



# Extensible Stylesheet Language (XSL)

A modified introduction for testing purposes

## Abstract

XSL is a language for expressing stylesheets. It consists of two parts:

- a language for transforming XML documents, and
- an XML vocabulary for specifying formatting semantics.

An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary.

## Contents

1 Introduction and Overview.....	1
1.1 Processing a Stylesheet.....	2
1.1.1 Introduction to stylesheet processing, some definition of terms, and some other useful background information for the reader.....	2
1.1.2 Tree Transformations.....	2
1.1.3 Formatting.....	3
1.2 Benefits of XSL.....	5
1.2.1 Paging and Scrolling.....	5
1.2.2 Selectors and Tree Construction.....	6
1.2.3 An Extended Page Layout Model.....	6
1.2.4 A Comprehensive Area Model.....	7
1.2.5 Internationalization and Writing-Modes.....	7
1.2.6 Linking.....	7

## 1 Introduction and Overview

This specification defines the Extensible Stylesheet Language (XSL). XSL is a language for expressing stylesheets. Given a class of arbitrarily structured XML documents or data files, designers use an XSL stylesheet to express their intentions about how that structured content should be presented; that is, how the source content should be styled, laid out, and paginated onto some presentation medium, such as a window in a Web browser or a hand-held device, or a set of physical pages in a catalog, report, pamphlet, or book.

## 1.1 Processing a Stylesheet

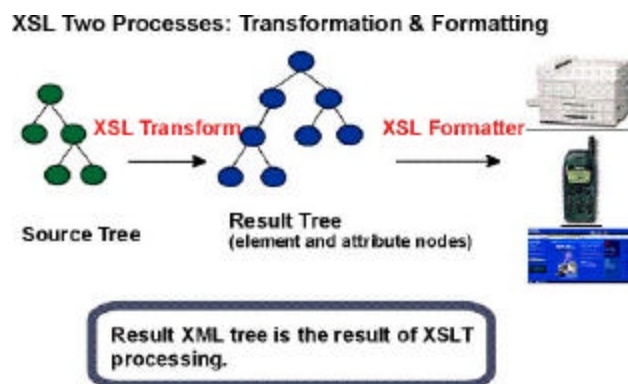
### 1.1.1 Introduction to stylesheet processing, some definition of terms, and some other useful background information for the reader

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 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.

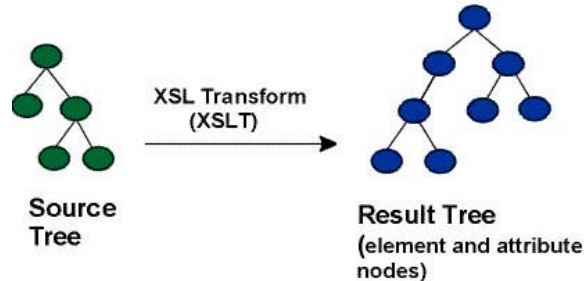


### 1.1.2 Tree Transformations

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. A diagram depicting this conceptual process is shown below.

## Transform to Another Vocabulary

With tree transformation, the structure of the result tree can be quite different from the structure of the source tree



In constructing the result tree, the source tree can be filtered and reordered, and arbitrary structure and generated content can be added.

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.

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 output. 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.

### 1.1.3 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 children 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 properties are used to determine the traits for objects of that class.

## 1.2 Benefits of XSL

Unlike the case of HTML, element names in XML have no intrinsic presentation semantics. Absent a stylesheet, a processor could not possibly know how to render the content of an XML document other than as an undifferentiated string of characters. XSL provides a comprehensive model and a vocabulary for writing such stylesheets using XML syntax.

This document is intended for implementors of such XSL processors. Although it can be used as a reference manual for writers of XSL style sheets, it is not tutorial in nature.

XSL builds on the prior work on Cascading Style Sheets and the Document Style Semantics and Specification Language. While many of XSL's formatting objects and properties correspond to the common set of properties, this would not be sufficient by itself to accomplish all the goals of XSL. In particular, XSL introduces a model for pagination and layout that extends what is currently available and that can in turn be extended, in a straightforward way, to page structures beyond the simple page models described in this specification.

### 1.2.1 Paging and Scrolling

Doing both scrollable document windows and pagination introduces new complexities to the styling (and pagination) of XML content. Because pagination introduces arbitrary boundaries (pages or regions on pages) on the content, concepts such as the control of spacing at page, region, and block boundaries become extremely important. There are also concepts related to adjusting the spaces between lines (to adjust the page vertically) and between words and letters (to justify the lines of text). These do not always arise with simple scrollable document windows, such as those found in today's browsers. However, there is a correspondence between a page with multiple regions, such as a body, header, footer, and left and right sidebars, and a Web presentation using “frames”. The distribution of content into the regions is basically the same in both cases, and XSL handles both cases in an analogous fashion.

XSL was developed to give designers control over the features needed when documents are paginated as well as to provide an equivalent “frame” based structure for browsing on the Web. To achieve this control, XSL has extended the set of formatting objects and formatting properties. In addition, the selection of XML source components that can be styled (elements, attributes, text nodes, comments, and processing instructions) is based on XSLT and XPath, thus providing the user with an extremely powerful selection mechanism.

The design of the formatting objects and properties extensions was first inspired by DSSSL. The actual extensions, however, do not always look like the DSSSL constructs on which they were based. To either conform more closely with the CSS2 specification or to handle cases more simply than in DSSSL, some extensions have diverged from DSSSL.

There are several ways in which extensions were made. In some cases, it sufficed to add new values, as in the case of those added to reflect a variety of writing-modes, such as top-to-bottom and bottom-to-top, rather than just left-to-right and right-to-left.

In other cases, common properties that are expressed in CSS2 as one property with multiple simultaneous values, are split into several new properties to provide independent control over independent aspects of the property. For example, the “white-space” property was split into four properties: a “space-treatment” property that controls how white-space is processed, a “linefeed-treatment” property that controls how line-feeds are processed, a “white-space-collapse” property that controls how multiple consecutive spaces are collapsed, and a “wrap-option” property that controls whether lines are automatically wrapped when they encounter a boundary, such as the

edge of a column. The effect of splitting a property into two or more (sub-)properties is to make the equivalent existing CSS2 property a “shorthand” for the set of sub-properties it subsumes.

In still other cases, it was necessary to create new properties. For example, there are a number of new properties that control how hyphenation is done. These include identifying the script and country the text is from as well as such properties as “hyphenation-character” (which varies from script to script).

Some of the formatting objects and many of the properties in XSL come from the CSS2 specification, ensuring compatibility between the two.

There are four classes of XSL properties that can be identified as:

1. CSS properties by copy (unchanged from their CSS2 semantics)
2. CSS properties with extended values (some of the extended values handle new semantics while others subsume semantics of one or more existing CSS property values, sometimes due to the need to provide writing-direction independent controls)
3. CSS properties broken apart and/or extended
4. XSL-only properties

### **1.2.2 Selectors and Tree Construction**

As mentioned above, XSL uses XSLT and XPath for tree construction and pattern selection, thus providing a high degree of control over how portions of the source content are presented, and what properties are associated with those content portions, even where mixed namespaces are involved.

For example, the patterns of XPath allow the selection of a portion of a string or the Nth text node in a paragraph. This allows users to have a rule that makes all third paragraphs in procedural steps appear in bold, for instance. In addition, properties can be associated with a content portion based on the numeric value of that content portion or attributes on the containing element. This allows one to have a style rule that makes negative values appear in “red” and positive values appear in “black”. Also, text can be generated depending on a particular context in the source tree, or portions of the source tree may be presented multiple times with different styles.

### **1.2.3 An Extended Page Layout Model**

There is a set of formatting objects in XSL to describe both the layout structure of a page or “frame” (how big is the body; are there multiple columns; are there headers, footers, or sidebars; how big are these) and the rules by which the XML source content is placed into these “containers”.

The layout structure is defined in terms of one or more instances of a “simple-page-master” formatting object. This formatting object allows one to define independently filled regions for the body (with multiple columns), a header, a footer, and sidebars on a page. These simple-page-masters can be used in page sequences that specify in which order the various simple-page-masters shall be used. The page sequence also specifies how styled content is to fill those pages. This model allows one to specify a sequence of simple-page-masters for a book chapter where the page instances are automatically generated by the formatter or an explicit sequence of pages such as used in a magazine layout. Styled content is assigned to the various regions on a page by associating the name of the region with names attached to styled content in the result tree.

In addition to these layout formatting objects and properties, there are properties designed to provide the level of control over formatting that is typical of paginated documents. This includes

control over hyphenation, and expanding the control over text that is kept with other text in the same line, column, or on the same page.

#### **1.2.4 A Comprehensive Area Model**

The extension of the properties and formatting objects, particularly in the area on control over the spacing of blocks, lines, and page regions and within lines, necessitated an extension of the CSS2 box formatting model. This extended model is described in [**4 Area Model**] of this specification. The CSS2 box model is a subset of this model. See the mapping of the CSS2 box model terminology to the XSL Area Model terminology in [**7.2 XSL Areas and the CSS Box Model**]. The area model provides a vocabulary for describing the relationships and space-adjustment between letters, words, lines, and blocks.

#### **1.2.5 Internationalization and Writing-Modes**

There are some scripts, in particular in the Far East, that are typically set with words proceeding from top-to-bottom and lines proceeding either from right-to-left (most common) or from left-to-right. Other directions are also used. Properties expressed in terms of a fixed, absolute frame of reference (using top, bottom, left, and right) and which apply only to a notion of words proceeding from left to right or right to left do not generalize well to text written in those scripts.

For this reason XSL (and before it DSSSL) uses a relative frame of reference for the formatting object and property descriptions. Just as the CSS2 frame of reference has four directions (top, bottom, left and right), so does the XSL relative frame of reference have four directions (before, after, start, and end), but these are relative to the “writing-mode”. The “writing-mode” property is a way of controlling the directions needed by a formatter to correctly place glyphs, words, lines, blocks, etc. on the page or screen. The “writing-mode” expresses the basic directions noted above. There are writing-modes for “left-to-right—top-to-bottom” (denoted as “lr-tb”), “right-to-left—top-to-bottom” (denoted as “rl-tb”), “top-to-bottom—right-to-left” (denoted as “tb-rl”) and more. See [**7.25.7 “writing-mode”**] for the description of the “writing-mode” property. Typically, the writing-mode value specifies two directions: the first is the inline-progression-direction which determines the direction in which words will be placed and the second is the block-progression-direction which determines the direction in which blocks (and lines) are placed one after another.

Besides the directions that are explicit in the name of the value of the “writing-mode” property, the writing-mode determines other directions needed by the formatter, such as the shift-direction (used for sub- and super-scripts), etc.

#### **1.2.6 Linking**

Because XML, unlike HTML, has no built-in semantics, there is no built-in notion of a hypertext link. In this context, “link” refers to “hypertext link” as defined in <http://www.w3.org/TR/html401/struct/links.html#h-12.1> as well as some of the aspects of “link” as defined in <http://www.w3.org/TR/xlink/#intro>, where “link is a relationship between two or more resources or portions of resources, made explicit by an XLink linking element”. Therefore, XSL has a formatting object that expresses the dual semantics of formatting the content of the link reference and the semantics of following the link.

#### **NOTE:**

During the CR period the XSL WG and Linking WG will jointly develop additional examples and guidance on how to use these formatting objects given XPointer and XLink XML source.

XSL provides a few mechanisms for changing the presentation of a link target that is being visited. One of these mechanisms permits indicating the link target as such; another allows for control over the placement of the link target in the viewing area; still another permits some degree of control over the way the link target is displayed in relationship to the originating link anchor.

XSL also provides a general mechanism for changing the way elements are formatted depending on their active state. This is particularly useful in relation to links, to indicate whether a given link reference has already been visited, or to apply a given style depending on whether the mouse, for instance, is hovering over the link reference or not.