

Extensible Stylesheet Language (XSL)

Abstract

This document is designed to test support of XSL 1.1 change bars

The resulting FO specifies change-bar-placement="alternate" on the fo:root element and allows it to be inherited by all instances of change-bar-begin.

On page 1, there is a change bar down the left side of most of the third paragraph. The bar is 2pt thick, and the center of the bar is the default 6pt distance from the left side of the paragraph text. This change bar is red.

On the following paragraph, there is a change bar that is on the left side of the part of the para in the first column and on the right side of the part of the para in the second column. This rule is 2pt thick and its center is 3pt from the edge of the paragraph text.

Starting with the second para in section 1.2 Tree Transformations, there is a change bar with thickness 1pt whose center is offset 3pt from the text. This bar continues through to a little more than halfway through the second column on page 2. Meanwhile, starting with the first full paragraph on page 2 and continuing until the end of the first partial paragraph on page 3 is a change bar whose thickness is 3pt and whose offset is 10pt.

Then starting with the last partial paragraph on page 2 and continuing through to the end of the penultimate paragraph of section 1.3 is a change bar with thickness 1pt whose center is offset 3pt from the text.

Contents

1 Processing a Stylesheet.....	1
1.1 Introduction to stylesheet processing.....	1
1.2 Tree Transformations.....	1
1.3 Formatting.....	2
1.4 Some test lists.....	3

1 Processing a Stylesheet

1.1 Introduction to stylesheet processing

An XSL **stylesheet processor** accepts a document or data in XML and an XSL stylesheet and produces the presentation of that XML source content that was intended by the designer of that stylesheet. There are two aspects of this presentation process: first, constructing a result tree from the XML source tree and second, interpreting the result tree to produce formatted results suitable for presentation on a display, on paper, in speech, or onto other media. The first aspect is called **tree transformation** and the second is called **formatting**. The process of formatting is performed by the **formatter**. This formatter may simply be a rendering engine inside a browser.

Tree transformation allows the structure of the result tree to be significantly different from the structure of the source tree. For example, one could add a table-of-contents as a filtered selection of an original source document, or one could rearrange source data into a sorted tabular presentation. In constructing the result tree, the tree transformation process also adds the information necessary to format that result tree.

Formatting is enabled by including formatting semantics in the result tree. Formatting semantics are expressed in terms of a catalog of classes of **formatting objects**. The nodes of the result tree are formatting objects. The classes of formatting objects denote typographic abstractions such as page, paragraph, table, and so forth. Finer control over the presentation of these abstractions is provided by a set of formatting properties, such as those controlling indents, word- and letter-spacing, and widow, orphan, and hyphenation control. In XSL, the classes of **formatting objects** and **formatting properties** provide the vocabulary for expressing presentation intent.

The nodes of the result tree are formatting objects. The classes of formatting objects denote typographic abstractions such as

page, paragraph, table, and so forth. Finer control over the presentation of these abstractions is provided by a set of formatting properties, such as those controlling indents, word- and letter-spacing, and widow, orphan, and hyphenation control. In XSL, the classes of formatting objects and formatting properties provide the vocabulary for expressing presentation intent.

The XSL processing model is intended to be conceptual only. An implementation is not mandated to provide these as separate processes. Furthermore, implementations are free to process the source document in any way that produces the same result as if it were processed using the conceptual XSL processing model. A diagram depicting the detailed conceptual model is shown below.

1.2 Tree Transformations

Tree transformation constructs the result tree. In XSL, this tree is called the **element and attribute tree**, with objects primarily in the “formatting object” namespace. In this tree, a formatting object is represented as an XML element, with the properties represented by a set of XML attribute-value pairs. The content of the formatting object is the content of the XML element. Tree transformation is defined in the XSLT Recommendation.

The XSL stylesheet is used in tree transformation. A stylesheet contains a set of tree construction rules. The tree construction rules have two parts: a pattern that is matched against elements in the source tree and a template that constructs a portion of the result tree. This allows a stylesheet to be applicable to a wide class of documents that have similar source tree structures. A diagram depicting this conceptual process is shown below.

In some implementations of XSL/XSLT, the result of tree construction can be output as an XML document. This would allow an XML document which contains formatting objects and formatting properties to be out-

put. This capability is neither necessary for an XSL processor nor is it encouraged.

There are, however, cases where this is important, such as a server preparing input for a known client; for example, the way that a WAP (<http://www.wapforum.org/faqs/index.htm>) server prepares specialized input for a WAP capable hand held device. To preserve accessibility, designers of Web systems should not develop architectures that require (or use) the transmission of documents containing formatting objects and properties unless either the transmitter knows that the client can accept formatting objects and properties or the transmitted document contains a reference to the source document(s) used in the construction of the document with the formatting objects and properties.

1.3 Formatting

Formatting interprets the result tree in its formatting object tree form to produce the presentation intended by the designer of the stylesheet from which the XML element and attribute tree in the “fo” namespace was constructed.

The vocabulary of formatting objects supported by XSL—the set of `fo:` element types—represents the set of typographic abstractions available to the designer. Semantically, each formatting object represents a specification for a part of the pagination, layout, and styling information that will be applied to the content of that formatting object as a result of formatting the whole result tree. Each formatting object class represents a particular kind of formatting behavior. For example, the block formatting object class represents the breaking of the content of a paragraph into lines. Other parts of the specification may come from other formatting objects; for example, the formatting of a paragraph (block formatting object) depends on both the specification of properties on the block formatting object and the specification of the layout structure into which the block is placed by the formatter.

The properties associated with an instance of a formatting object control the formatting of that object. Some of the properties, for example “color”, directly specify the formatted result. Other properties, for example “space-before”, only constrain the set of possible formatted results without specifying any particular formatted result. The formatter may make choices among other possible considerations such as esthetics.

Formatting consists of the generation of a tree of geometric areas, called the **area tree**. The geometric areas are positioned on a sequence of one or more pages (a browser typically uses a single page). Each geometric area has a position on the page, a specification of what to display in that area and may have a background, padding, and borders.

For example, formatting a single character generates an area sufficiently large enough to hold the glyph that is used to present the character visually and the glyph is what is displayed in this area. These areas may be nested. For example, the glyph may be positioned within a line, within a block, within a page.

Rendering takes the area tree, the abstract model of the presentation (in terms of pages and their collections of areas), and causes a presentation to appear on the relevant medium, such as a browser window on a computer display screen or sheets of paper. The semantics of rendering are not described in detail in this specification.

The first step in formatting is to “objectify” the element and attribute tree obtained via an XSLT transformation. Objectifying the tree basically consists of turning the elements in the tree into formatting object nodes and the attributes into property specifications. The result of this step is the **formatting object tree**.

As part of the step of objectifying, the characters that occur in the result tree are replaced by `fo:character` nodes. Characters in text nodes which consist solely of whitespace characters and which are chil-

dren of elements whose corresponding formatting objects do not permit `fo:character` nodes as children are ignored. Other characters within elements whose corresponding formatting objects do not permit `fo:character` nodes as children are errors. The first phase of the Unicode Bidirectional Algorithm is used to convert implicit Bidirectional markup to explicit nodes with the appropriate directional properties. Care is taken to insure that the explicit nodes so introduced are properly nested in the formatting object tree.

The content of the `fo:instream-foreign-object` is not objectified; instead the object representing the `fo:instream-foreign-object` element points to the appropriate node in the element and attribute tree.

Similarly any non-XSL namespace child element of `fo:declarations` is not objectified; instead the object representing the `fo:declarations` element points to the appropriate node in the element and attribute tree.

The second phase in formatting is to refine the formatting object tree to produce the **refined formatting object tree**. The refinement process handles the mapping from properties to traits.

This consists of: (1) shorthand expansion into individual properties, (2) mapping of corresponding properties, (3) determining computed values (may include expression evaluation), and (4) inheritance. Details on refinement are found in [**5 Property Refinement / Resolution**].

The third step in formatting is the construction of the area tree. The area tree is generated as described in the semantics of each formatting object. The traits applicable to each formatting object class control how the areas are generated. Although every formatting property may be specified on every formatting object, for each formatting object class, only a subset of the formatting prop-

erties are used to determine the traits for objects of that class.

1.4 Some test lists

An Ordered List:

1. The 1st item in this 1st level list.
2. The 2nd item in this 1st level list.
 - a. The 1st item in this 2nd level list.
 - b. The 2nd item in this 2nd level list.
 - i. The 1st item in this 3rd level list.
 - ii. The 2nd item in this 3rd level list.
 1. The 1st item in this 4th level list.
 2. The 2nd item in this 4th level list.
 3. The 3rd item in this 4th level list.
 4. The 4th item in this 4th level list.
 - iii. The 3rd item in this 3rd level list.
 - iv. The 4th item in this 3rd level list.
 - c. The 3rd item in this 2nd level list.
 - d. The 4th item in this 2nd level list.
3. The 3rd item in this 1st level list.
4. The 4th item in this 1st level list.

An Unordered List:

- The 1st item in this 1st level list.
- The 2nd item in this 1st level list.
 - o The 1st item in this 2nd level list.
 - o The 2nd item in this 2nd level list.
 - The 1st item in this 3rd level list.
 - The 2nd item in this 3rd level list.
 - The 1st item in this 4th level list.
 - The 2nd item in this 4th level list.
 - The 3rd item in this 4th level list.

- The 4th item in this 4th level list.
- The 3rd item in this 3rd level list.
- The 4th item in this 3rd level list.
- o The 3rd item in this 2nd level list.
- o The 4th item in this 2nd level list.
- The 3rd item in this 1st level list.
- The 4th item in this 1st level list.

A Definition List:

Dweeb

young excitable person who may mature into a *Nerd* or *Geek*

Hacker

a clever programmer

Nerd

technically bright but socially inept person