Implementation Overview

Following a Document Through FOP

by Arved Sandstrom

The purpose of this document is to tie together the FOP design (interface) with some of the key points where control is passed within FOP (implementation), so that developers can quickly find the section of code that is relevant to their needs. The process described is for a "typical" command-line document. All classes are in org.apache.fop unless otherwise designated.

1. Overview

The input FO document is sent to the FO tree builder via SAX events. Fragments of an FO Tree are built from this process. As each page-sequence element is completed, it is passed to a layout processor, which in turn converts it into an Area Tree. The Area Tree is then given to the Renderer, which converts it into a stream of data containing the output document. The sections below will provide additional details. Where needed differences between the trunk and maintenance branches are shown in tabular format.

2. Startup

- The job starts in *apps.Fop.main()*.
- Control is passed to *apps.CommandLineStarter.run()*.
- Control is passed to *apps.Driver.render()*. This class fires up a SAX parser, the events from which indirectly control the remaining processing, including building the FO Tree, building the Area Tree, rendering, output and logging.

3. Formatting Object Tree

Trunk	Maintenance
The SAX events that the parser creates are handled by *fo.FOTreeBuilder*, which uses `startElement()`, `endElement()`, and `characters()` methods to build the FO Tree.	
`fo.FOTreeBuilder.endElement()` runs the `end()` method for each node as it is created. The *fo.pagination.PageSequence* class	the end of a PageSequence element causes the PageSequence object to be passed to `apps.StreamRenderer.render()`, which in

overrides this `end()` method to run `apps.LayoutHandler.endPageSequence()`, which in turn runs `fo.pagination.PageSequence.format()`.	turn runs `fo.pagination.PageSequence.format()`.
`fo.pagination.PageSequence.format()` creates a *layoutmgr.PageLayoutManager*, passing the AreaTree and PageSequence objects to it, then calls its `run()` method.	`fo.pagination.PageSequence.addFlow()` programatically adds a Flow object to the page sequence.
.	`fo.pagination.PageSequence.makePage()` creates a BodyArea and passes it to *fo.Flow.layout*
.	the layout process is then driven from `fo.pagination.PageSequence.format()`.

4. Layout

There are layout managers for each type of layout decision. They take an FO Tree as input and build a laid-out Area Tree from it. The layout process involves finding out where line breaks and page breaks should be made, then creating the areas on the page. Static areas can then be added for any static regions. As pages are completed, they are added to the Area Tree.

5. Area Tree

The area tree is a data structure designed to hold the page areas. These pages are then filled with the page regions and various areas. The area tree is used primarily as a minimal structure that can be rendered by the renderers.

The area tree is supported by an area tree model. This model handles the adding of pages to the area tree. It also handles page sequence starts, document level extensions, id references and unresolved id areas. This model allows the pages to be handled directly by a renderer or to store the pages for later use.

6. Rendering

The renderer receives pages from the area tree and renders those pages. If a renderer supports out of order rendering then it will either render or prepare a page in the correct order. Otherwise the pages are rendered in order. The task of the renderer is to take the pages and output them to the requested type. In the case of the AWTRenderer it needs to be able to view any page.

When rendering a page it takes the page and renders each page region. The main work for a renderer implementation is to handle the viewports and inline areas. The inline areas need to be drawn on the page in the correct place.