FOP Design: Properties

$Revision: 911792 $

by Karen Lease

Table of contents

1. Introduction

As the input XSL-FO is being parsed and the FO Tree is being built, the attributes of the FO elements are passed by the parser to the related FO object. The java object that represent the FO object then converts the attributes into properties that are stored in the FO Tree.

2. Issues

The following are some issues when dealing with properties:

- Initial Property Set
- Inheritance: Some properties can be inherited from parent objects.
- Adoption: The parentage for some elements can move around. Markers are one example.
- Multiple Namespaces: The properties for foreign namespaces must be handled.
- Expressions: XSL-FO expressions can be included in properties.

3. Overview of Processing

The general flow of property processing is as follows:

- As part of `FOTreeBuilder.startElement()`, `FObj.handleAttrs` is passed a list of attributes to be processed for the new FObj.
- FObj.handleAttrs gets a PropertyListBuilder and asks it to create a Property List from the list of attributes. There is currently only one static PropertyListBuilder, which handles the fo: namespace.
- FObj.handleAttrs then cross-references the returned PropertyList with the FObj, creates a PropertyManager to facilitate downstream processing of the PropertyList, and handles the special case of the writing-mode property.

4. PropertyListBuilder

Each plb object contains a hash of property names and *their* respective Makers. It may also contain element-specific property maker hashes; these are based on the *local name* of the flow object, ie. *table-row*, not *fo:table-row*. If an element-specific property mapping exists, it is preferred to the generic mapping.

The PLB loops through each attribute in the list, finds an appropriate "Maker" for it, then calls the Maker to convert the attribute value into a Property object of the correct type, and stores that Property in the PropertyList.

5. Property datatypes

The property datatypes are defined in the org.apache.fop.datatypes package, except Number and String which are java primitives. The FOP datatypes are:

PDF created by Apache FOP
http://xmlgraphics.apache.org/fop/

- Number
- String
- ColorType
- Length (has several subclasses)
- CondLength (compound)
- LengthRange (compound)
- Space (compound)
- Keep (compound)

The *org.apache.fop.fo.Property* class is the superclass for all Property subclasses. There is a subclass for each kind of property datatype. These are named using the datatype name plus the word Property, resulting in NumberProperty, StringProperty, and so on. There is also a class EnumProperty which uses an `int` primitive to hold enumerated values. There is no corresponding Enum datatype class.

The Property class provides a "wrapper" around any possible property value. Code manipulating property values (in layout for example) usually knows what kind (or kinds) of datatypes are acceptable for a given property and will use the appropriate accessor.

The base Property class defines accessor methods for all FO property datatypes, such as getNumber(), getColorType(), getSpace(), getEnum(), etc. It doesn't define accessors for SVG types, since these are handled separately (at least for now.) In the base Property class, all of these methods return null, except getEnum which returns 0. Individual subclasses return a value of the appropriate type, such as Length or ColorType. A subclass may also choose to return a reasonable value for other accessor types. For example, a SpaceProperty will return the optimum value if asked for a Length.

6. Property Makers

The Property class contains a nested class called *Maker*. This is the base class for all other property Makers. It provides basic framework functionality which is overridden by the code generated by properties.xsl from the *properties.xml files. In particular it provides basic expression evaluation, using PropertyParser class in the org.apache.fop.fo.expr package.

Other Property subclasses such as LengthProperty define their own nested Maker classes (subclasses of Property.Maker). These handle conversion from the Property subclass returned from expression evaluation into the appropriate subclass for the property.

For each generic or specific property definition in the properties.xml files, a new subclass of one of the Maker classes is created. Note that no new Property subclasses are created, only new PropertyMaker subclasses. Once the property value has been parsed and stored, it has no specific functionality. Only the Maker code is specific. Maker subclasses define such aspects as keyword substitutions, whether the property can be inherited or not, which enumerated values are legal, default values, corresponding properties and specific datatype conversions.

The PLB finds a "Maker" for the property based on the attribute name and the element name. Most Makers are generic and handle the attribute on any element, but it's possible to set up an element-specific property Maker. The attribute name to Maker mappings are automatically created during the code generation phase by processing the XML property description files.

PDF created by Apache FOP
http://xmlgraphics.apache.org/fop/

7. Processing the attribute list

The PLB first looks to see if the font-size property is specified, since it sets up relative units which can be used in other property specifications. Each attribute is then handled in turn. If the attribute specifies part of a compound property such as space-before.optimum, the PLB looks to see if the attribute list also contains the "base" property (space-before in this case) and processes that first.

8. How the Property Maker works

There is a family of Maker objects for each of the property datatypes, such as Length, Number, Enumerated, Space, etc. But since each Property has specific aspects such as whether it's inherited, its default value, its corresponding properties, etc. there is usually a specific Maker for each Property. All these Maker classes are created during the code generation phase by processing (using XSLT) the XML property description files to create Java classes.

The Maker first checks for "keyword" values for a property. These are things like "thin, medium, thick" for the border-width property. The datatype is really a Length but it can be specified using these keywords whose actual value is determined by the "User Agent" rather than being specified in the XSL standard. For FOP, these values are currently defined in foproperties.xml. The keyword value is just a string, so it still needs to be parsed as described next.

The Maker also checks to see if the property is an Enumerated type and then checks whether the value matches one of the specified enumeration values.

Otherwise the Maker uses the property parser in the fo.expr package to evaluate the attribute value and return a Property object. The parser interprets the expression language and performs numeric operations and function call evaluations.

If the returned Property value is of the correct type (specificed in foproperties.xml, where else?), the Maker returns it. Otherwise, it may be able to convert the returned type into the correct type.

Some kinds of property values can't be fully resolved during FO tree building because they depend on layout information. This is the case of length values specified as percentages and of the special proportional-column-width(x) specification for table-column widths. These are stored as special kinds of Length objects which are evaluated during layout. Expressions involving "em" units which are relative to font-size _are_ resolved during the FO tree building however.

9. Structure of the PropertyList

The PropertyList extends HashMap and its basic function is to associate Property value objects with Property names. The Property objects are all subclasses of the base Property class. Each one simply contains a reference to one of the property datatype objects. Property provides accessors for all known datatypes and various subclasses override the accessor(s) which are reasonable for the datatype they store.

PDF created by Apache FOP
http://xmlgraphics.apache.org/fop/

The PropertyList itself provides various ways of looking up Property values to handle such issues as inheritance and corresponding properties.

The main logic is:
If the property is a writing-mode relative property (using start, end, before or after in its name), the corresponding absolute property value is returned if it's explicitly set on this FO.
Otherwise, the writing-mode relative value is returned if it's explicitly set. If the property is inherited, the process repeats using the PropertyList of the FO's parent object. (This is easy because each PropertyList points to the PropertyList of the nearest ancestor FO.) If the property isn't inherited or no value is found at any level, the initial value is returned.

10. Implementing Standard Properties

Because the properties defined in the standard are basically static, FOP currently builds the source code for the related Property classes from an XML data file. All properties are specified in src/codegen/foproperties.xml. The related classes are created automatically during the build process by applying an XSLT stylesheet to the foproperties.xml file.

10.1. Generic properties

In the properties xml files, one can define generic property definitions which can serve as a basis for individual property definitions. There are currently several generic properties defined in foproperties.xml. An example is GenericColor, which defines basic properties for all ColorType properties. Since the generic specification doesn't include the inherited or default elements, these should be set in each property which is based on GenericColor. Here is an example:

```
<property type='generic'> <name>background-color</name>
<use-generic>GenericColor</use-generic> <inherited>false</inherited>
<default>transparent</default> </property>
```

A generic property specification can include all of the elements defined for the property element in the DTD, including the description of components for compound properties, and the specification of keyword shorthands.

Generic property specifications can be based on other generic specifications. An example is GenericCondPadding template which is based on the GenericCondLength definition but which extends it by adding an inherited element and a default value for the length component.

Generic properties can specify enumerated values, as in the GenericBorderStyle template. This means that the list of values, which is used by 8 properties (the "absolute" and "writing-mode-relative" variants for each BorderStyle property) is only specified one time.

When a property includes a "use-generic" element and includes no other elements (except the "name" element), then no class is generated for the property. Instead the generated mapping will associate this property directly with an instance of the generic Maker.

A generic class may also be hand-coded, rather than generated from the properties file. Properties based

PDF created by Apache FOP
http://xmlgraphics.apache.org/fop/

on such a generic class are indicated by the attribute `ispropclass='true'` on the *use-generic* element.

This is illustrated by the SVG properties, most of which use one of the Property subclasses defined in the *org.apache.fop.svg* package. Although all of these properties are now declared in svgproperties.xml, no specific classes are generated. Classes are only generated for those SVG properties which are not based on generic classes defined in svg.

10.2. Element-specific properties

Properties may be defined for all flow objects or only for particular flow objects. A PropertyListBuilder object will always look first for a Property.Maker for the flow object before looking in the general list. These are specified in the `element-property-list` section of the properties.xml files. The `localname` element children of this element specify for which flow-object elements the property should be registered.

NOTE: All the properties for an object or set of objects must be specified in a single element-property-list element. If the same localname appears in several element lists, the later set of properties will hide the earlier ones! Use the *ref* functionality if the same property is to be used in different sets of element-specific mappings.

10.3. Reference properties

A property element may have a type attribute with the value `ref`. The content of the *name* child element is the name of the referenced property (not its class-name!). This indicates that the property specification has already been given, either in this same specification file or in a different one (indicated by the `family` attribute). The value of the family attribute is *XX* where the file *XXproperties.xml* defines the referenced property. For example, some SVG objects may have properties defined for FO. Rather than defining them again with a new name, the SVG properties simply reference the defined FO properties. The generating mapping for the SVG properties will use the FO Maker classes.

10.4. Corresponding properties

Some properties have both *absolute* and *writing-mode-relative* forms. In general, the absolute forms are equivalent to CSS properties, and the writing-mode-relative forms are based on DSSSL. FO files may use either or both forms. In FOP code, a request for an absolute form will retrieve that value if it was specified on the FO; otherwise the corresponding relative property will be used if it was specified. However, a request for a relative form will only use the specified relative value if the corresponding absolute value was *not* specified for that FO.

Corresponding properties are specified in the properties.xml files using the element `corresponding`, which has at least one `propval` child and may have a `propexpr` child, if the corresponding value is calculated based on several other properties, as for `start-indent`.

NOTE: most current FOP code accesses the absolute variants of these properties, notably for padding, border, height and width attributes. However it does use start-indent and end-indent, rather than the

PDF created by Apache FOP
http://xmlgraphics.apache.org/fop/

"absolute" margin properties.

11. Mapping

The XSL script `propmap.xsl` is used to generate property mappings based on both foproperties.xml and svgproperties.xml. The mapping classes in the main fop packages simply load these automatically generated mappings. The mapping code still uses the static "maker" function of the generated object to obtain a Maker object. However, for all generated classes, this method returns an instance of the class itself (which is a subclass of Property.Maker) and not an instance of a separate nested Maker class.

For most SVG properties which use the SVG Property classes directly, the generated mapper code calls the "maker" method of the SVG Property class, which returns an instance of its nested Maker class.

The property generation also handles element-specific property mappings as specified in the properties XML files.

12. Enumerated values

For any property whose datatype is `Enum` or which contains possible enumerated values, FOP code may need to access enumeration constants. These are defined in the interfaces whose name is the same as the generated class name for the property, for example `BorderBeforeStyle.NONE`. These interface classes are generated by the XSL script `enumgen.xsl`. A separate interface defining the enumeration constants is always generated for every property which uses the constants, even if the constants themselves are defined in a generic class, as in BorderStyle.

If a subproperty or component of a compound property has enumerated values, the constants are defined in a nested interface whose name is the name of the subproperty (using appropriate capitalization rules). For example, the keep properties may have values of AUTO or FORCE or an integer value. These are defined for each kind of keep property. For example, the keep-together property is a compound property with the components within-line, within-column and within-page. Since each component may have the values AUTO or FORCE, the KeepTogether interface defines three nested interfaces, one for each component, and each defines these two constants. An example of a reference in code to the constant is `KeepTogether.WithinPage.AUTO`.

13. Compound property types

Some XSL FO properties are specified by compound datatypes. In the FO file, these are defined by a group of attributes, each having a name of the form `property.component`, for example `space-before.minimum`. These are several compound datatypes:

- LengthConditional, with components length and conditionality
- LengthRange, with components minimum, optimum, and maximum
- Space, with components minimum, optimum, maximum, precedence and conditionality
- Keep, with components within-line, within-column and within-page

PDF created by Apache FOP
http://xmlgraphics.apache.org/fop/

These are described in the properties.xml files using the element `compound` which has `subproperty` children. A subproperty element is much like a property element, although it may not have an `inherited` child element, as only a complete property object may be inherited.

Specific datatype classes exist for each compound property. Each component of a compound datatype is itself stored as a Property object. Individual components may be accessed either by directly performing a get operation on the name, using the "dot" notation, eg. `get("space-before.optimum")`; or by using an accessor on the compound property, eg. `get("space-before").getOptimum()`. In either case, the result is a Property object, and the actual value may be accessed (in this example) by using the "getLength()" accessor.

14. Refinement

The **Refinement** step is part of reading and using the properties which may happen immediately or during the layout process. FOP does not currently use a separate Refinement process, but tends to handle refining steps as the FO Tree is built.

15. Refined FO Tree

The Refined FO Tree is the result of the Refinement process.

PDF created by Apache FOP
http://xmlgraphics.apache.org/fop/