Apache™ FOP Design: FO Tree

Version 1298724

by Keiron Liddle

Table of contents

1 Introduction

The FO Tree is an internal hierarchical representation (java objects and properties) of the input XSL-FO document, and is created from the parsing of that XSL-FO document. The process of building the FO Tree corresponds to the **Objectify** step in the XSL-FO spec. The FO Tree is an intermediate structure which will later be converted into the area tree.

2 Processing

The SAX Events that are fired by the parsing process are caught by the FO Tree system. Events for starting an element, ending an element, and text data are assembled by the FO Tree system into a set of objects that represent the input FO document. A class exists for each element in the XSL-FO set, and an object in the appropriate class is created for each element in the input XSL-FO.

For attributes attached to an XSL-FO element, a property list mapping is used to convert the attribute into properties of the object related to the element.

To the extent possible, validation is checked as the FO Tree is built. An appropriate error message is returned to the user, and processing continues if possible.

Elements from foreign namespaces that are recognized by FOP fall into the following categories:

- Pass-thru: These are placed into a DOM object, which is then passed through FOP directly to the renderer. SVG is an example.
- FOP Internal: These are placed into objects that can then be used by FOP. An example of this would be an element that the layout process will use to create an area. Another example would be an element that contains setup information for the renderer.

For unrecognized namespaces, a dummy object or a generic DOM is created.

While the tree building is mainly about creating the FO Tree, some FO Tree events trigger processes in other parts of FOP. The end of a page-sequence element triggers the layout process for that page-sequence (see discussion of Recycling). Also, the end of the XML document tells the renderer that it can finalize the output document.

3 Recycling FO Tree Memory

To minimize the amount of memory used by FOP, we wish to recycle FO Tree memory as much as possible. There are at least three possible places that FO Tree fragments could be passed to the Layout process, so that their memory can be reused:

- **fo:block** It might be tempting to start laying out pages as soon as the first fo:block object is finished. However, there are many downstream things that can affect the placement of that block on a page, such as graphics and footnotes. So, in order to maintain conformance to the XSL-FO specification, and create high-quality output, we must see more of the document.
- **fo:root** The other extreme is to wait until the entire document is read in before processing any of it. This essentially means that there is no memory recycling. Processing the document correctly is more important than saving memory, so this option would be used if there were no better alternative.
- **fo:page-sequence** The page-sequence object provides a nice clean break in the document. Content from one page-sequence will never interfere with nor affect the placement of the content of another.

FOP uses this option as the optimum way to maintain compliance with the standard and to minimize memory consumption.

4 FO Tree Serialization

This issue is implied by the requirement to process documents of arbitrary size. Unless some arbitrary limit is placed on the size of page-sequence objects, FOP must be able to serialize FO tree fragments as necessary.

5 Notes About Specific Elements

5.1 page-master

The first elements in a document are the elements for the page master setup. This is usually only a small number and will be used throughout the document to create new pages. The root element keeps these objects as a factory to create the page and appropriate regions whenever a new page is requested by the layout. The objects in the FO Tree that represent these elements are themselves the factory.

5.2 flow

The elements within the flow of a page-sequence are those that are needed for the layout process. These element will later be used to create areas in the layout process.

5.3 Other Elements

The remaining FO Objects are things like page-sequence, title and color-profile. Each is handled by its parent element. The root handles declarations, and declarations maintains a list of colour profiles. The page-sequences are direct descendants of root.

6 Implementation Notes

6.1 FONode

The base class for all objects in the tree is FONode. The base class for all FO Objects is FObj (which is a subclass of FONode). Other FONode subclasses are for foreign and unknown XML.

Each FO in FOP has a parent (except root) and a Vector of children. Java inheritance (superclasses and subclasses) is used to enforce constraints required by the FO hierarchy.

FONode, among other things, ensures that each FO has a parent, and provides the mechanism for keeping track of its children.

Each xml element is represented by a java object. The classes for these objects are grouped into packages for convenience:

- pagination: `org.apache.fop.fo.pagination.*`
- flow: `org.apache.fop.fo.flow.*`
- other: `org.apache.fop.fo.*.`

6.2 Creating FO Objects

The process of creating an FO Object is as follows (see `FOTreeBuilder.startElement()` for details):

- An FO maker is selected from a hashmap lookup using the namespace and element name.
- This maker is then used to create the new object that represents the FO element.
- The new object is given its element name, an FOUserAgent (for resolving properties, etc.), a logger and its attributes.
- The new object is added to the FO tree as a child of the current parent.
- Child elements are then processed. This is an iterative process: the child elements go through the same process here documented for their parent.

6.3 Foreign XML

For SVG, the DOM needs to be created with Batik, so an element mapping is used to read all elements in the SVG namespace and pass them into the Batik DOM.

The base class for foreign XML is XMLObj. This class handles creating a DOM Element and the setting of attributes. It also can create a DOM Document if it is a top level element, class XMLElement. This class must be extended for the namespace of the XML elements. For unknown namespaces the class is UnknowXMLObj.

If some special processing is needed, then the top level element can extend the XMLObj. For example the SVGElement makes the special DOM required for batik and gets the size of the svg.

Foreign XML will usually be in an fo:instream-foreign-object. The XML will be passed to the renderer as a DOM, which is expected to know how to handle it. XML from an unknown namespace will be ignored.

See Input Parsing Namespaces for more discussion and links to information about using foreign XML in FOP.

6.4 Unknown Elements

If an element is in a known namespace but the element is unknown within that namespace, then an Unknown object is created. This generally indicates an input error: perhaps an element from an older version of the XSL-FO standard, or a misspelling. The Unknown object is mainly used to provide information to the user.