Implementing Properties

by Peter B. West

Table of contents

1. An alternative properties implementation

Note:
The following discussion focusses on the relationship between Flow Objects in the Flow Object tree, and properties. There is no (or only passing) discussion of the relationship between properties and traits, and by extension, between properties and the Area tree.

Property handling is complex and expensive. Varying numbers of properties **apply** to individual Flow Objects **(FOs)** in the **FO tree** but any property may effectively be assigned a value on any element of the tree. If that property is inheritable, its defined value will then be available to any children of the defining FO.

Note:
(XSL 1.0 Rec) **5.1.4 Inheritance** ...The inheritable properties can be placed on any formatting object.

Even if the value is not inheritable, it may be accessed by its children through the `inherit` keyword or the `from-parent()` core function, and potentially by any of its descendents through the `from-nearest-specified-value()` core function.

In addition to the assigned values of properties, almost every property has an **initial value** which is used when no value has been assigned.

1.1. The history problem

The difficulty and expense of handling properties comes from this univeral inheritance possibility. The list of properties which are assigned values on any particular *FO* element will not generally be large, but a current value is required for each property which applies to the *FO* being processed.

The environment from which these values may be selected includes, for each *FO*, **for each applicable property**, the value assigned on this *FO*, the value which applied to the parent of this *FO*, the nearest value specified on an ancestor of this element, and the initial value of the property.

1.2. The construction hierarchy

Properties are resoved in the **FO tree** in a strictly hierarchical manner. Nodes are detected in the input in a **pre-order** traversal, and are built in the same order. This imples that there are two phases, or states, of property resolution and construction. Any particular FO node is either in a state of constructing its own subtree, or in a stable state where the subtree construction is complete. These states have differenct data requirements.

Subtree building
In this state, all properties defined on this node, or any of its ancestors must be available to the subtree. In effect, any property defined on this node must be available to its descendants, as all properties defined on any ancestor are available to this node.
Stable: subtree building complete
In this state, only the properties **applicable to this node** need be available.

1.3. Representing properties: <property> classes

1.3.1. Class vs instance

What information is required of property objects? More particularly, what information is particular to the property classes, and what to the instantiated objects? The answer to this question depend largely on how the property objects are used in the context of layout and Area tree construction. The approach taken in this implementation is that properties are simply flags on certain data values associated with FOs. The semantics of these flags are determined within the layout engine.

Certain constant information attaches to individual property classes. This information is detailed in the descriptions of individual properties in *Section 7* of the specification. Such information is represented in **class** fields and data structures within the classes.

The "instance" information content of a property is:

- explicitly, the `PropertyValue` datum of the property, and
- implicitly, the **Flow Object** to which the property is attached.

Properties, then, serve essentially to link *FO instances* with *PropertyValue instances*, attaching certain invariant semantic markers to the PropertyValues in the process. In this implementation, these functions can be realised entirely within the property **classes** themselves, without the need to instantiate any objects. In practice, **property singletons** are instantiated to make access to some invariants simpler.

Next: property classes overview. (classes-overview.html)