P a g e

def letterLandPage():
    return PageDims(
        pageBox=Box(FoDim("11", "in"), FoDim("8.5", "in")),
        pageMargins=MarginSet(top=FoDim("1", "in"),
            bot=FoDim("1", "in"), left=FoDim("1", "in"),
            right=FoDim("1", "in")),
        frameMargins=MarginSet(top=FoDim("3", "pc"),
            bot=FoDim("3", "pc"), left=FoDim("0", "pc"),
            right=FoDim("0", "pc")),
        bodyMargins=MarginSet(top=FoDim("3", "pc"),
            bot=FoDim("3", "pc"), left=FoDim("0", "pc"),
            right=FoDim("0", "pc"))

34. legalPage()
Returns a new PageDims instance for legal-size paper in portrait orientation.
def legalPage():
    return PageDims(
        pageBox=Box(FoDim("8.5", "in"), FoDim("14", "in")),
        pageMargins=MarginSet(top=FoDim("1", "in"),
                                bot=FoDim("1", "in"), left=FoDim("1", "in"),
                                right=FoDim("1", "in")),
        frameMargins=MarginSet(top=FoDim("3", "pc"),
                                bot=FoDim("3", "pc"), left=FoDim("0", "pc"),
                                right=FoDim("0", "pc")),
        bodyMargins=MarginSet(top=FoDim("3", "pc"),
                                bot=FoDim("3", "pc"), left=FoDim("0", "pc"),
                                right=FoDim("0", "pc"))
    )

35. indexCardPage()

Returns a new PageDims instance for a standard 3"×5" card in portrait orientation.

def indexCardPage():
    return PageDims(
        pageBox=Box(FoDim("3", "in"), FoDim("5", "in")),
        pageMargins=MarginSet(top=FoDim("0.1", "in"),
                                bot=FoDim("0.1", "in"), left=FoDim("0.25", "in"),
                                right=FoDim("0.25", "in")),
        frameMargins=MarginSet(top=FoDim("1.5", "pc"),
                                bot=FoDim("1", "pc"), left=FoDim("0", "pc"),
                                right=FoDim("0", "pc")),
        bodyMargins=MarginSet(top=FoDim("2", "pc"),
                                bot=FoDim("2", "pc"), left=FoDim("0", "pc"),
                                right=FoDim("0", "pc"))
    )

36. paperTypeMap

This dictionary is used to convert a standard paper type codes from Section 10.3, “The PAPER sizes” (p. 25) into a PageDims instance.

def paperTypeMap:
    return {
        PAPER_LETTER: letterPage,
        PAPER_LETTER_LAND: letterLandPage,
        PAPER_LEGAL: legalPage,
        PAPER_3X5: indexCardPage
    }
37. **font()**

This function uses Section 40, “deCamel(): Convert from camel-case to dashed attribute names” (p. 42) to convert camel-case attribute names to XSL-FO attribute names.

```python
# - - - f o n t

def font(**kw):
    '''Return a dictionary of font properties.
    '''
    #-- 1
    result = {}
    #-- 2
    # [ result += key-value pairs from kw but with each key
    # replaced by de-camel(key) ]
    for key in kw:
        result[deCamel(key)] = kw[key]
    #-- 3
    return result
```

38. **leader()**

This function generates a leader element. If no length is specified, the resulting leader is stretchable; otherwise we use the leader-length attribute to specify the fixed length. The default pattern is `space`, meaning that it is rendered as whitespace.

```python
# - - - l e a d e r

def leader(s, length=None, pattern=None):
    '''Produce a leader element.
    '''
    #--1
    if length is None: d = {}
    else:
        d = dash(leaderLength=str(length))
    #-- 2
    if pattern is None: pat = 'space'
    else:
        pat = pattern
    #-- 3
    s.leaf('leader', d, dash(leaderPattern=pat))
```

39. **dash()**

This function converts a dictionary with keys in camel-case to a new dictionary with the keys run through Section 40, “deCamel(): Convert from camel-case to dashed attribute names” (p. 42).
# dash

def dash(**kw):
    '''XSL-FO attribute dictionary builder.

    kw is a dictionary ->
    return a new dictionary with the same elements as kw,
    but with each key transformed by substituting "x-y" for any sequence "xY" where x is lowercase and Y is uppercase
    ...'''
    #-- 1
    result = {}
    #-- 2
    for key in kw:
        result[deCamel(key)] = kw[key]
    #-- 3
    return result

40. deCamel(): Convert from camel-case to dashed attribute names

def deCamel(name):
    '''Change camelCase names to hyphenated-names.

    name is a str ->
    return name, transformed by substituting "x-y" for any sequence "xY" where x is lowercase and Y is uppercase
    ...'''
    #-- 1
    last = 'A'
    result = []
    #-- 2
    # [ result += all but the first character of (last+name)
    #   with sequences "xY" replaced replaced by "x-y" and
    #   "." replaced by "_"
    #   last := the last character of name ]
    for c in name:
        #-- 2 body
        if last.islower() and c.isupper():
            result.append("-%" + c.lower())
        elif c=='.:
            result.append('_')
41. confactest: Test driver for FoDim unit conversions

This test driver demonstrates all possible unit conversions to verify that Section 27.8, “FoDim.convert(): Convert to different units” (p. 34) is working correctly.

```python
#!/usr/bin/env python
# confactest: Test driver for FoDim.confactor()
# For documentation, see:
# http://www.nmt.edu/tcc/projects/fohelpers/
import fosox as f
u = f.FoDim.unitList
for fromPos in range(0, len(u)):
    for toPos in range(0, len(u)):
        factor = f.FoDim.confactor(u[fromPos], u[toPos])
        print ("To convert from %s to %s, multiply by %s" %
               (u[fromPos], u[toPos], factor) )
```

```python
else:
    result.append(c)
last = c

#-- 3
return ''.join(result)
```