(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 November 2004 (18.11.2004)

PCT

(10) International Publication Number
WO 2004/100015 A2

(51) International Patent Classification[7]: G06F 17/21

(21) International Application Number:
PCT/US2004/013885

(22) International Filing Date: 5 May 2004 (05.05.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/468,126 5 May 2003 (05.05.2003) US

(71) Applicant *(for all designated States except US)*: **ARBOR-TEXT, INC.** [US/US]; 1000 Victors Way, Ann Arbor, MI 48108 (US).

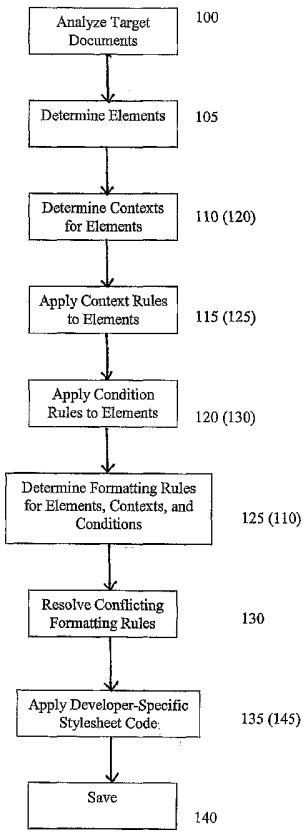
(71) Applicants and
(72) Inventors: **ALLEN, Timothy, P.**; 2640 Wildwood Trail, Saline, MI 48176 (US). **KOENIG, John, A.**; 626 Duane Ct., Ann Arbor, MI 48103 (US). **MALOUIN, Curt, M.**; 15701 Bradner, Rd., Northville, MI 48137 (US).

(71) Applicants and
(72) Inventors *(for all designated States except US)*: **FANG, Ying-Che**; 3169 Tiger Lily Ct., Ann Arbor, MI 48103 (US). **DOBROWOLSKI, Andrew, E.**; 3064 Wildwood, Ct., Saline, MI 48176 (US). **LLOYD, Johnson, J.**; 1029 Spring St., Ann Arbor, MI 48103 (US). **WHITE, David, L.**; 175 Templar, Pinckney, MI 48169 (US).

(71) Applicants and
(72) Inventors *(for all designated States except US)*: **GROSSO, Paul, B.**; 2100 Westlake, Dr., Austin, TX 78746 (US). **BARTLETT, Peter, G.**; 13302 Rainbow Dr., Gregory, MI 48137 (US). **LAMBERT, Robert, M.**; 1132 Rollingwood Arch, Virginia Beach, VA 23464 (US). **PIERCE, Kenneth, D.**; 227 Woodland, Hampton, VA 23669 (US).

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR DEFINING SPECIFICATIONS FOR OUTPUTTING CONTENT IN MULTIPLE FORMATS



(57) Abstract: A system and method for defining specifications for outputting content to multiple formats. More specifically, a system and method for creating a single multi-purpose stylesheet file from a single user interface, the single multi-purpose stylesheet being capable of designing multiple output media formats. For example, the multi-purpose stylesheet may convert input XML content to HTML, PDF, PostScript, WML, or other output formats. The method includes creating or modifying a multipurpose stylesheet containing formatting rules corresponding to various properties of different output media formats. The user can configure the multi-purpose stylesheet by selecting different formatting properties for different output media formats.

WO 2004/100015 A2

(71) **Applicants and**
(72) **Inventors** *(for all designated States except US)*: **BROW, Samuel, S.**; 598 Rosebud Ct., Saline, MI 48176 (US). **SVECHOTA, Joyce, M.**; 4932 Lohr Rd., Ann Arbor, MI 48108 (US).

(72) **Inventor; and**
(75) **Inventor/Applicant** *(for US only)*: **DREYSTADT, John, N.**; 2225 Sosna, Pickney, MI 48169 (US).

(74) **Agents: MILLS, Alicia, Esq.** et al.; Dorsey & Whitney LLP, Intellectual Property Department, Suite 1500, 50 South Sixth Street, Minneapolis, MN 55402-1498 (US).

(81) **Designated States** *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEM AND METHOD FOR DEFINING SPECIFICATIONS FOR OUTPUTTING CONTENT IN MULTIPLE FORMATS

FIELD OF THE INVENTION

[001] This invention relates generally to a system for defining specifications (or stylesheets) for outputting content in multiple formats. More specifically, it relates to a system and method for defining a single specification or multi-purpose stylesheet for outputting the content to multiple output media formats.

BACKGROUND OF THE INVENTION

[002] In today's content publishing environment, content may be generated and edited using a variety of editors, such as Microsoft Word, Web editors (e.g., Arbortext's Contributor), and Extensible Markup Language (XML) editors (e.g., Arbortext's Epic Editor). Similarly this content may be published to a variety of output media formats, including print, Portable Document Format (PDF), various forms of Hypertext Markup Language (HTML), Wireless Markup Language (WML), and PostScript. Content may also be published to compiled formats such as HTML-Help, MS Reader, formats for personal digital assistants (PDAs), and formats for mobile phones.

[003] Currently, designers and programmers create separate stylesheets to output content with a desired layout for each desired output media format. A stylesheet typically includes information such as the font and margins, the formatting layout, and other types of information that define appearance of the outputted document. A stylesheet may be used to convert content from one format to another. For example, it may convert XML content to one of the various output media formats specified above.

[004] Different stylesheets can be used to create different types of output for different types of devices, including differences in both content and layout. The need to perform this type of content tailoring is increasing rapidly as the popularity of new types of computing devices increases. One exemplary type is mobile devices, which typically have limited bandwidth, support fewer colors at a lower resolution and may not support audio content. Thus, XML documents originally created with the expectation that the document would be

rendered on a full-function desktop computer screen or in a printed form may contain a significant amount of content that can not or should not be presented to a mobile device.

[005] The stylesheets in the prior art are limited in that they are designed to generate output to only one specific output media format. If output to multiple formats is required, separate stylesheets must be created for each type of output. Along with the ever increasing number of display devices and display formats, there is an increasing need for a tool to automatically generate output to multiple output media formats. Thus, there is a need for a system for defining the specifications for outputting content to multiple output media formats, based on a single, multi-purpose stylesheet.

BRIEF SUMMARY OF THE INVENTION

[006] The present invention provides a method and system for defining the specifications for and creating a single multi-purpose stylesheet file from a single user interface ("UI"), the multi-purpose stylesheet being capable of designing multiple output media formats (multichannel outputs).

[007] Generally, the present invention provides a method and system for creating a single, multi-purpose stylesheet capable of accepting Extensible Markup Language (XML)-based content from multiple sources and outputting such content to multiple, user-configurable output media formats each with possible variations of style.

[008] More specifically, the invention may provide a multi-purpose stylesheet and associated editor that accepts an XML input document, or other input document, and appropriately configures an output document of a user-selectable type to present the XML input document in a specific format (colloquially referred to as "rules-based formatting"). The "user-selectable type" of document may be either file-specific, such as hypertext markup language (HTML), portable document format (PDF), Microsoft WORD, or other file types, or may be device-specific, such as printer, personal digital assistant (PDA), liquid crystal display (LCD), cellular telephone, and so on. Where the user-selectable type is file-specific, the invention formats the input

document for display as a computer-readable file. Where the user-selectable type is device-specific, the embodiment formats the input document for display on the specific device selected. Such formatting (either for file or device) is generally referred to as "styling." Styling is performed by applying one or more formatting rules, defined in a stylesheet, to the input.

[009] For example, in one embodiment of the invention, an input XML document is used to prepare or modify a multi-purpose stylesheet, the multi-purpose stylesheet being useful for outputting content to multiple output media formats. The invention opens or otherwise accesses a stylesheet which includes multiple styling/formatting rules. The multi-purpose stylesheet may apply a first rule to a portion of a document for outputting as an HTML document, but a second rule (different from the first) to the same portion of the document where a WORD output file is desired. Such rules may, for example, deal with the positioning of text on a display device when the output file is opened, the color of portions of the document, which graphic (if any) is displayed within a section of the output document when viewed, or whether a specific portion of the document is included in the output file.

[010] For example, the input file or document may include five versions of the same graphic file, or links to five external versions of the same graphic file, each graphic optimized for different viewer technology. The multi-purpose stylesheet created in accordance with the invention may choose the lowest possible resolution graphic for inclusion in an HTML output file, but select a higher resolution graphic for inclusion in a PDF or print file. Similarly, the graphic may be positioned at a certain location in the HTML file when viewed (for example, the upper left corner of the output file), but differently positioned for different output files such as PDF (for example, the middle of the output document).

[011] A further use of the multi-purpose stylesheet is to define page length for print/PDF outputs but eliminate page lengths from HTML outputs or other outputs that do not have page length limitations. Additionally, the multi-purpose stylesheet may include optional rules-based functionality not available for all output types. HTML outputs, for example, may permit absolute

positioning of text, graphics, or other objects either on a display device or within the output file, while WORD and/or PDF documents may not. The multi-purpose stylesheet may be configured to recognize such differences and apply appropriate rules to permit absolute positioning in an HTML document, but relative positioning in other output formats.

[012] One manner in which the present invention applies formatting rules is through the identification, creation and use of "style archetypes" which further automate the outputting of content in multiple formats. Style archetypes are models or prototypes used for processing of common publishing constructs like tables of contents, indexes, bulleted or numbered lists, hyperlinks, etc. Identification of an input element as an instance of a specific style archetype may automatically apply default formatting characteristics common to the specific archetype, and may affect the use of the input element in additional output formatting applications or affect the outputs of other elements assigned to style archetypes. Each element, or group of elements, in a document may be assigned a style archetype, thereby assigning the associated formatting rules and options to the element.

[013] Generally, the above discussion has been conducted with reference only to output file types. However, the same rules may be applied where the output is device-based. For example, with multiple graphics showing the same picture at different resolutions, an embodiment of the present invention may style output to a printer to include a higher-resolution graphic than output to a liquid crystal display (LCD) display.

[014] Further, a multi-purpose stylesheet in accordance with the present invention may include context-specific rules. That is, a block of text may be styled differently, depending on its hierarchical location in an input document. For example, a continuous text section may be indented once when the text occurs at an XML chapter level, but indented twice (or underlined, bolded, presented in all capital letters, omitted, and so forth) when the text occurs at a sub-chapter level.

[015] Still further, styling, in accordance with the present invention, may be user-specific in addition to output- and/or context-specific. When outputs are

presented to different users, each user may be assigned a level, class, context, security clearance, or other identifier. The user identifiers may be included in the stylesheet, so that when a user of a specific class or identifier accesses an output file or otherwise operates the stylesheet, the output is differently formatted than it may be for a user of a differing class or identifier.

[016] All of the above functionality is generally user-specifiable. In yet another embodiment of the present invention, such user specification may be accomplished through one or more graphic user interfaces (GUIs) specifically configured to simplify the creation of a multi-purpose stylesheet for a user.

[017] Further, the present invention allows the user to extend the formatting features of the multi-purpose stylesheet through editing, for example, manually, the specification of a particular element, context or condition (explained below) for a specific output media type or device. More specifically, the software allows the user to write code, for example, Extensible Stylesheet Language (XSL) or Formatting Output Specification Instance (FOSI), for a particular output. The code is stored in a .style file. During future use of the software, the code snippets may be managed and reused. Further, a library external to the .style file may be provided which allows the code to be used in multiple stylesheets. For example, user XSL code may be edited adding custom Dynamic HTML scripting for an HTML output. This code may be used to make all part-number defining content automatically hyperlinked to a specific page of a website.

[018] Thus, the present invention generally provides a system and method for creating a single multi-purpose stylesheet whereby using a single multi-purpose stylesheet abstraction, the baseline for formatting documents for use in publishing in multiple outputs may be defined once. Then each format or device specific output may have defined formatting that overrides the baseline formatting .

[019] While multiple embodiments are disclosed, still other embodiments of the present invention will become apparent to those skilled in the art from the following detailed description. As will be apparent, the invention is capable of modifications in various obvious aspects, all without departing from the spirit

and scope of the present invention. Accordingly, the drawings and detailed description are to be regarded as illustrative in nature and not restrictive.

BRIEF DESCRIPTION OF THE DRAWINGS

[020] Figure 1 is a flow chart illustrating a method for converting an input file to an output file in accordance with one embodiment of the present invention.

[021] Figure 2 is a screenshot of a user interface of an editor in accordance with one embodiment of the present invention.

[022] Figure 3 is a screenshot of a font dialog box for editing font properties in accordance with one embodiment of the present invention.

[023] Figure 4 is a screenshot of an indent dialog box for editing indent properties in accordance with one embodiment of the present invention.

[024] Figure 5 is a screenshot of a spacing dialog box for editing spacing properties in accordance with one embodiment of the present invention.

[025] Figure 6 is a screenshot of a breaks dialog box for editing breaks properties in accordance with one embodiment of the present invention.

[026] Figure 7 is a screenshot of an advanced breaks properties dialog box accessible through the breaks dialog box of Figure 6 for editing advanced breaks properties in accordance with one embodiment of the present invention.

[027] Figure 8 is a screenshot of a gentext dialog box for editing gentext properties in accordance with one embodiment of the present invention.

[028] Figure 9 is a screenshot of a sets dialog box for viewing sets in accordance with one embodiment of the present invention.

[029] Figure 10 is a screenshot of an edit property set dialog box for editing a property set in accordance with one embodiment of the present invention.

[030] Figure 11 is a screenshot of a new property set dialog box for adding a new property set in accordance with one embodiment of the present invention.

[031] Figure 12 is a screenshot of a user interface showing style archetype options of an editor in accordance with one embodiment of the present invention.

[032] Figure 13 is a screenshot of a user interface enabling editing of stylesheet source for an element in a context or under a defined condition in accordance with one embodiment of the present invention.

[033] Figure 14 is a screenshot of editing stylesheet file source code in accordance with one embodiment of the present invention.

[034] Figure 15 is a screenshot of assigning style characteristics to a combination of outputs in accordance with one embodiment of the present invention.

[035] Figure 16 is a screenshot of a condition dialog box in accordance with one embodiment of the present invention.

[036] Figure 17 is a screenshot of an attribute test for a new condition in accordance with one embodiment of the present invention.

[037] Figure 18 is a screenshot of user options for exporting to a specific stylesheet language for use in other applications in accordance with one embodiment of the present invention.

[038] Figure 19 is a graphical illustration of the architecture of a stylesheet editor in accordance with one embodiment of the present invention.

[039] Figure 20 illustrates Base Classes and Stylesheet Level Classes.

[040] Figure 21 illustrates IElementNode Subclasses.

[041] Figure 22 illustrates Property-related classes.

[042] Figure 23 illustrates IPropertyElementBase subclasses.

DETAILED DESCRIPTION

[043] Generally, the present invention provides a system for defining specifications (stylesheets) for outputting content in multiple formats. More specifically, the present invention provides a system for defining a single multi-purpose stylesheet capable of accepting an input file stored in one of a variety of file formats, converting the input file to a second format using the multipurpose stylesheet and outputting the file. As such, the software of the present invention enables publishing of a document (content of which may be obtained from one or more sources) in multi-channels using a single multi-purpose stylesheet. As with the input file, the output file may be formatted as

one of a variety of computer-readable file formats. The present invention provides for output- and/or context-specific styling based upon file type desired. The file type may be based upon output media format, including print, Portable Document Format (PDF), Hypertext Markup Language (HTML), Wireless Markup Language (WML), PostScript, etc. or may be based upon the output device. Exemplary output devices for use with an embodiment of the present invention include monitors, printers, televisions, liquid crystal display (LCD) screens, web tablets, cell phone displays, pagers, or any other devices capable of receiving a transmission (either wired or wireless) and displaying an alphanumeric or graphical output.

[044] In addition to output- and/or context- specific styling, the present invention may be used for user-specific styling. Thus, when outputs are presented to different users, each user may be assigned a level, class, context, security clearance, or other identifier. The user identifiers may be included in the multi-purpose stylesheet so that when a user of a specific class or identifier accesses an output file or otherwise operates the multi-purpose stylesheet, the output is differently formatted than it may be for a user of a differing class or identifier.

[045] Using the system and method of the present invention, it is not necessary to individually "style" or format all documents that use a same set of elements. Using a multi-purpose stylesheet of the present invention, all documents of the "document type" are formatted without any manual intervention.

[046] To style an input file, formatting rules are applied to the file or portions thereof. Exactly which formatting rules are applied to a given document portion (or "element") is discussed in more detail with respect to Figure 1. Generally, a "formatting rule" is a computer-executable instruction or specification indicating a certain output format, placement, font, graphic, and so forth to be applied to an element. In other words, a formatting rule controls and/or modifies the display of an element. Alternately, a formatting rule may define the interactions a user could engage such as a hyperlink or start/stop of animation.

[047] One manner in which the present invention applies formatting rules is through the identification, creation and use of "style archetypes" which further automate the outputting of content in multiple formats. Style archetypes may be used to affect the formatting and usage of element style assignments. Style archetypes capture the purpose of a particular element with regard to how it can be formatted in the document. Each style archetype determines process behavior in other areas of the stylesheet. Style archetypes are models or prototypes used for processing of common publishing constructs like tables of contents, indexes, bulleted or numbered lists, hyperlinks, etc. Identification of an input element as an instance of a specific style archetype may automatically apply default formatting characteristics common to the specific archetype, and may affect the use of the input element in additional output formatting applications or affect the outputs of other elements assigned to style archetypes. Each element, or group of elements, in a document may be assigned a style archetype, thereby assigning the associated formatting rules and options to the element.

[048] Figure 1 is a flowchart illustrating an exemplary process for styling an input file of a first format to a group of differing output files of a second format. In the present embodiment, input files in accordance with the Extensible Markup Language (XML) are accepted as input files. Generally, XML provides a useful, human and machine readable, standardized syntax for representing content, metadata, links, queries, transforms, and configuration information, which permits simplified processing of dynamic content. Document content (prose) can be created as XML natively, or transformed from proprietary applications to XML. Data from a variety of software applications including directly from many databases is accessible as XML. W3C Standards such as XML Linking Language (Xlink), XML Path Language (XPath), and Extensible Stylesheet Language (XSL) can be leveraged to enable application programmers and users with a standard, application interchangeable syntax.

[049] Input files of another file type (e.g., WORD, PDF, HTML, and so on) may be converted to an XML file prior to the styling process explained herein.

Although embodiments shown employ a separate computer-readable program or module to implement this conversion (such as Arbortext's Interchange), such a conversion process may be performed by a further embodiment of the present invention. In further embodiments, file types other than XML may be natively accepted as input files.

[050] As herein described, the present invention creates a multi-purpose stylesheet. In an exemplary embodiment, the invention creates an XML stylesheet , called a ".style" file. Optionally, the invention may generate from that .style file a standards compliant XSL or XSL Formatting Objects (XSL-FO) stylesheet file. XSL stylesheets include template rules, which define an input pattern and a template to use in creating an output result. When applying a stylesheet, the patterns in the templates are matched against the syntax of the XML source document. When a match is found with the template pattern, an output document fragment is created according to the actions specified in the template. The source document is parsed recursively, until no more matching patterns are found. The resulting document fragments are then aggregated to yield a complete output document. Of course other types of stylesheets may be created in accordance with the present invention.

[051] Generally, a style is associated with an element of the document. Context refers to the location (or "element selector") of the element within the input file Some elements use different properties depending on the context in which the element occurs. Thus, the present invention generally associates formatting properties with the element context or with the element directly. Further, "conditions" may be used to associate different formatting properties with the same context. A condition comprises a set of attribute tests, plus a collection of formatting properties.

[052] Referring to Figure 1, the styling process of the present invention begins in operation 100, where target documents are analyzed. One or more input files (or "documents") may be analyzed either by the invention or by a user of the invention to create a Document Type Definition (DTD) or XML schema having tags labeling each element.

[053] Figure 2 is a screen shot of a user interface 20 showing element tags 22, context tags 24, style tags 26, and output-based style editors 28. In operation 105 of Figure 1, the invention determines the location and type of each element in the input file. Generally, an "element" is a portion of a document corresponding to a specific file marker, tag, or header. The tags used to define elements are generally part of the input file, and may be either automatically generated as part of the input file or manually entered. Element tags are assigned to each "element" in the input. For example, as seen in Figure 2, the elements may be tagged "section," "subtitle," "table," "title, etc. Any suitable tag may be used and those listed and/or shown are intended to be illustrative only. XML input documents have tags natively assigned. If the input document is not an XML document, tags should otherwise be assigned. Once the appropriate file tags have been recognized and the file parsed into individual elements, each element is assigned to a specific element category. Again generally, these elements correspond to various XML elements.

[054] After the elements have been identified and assigned, the invention determines the specific formatting rules associated with each element in operation 125 of Figure 1. An "element" may be thought of as a portion of a document having a set of formatting properties. The formatting properties associated with the element depend on the context and/or condition associated with the element. Thus, before the specific formatting rules associated with the element can be determined, the invention determines the contexts and conditions of the elements and applies the associated context and condition rules, seen at operations 110, 115, and 120 of Figure 1. Contexts, conditions and rules associated with contexts and conditions are described more fully below.

[055] Each element of a specific type generally has identical formatting properties, although such properties may change in different contexts, as discussed in more detail below. For example, a block of text identified as a "para" element may have simple formatting rules, such as "always indent one tab space; hard paragraph return at end of element; 12 point Times New

Roman font;" and so on. The formatting rules define how the element will be displayed, absent any more specific formatting rules that may later be applied.

[056] Figures 3 through 11 are screenshots illustrating properties which may be formatted. Figure 3 shows a font dialog box 30 for editing font properties such as font, size, bold, italic, color, shading, all caps, small caps, super/subscript, underline, strikethrough, etc. Figure 4 shows an indent dialog box 40 for editing indent properties such as alignment, first line indent, left, right, etc. Figure 5 shows a spacing dialog box 50 for editing spacing properties such as line spacing, spacing before, spacing after, etc. Figure 6 shows a breaks dialog box 60 for editing breaks properties such as structure type, columns, hyphenation, etc. Figure 7 shows an advanced breaks properties dialog box 70 accessible through the breaks dialog box 60 of Figure 6 for editing advanced breaks properties such as the number of lines to keep at the top of a page, the number of lines to keep at the bottom of the page, keep content together, etc. Figure 8 shows a gentext (automatically generated text) dialog box 80 for editing gentext properties such as text to be added before an element, text to be added after an element, etc.

[057] Figure 9 shows a sets dialog box 90 that shows the property sets in the multi-purpose stylesheet (Available property sets 92) and the property sets referenced by the current object(s) (Used property sets 94). The Description field 96 displays the description of the selected property sets. Figure 10 shows a dialog box 150 for editing a property set. Figure 11 shows a dialog 160 box for adding a new property set.

[058] The process of the present invention provides using style archetypes for generating user interface changes/behavior. Identification, creation and use of style archetypes further automates the outputting of content in multiple formats. Each style archetype determines process behavior in other areas of the stylesheet. Within the stylesheet, style archetypes, having a defined set of formatting rules, are assigned to each element, based on the tag. For example, as seen in Figure 2, the style archetype assigned may be "Title," "Table," etc. Usage of the style archetype assignments affects the formatting and usage of other element style assignments. Style archetypes are models or prototypes

used for processing of common publishing constructs like tables of contents, indexes, bulleted or numbered lists, hyperlinks, etc. Identification of an input element as an instance of a specific style archetype may automatically apply default formatting characteristics common to the specific archetype, and may affect the use of the input element in additional output formatting applications or affect the outputs of other elements assigned to style archetypes.

[059] As seen in Figure 12, associated with each style archetype 26 are formatting options such as font, indent, spacing, breaks, gentext, and sets. Of course, other formatting options may be associated with each style archetype. As discussed in relation to Figure 2, each element is assigned a style archetype, thereby assigning the associated formatting rules and options to the element. Using an example of three input elements: identifying an input element as a Division applies default formatting of a forced line or page break before the start of the Division and after the end of the Division; identifying another element as a Title archetype applies line breaks, and font bolding; identifying a third element as a Table of Contents archetype applies basic outline formatting and may automatically or at user choice allow for the inclusion or exclusion of the Titles of Divisions as entries in the output of the Table of Contents. Once style archetypes have been defined, they can be modified or extended to create specialized behaviors for different users or different output media. A conventional-looking table of contents in a printed document, for example, might be represented in a completely different form on a Web browser or a PDA. Style archetypes capture the purpose of a particular element with regard to how it is formatted in a document. For example, Titles in Divisions and Formal Blocks may be identified as having specific formatting in the location in a document but may also appear in Table of Contents, List of Figures, or List of Tables. Each element, or group of elements, in a document may be assigned a style archetype, thereby assigning the associated formatting rules and options to the element.

[060] The assigned style characteristic may be based on the output. A baseline characteristic is assigned to all outputs unless an output specific override assigns a specific characteristic. The output specific overrides may

be editing specific (e.g., Epic Editor, Contributor (web editing), or Companion (Word 2003 editing)). Alternately, the output specific overrides may be publishing specific (e.g., HTML file (flat file), HTML Help (MS help Compiler), Web (folder of many HTML files, index and TOC), or Print/PDF). The formatting rules for an element may not be associated directly with the element, but instead with the element context, as discussed in more detail below. In addition, each element, may have a "default context." This default context is the same for each element, and contains the base formatting rules for the element.

[061] Further, style characteristics may be assigned to a single output target or to a combination of outputs. Figure 15 illustrates a screen shot of an Edit Uses/Outputs List dialog box 180 for creating a combination of outputs for assigning a style characteristic. As shown, the combination of outputs is Companion and Contributor. This enables the same style characteristic to be assigned to an element whether the output is ultimately Companion or Contributor. Of course, any combination of outputs may be selected. This capability allows users to combine outputs whose properties are generally the same. When outputs are combined, they can later be separated so changes can be made specific to one output. While they are combined, changes that are made are made to each output. Further, users may define their own outputs and add the output to the outputs list 182.

[062] In the example embodiment, each element is always assigned to a style. Generally, an element may be mapped to only a single style, although alternative embodiments may permit mapping an element to multiple styles. In such an embodiment, potentially conflicting formatting rules may arise. Such conflicts may be resolved (at operation 130 of Figure 1, discussed in more detail below) by applying the most specific set of style rules, by determining the most appropriate style rules based on the element context, or by applying user-implemented criteria designed to resolve formatting rules conflicts.

[063] As previously mentioned, a user may modify formatting rules associated with a given style. When a user modifies a style's formatting rules, all elements associated with that style are appropriately updated.

[064] Returning to Figure 1, in operation 110, the invention determines the various contexts in which each element will appear in an output. Context is defined by the combination of an element selector and the formatting properties used for elements that match the aforementioned selector. Alternatively, "context" may refer to the element selector alone. Contexts generally correspond closely with templates in XSL Stylesheets and elements-in-context in Formatting Output Specification Instance (FOSI) stylesheets.

[065] Formatting rules are generally associated with contexts, rather than elements, in order to ensure that the proper formatting ruleset for a given context is applied to an element without conflicting with any element-specific formatting rules. The context varies depending on the location of the element within the input file (the "element selector.") Generally, with reference to an XML input file, elements may be nested in sub-elements. For example, a "title" element may be located within a "book" element, a "chapter" element, an "abstract" element, or a "figure" element, among others. Each of these combinations is a different element selector for the title element, and accordingly each may have a different set of formatting rules.

[066] Referring to Figure 2, the element "title" can be seen at position 23. Listed below the title element 23 are each of the contexts 24 for that element, including, for example, "title in section," "title in appendix," "title in figure," "title in article," etc. Selecting the element title will select title in each of the contexts appearing thereunder and any properties applied will be applied to title in every context. Selecting one of the contexts, however, will allow properties to be applied only to title in that context.

[067] Once the context of the element is determined, in operation 115 of Figure 1, the stylesheet applies these context-specific formatting rules to each element. Thus, depending on the context of the title element, the style in which the title is displayed may vary. For example, the title element in a book context may display in 24 point font, centered, while the title element in a

figure context may display in bold, blue text with a 10 point font and right justified. Accordingly, the context rules applied to the element are determined on a context-by-context basis.

[068] When multiple contexts may apply to a single element, only the most specific context formatting rules will be applied. Thus, at operation 130 of Figure 1, formatting rules are resolved. To return to the above example, a title element within an abstract element, which is in turn located in a book element, has two valid contexts: "title in abstract" and "title in book." (It should be noted that the contexts given are illustrative only.) The title element is nested immediately within the abstract. By contrast, although the title is located within the book, there is at least one intervening element, namely the abstract. Accordingly, in this example, the "title in abstract" formatting rules are more specific, and will be applied. The "title in book" formatting rules will not be applied.

[069] Thus, in accordance with the present invention, formatting properties are specified for elements. An element may have just one collection of formatting properties that applies to all occurrences of the element in each document or an element may specify different properties for different contexts. Only one context is used for a given occurrence of an element.

[070] In a specific embodiment, contexts are mapped to eics in FOSIs. Contexts are generally built such that they can be translated to FOSIs as standard eics using context and occur attributes. In XSL, contexts are implemented as XSL:template element match attributes. Output specific properties are handled by Styler emitting different XSL for different options.

[071] In some situations it may be desirable to have different formatting properties used for the same context, based on attribute tests. For this purpose, the present invention permits associating "conditions" with a context. A condition comprises a set of attributes tests, plus a collection of formatting properties. If all of the tests are true (or a specified set of tests is true), the properties are overlaid on the properties specified for the context.

[072] In operation 120 of Figure 1, the embodiment applies condition-based formatting rules to an element. A "condition," as used herein, refers to at least

one attribute test plus at least one formatting rule. As explained above, if the attribute test is met, then the formatting rule is applied.

[073] Figure 16 is a screenshot illustrating a dialog box 190 for creating a new condition. The Condition dialog box 190 shown may be launched under the Elements -> New Condition menu item or in the context menu in the element list. Figure 16 illustrates the dialog box 190 that appears when a user clicks new or selects a test and clicks to edit.

[074] There are two parts to a condition, the tests that determine whether a condition is true and the formatting properties that are overlaid on the context's formatting properties when the condition is true. Attribute tests may be added to the multi-purpose stylesheet by the user. Figure 17 is a screenshot of a dialog box 200 for assigning an attribute test for a new condition. A single condition may have multiple attribute tests associated therewith. In such a case, a user may specify when creating the condition whether all attribute tests must be met to apply the associated formatting rule or rules, or whether only a single test must be met. In Boolean terms, the user may specify whether the attribute test is an "AND" or "OR" style test.

[075] Conditions apply to all outputs. However, a user can specify output specific formatting properties to be applied when a condition is true. For example, the user may add a condition to test an attribute and, in the formatting properties for the condition, select a specific output (e.g., HTML file) in the Properties to edit list, then specify the properties to be overlaid. Such a condition will only affect the selected output. Similarly, the user may add a condition to test an attribute and, in the formatting properties for the condition, make different settings for different outputs.

[076] Generally, conditions vary from contexts. Unlike contexts, multiple condition formatting rules may be applied to a single element. If an element has multiple applicable conditions, formatting rules for each condition met are applied to the element. A user may specify an order for condition tests- a first condition may be initially checked, then a second condition, and so on.

[077] Even though multiple conditions may apply formatting rules to a single element, it may happen that two or more conditions may attempt to apply

mutually contradictory formatting rules. If formatting rules for different applied conditions conflict, later conditions in the user-specified order overrides conflicting formatting rules for earlier conditions in the order. Alternately, the invention randomly selects which formatting rules to apply. Again, at operation 130 of Figure 1, conflicting formatting rules are resolved.

[078] Although more specific context formatting rules override less specific context formatting rules, the same is not necessarily true of class- or output-based rules. For example, an input file styled to an output file for a specific display device, in a specific file format, to be viewed by a user of a specific class, may have all three sets of output formatting rules applied (namely, class-, output-, and file-formatting rules.) In the event that various types of formatting rules (i.e., class-, output-, and/or file-formatting rules) conflict with one another, a user may resolve such conflicts during stylesheet creation. Alternatively, the invention may choose one ruleset as the dominant formatting ruleset. In such a case, the user may be presented with a prompt or warning indicating the choice.

[079] Using contexts and conditions, the user can specify precise "rule-based formatting" that produces the desired appearance in printed and web pages, as well as in Epic editor for whole classes of documents. Property sets may be used to simplify maintenance of the multi-purpose stylesheet. Property set dialog boxes are illustrated in Figures 9, 10 and 11. Figure 9 illustrates a dialog box 90 for applying a property set from a list of available property sets 92. Figure 10 illustrates a dialog box 150 for editing a property set and Figure 11 illustrates a dialog box 160 for adding a new property set. A property set is a named set having specific style properties. For example, if ten elements are configured to be 10pt, italic, and red, it may be useful to create a property set having those style properties. Thus, the property set may be called WarningFont with the associated style properties being 10pt, italic, and red. To apply the style properties to the ten elements, WarningFont is applied to each element. An advantage of using the property set is that, if it is later desired that all ten elements be 11pt, italic, and red, the user need only change the font size property for WarningFont, not for each of the ten elements.

[080] The present invention allows the user to edit the source stylesheet code for a given element in context and/or condition for any given output. More specifically, the software allows the user to write stylesheet language specific code such as XSL for a particular output. The custom stylesheet code is stored in the .style file. During future use of the software, the custom stylesheet code may be managed and reused. Further, a library external to the .style file may be provided which allows use of custom stylesheet code in multiple stylesheets. Custom stylesheet code (including, for example, javascript) is added, as at block 135of Figure 1. The output results of the multi-purpose stylesheet include formatting characteristics of the multi-purpose stylesheet (unless deleted/modified by the developer) as well as the custom stylesheet code.

[081] The process of the present invention further enables developer customizations for specific elements/contexts and outputs while maintaining Styler control of all other elements/contexts/outputs. The multi-purpose stylesheet allows extended customization beyond formatting. Editing the element source enables developers to provide additional capabilities while still allowing all other elements and outputs to be modified by the multi-purpose stylesheet. As shown in Figure 13, to edit the element source, the element source is identified (e.g., XSL-HTML file, XSL-FO, XSL-HTML Help, XSL-Web, XSL-Contributor, or FOSI). An example of editing an XSL-HTML file source is shown in Figure 14. In Figure 14, the "copyright" element 170 of an HTML File output is edited.

[082] Figure 18 illustrates a dialog box 200 showing user options for exporting to a specific output. Based upon the multi-purpose stylesheet (a single .style file), multiple, target-specific stylesheet languages may be exported. Thus, a single user interface and a single .style file may be used to output, for example, XSLT for a flat HTML file, XSLT for an MS HTML Help file, XSLT for Web output, XSL-FO for Print output, FOSI (DOD standard for SGML formatting) for print and editing in Epic Editor, XSLT for editing in Contributor, .wstyle for editing in Companion (add-in to MS Word

2003), or other outputs. The listed outputs are intended to be illustrative, not limiting.

[083] Finally, the output styling process terminates in end operation 140 such as a save operation.

[084] Although the above operations are listed in a specific order, it should be understood that an embodiment of the present invention may omit certain operations, add others, and/or change the order in which operations are executed. For example, links may be determined and managed prior to applying styles, or user-based formatting rules may be applied before output-based formatting rules. Accordingly, the aforementioned operations are illustrative of the manner of operation of a single embodiment of the present invention, and should not be construed as limiting all embodiments.

[085] Generally, the aforementioned styling procedure is accomplished by the present invention by applying a multi-purpose stylesheet to an input file. The multi-purpose stylesheet contains a variety of formatting rules, conditions, contexts, styles, and so forth that may operate on the input file. The multi-purpose stylesheet contents may be user-specified or may be a default specified by the embodiment.

[086] A user may create, edit, or otherwise modify a multi-purpose stylesheet in accordance with the present invention through the use of a graphical user interface (GUI) associated with an embodiment of the present invention. The GUI Layer may use the Single Document Interface (SDI) for the main window. Normally, an SDI application comprises four visible objects: the main frame window (CFrameWnd), a toolbar (CtoolBar), the view (Cview), and a status bar at the bottom (CstatusBar). In the embodiment shown in Figure 2, 12 and 13, Styler's main window base class, StylerFrameWnd, derives from the MFC CframeWnd class. Thus, as shown, Styler comprises three splitter dialog windows. The splitter windows, StylerTreeView 210, StylerDescriptionView 212, and StylerPropView 214 are derived from StylerView which is a subclass of MFC's CformView class.

[087] StylerTreeView 210 displays the top splitter window where the element 22 and styler 26 tree controls are displayed. StylerDescriptionView

212 displays the middle splitter window where the description text is displayed. Typically, the information displayed in StylerDescriptionView 212 is read-only text description information of the currently selected item(s) in the top tree controls. StylerPropView 214 displays the bottom splitter window where the properties are displayed. The StylerPropView 214 manages displaying and modifying of properties associated with the selected item(s) in the top tree controls. Of course, while reference is made to top, middle, and bottom, these are intended as examples to be illustrative and are not intended as limiting.

[088] In use, the user selects an item(s) (e.g., an element, context, or condition) in tree control in the top view, StylerTreeView 210. The tab page currently displayed in the bottom view, StylerPropView 214, shows the resolved values which apply to the selected tree control items. A user may change a value, either by selecting a new value within a dropdown or by hand-entering in an edit box.

[089] When elements are shown, the Properties to edit: dropdown list 216 is visible above the property tabs. When property sets are shown, this control disappears. In the embodiment shown in Figure 2, the list 216 initially includes <Edit this list>, Base (all outputs), Epic Editor, HTML File, HTML Help, Print/PDF, Web, Word XML, and E3 Editor. Base (all outputs) is generally selected by default. When Base (all outputs) is selected, the user sees the common settings in the property tabs and any property changes made will apply as base settings for all outputs. When a different output is selected, the user still sees the common settings except where there is a different setting for the selected output, in which case, that setting is shown.

[090] When a user first creates a multi-purpose stylesheet in accordance with the present invention, the elements are assigned the style "Unstyled." The first phase of creating the new multi-purpose stylesheet is mapping the elements to "styles" such as "division", "title", "numbered list", "list item", "paragraph", etc. Using these mappings, Styler knows what contexts need to be created for an element, and can also assign default properties. Simply mapping all elements to their correct styles creates a surprisingly effective stylesheet in

very little time. Mapping the elements to a "style" may be done by clicking in the editor to select the element to be styled. The user may start at the beginning of the document and work their way through or may skip over some sections and return to it later. Generally, it is useful to style all the ancestors of the section being worked on.

[091] The user may then tweak the formatting properties on an element basis. In this situation, the user does not expand each element and select individual contexts but instead selects the collapsed elements and makes changes that apply to all contexts. The user may then select individual contexts to make select formatting properties specific to the contexts. Further, the user may add conditions to various contexts. The user may preview the result in different outputs and continue making modifications until satisfied with the results.

[092] The user then saves the multi-purpose stylesheet as a .style file. For future use, when the user edits documents of that document type, the .style file saved is used to output the documents as specified in the multi-purpose stylesheet.

[093] The Styler user interface is especially suited for a Windows platform. However, styler multi-purpose stylesheets may be used for editor display or composition on Unix or other suitable platform.

[094] Figure 19 illustrates the architecture 230 of an embodiment of the stylesheet editor (Styler) of the present invention. The architecture shown is intended to be illustrative of suitable architecture for the multi-purpose stylesheet editor and is not intended to be limiting. The central repository for the stylesheet data is an XML document 232 (.style). The software design is based around the reading and writing of data to and from the stylesheet document 232. The User Interface (the GUI Layer 234) communicates with the XML document 230 via a set of portable C++ classes (the Portable Layer 236) which serve as an interface to the Document Object Model (DOM) document. The Export operations 238 communicate with the same document 212 using ACL or AOM.

[095] In a Windows system, the Portable Layer 236 and the GUI Layer 234 make up a library. In a Unix system, the Portable Layer 236 makes up the library. Generally, the library is standalone as well as accessible via Epic.

[096] The Styler architecture 230 shown is designed such that the Portable Layer 236 is distinct and separate from the GUI Layer 234. The Styler architecture 230 facilitates support for multiple window systems. The Portable Layer 236 may reside in a standalone DLL (styler.dll) on Windows, as a shared object (styler.so) on Unix, or as otherwise suitable. The source files for the Portable Layer 236 may comprise general purpose C++ classes. Figures 20, 21, 22 and 23 illustrate example classes for use with the present invention.

[097] Figure 20 illustrates Base Classes and Stylesheet Level Classes. Base classes are classes which are generally never intended to be unsubstantiated, although they may not all be abstract classes with any pure virtual functions. The IStylesheet class is an IElementBase subclass which represents the main XML file controlling the Styler application. It exists on the system as an XML file using the Styler doctype. It persists as the doctype element named "styler."

[098] Figure 21 illustrates IElementNode Subclasses. The IElementNodeBase class is derived from IElementBase and represents element objects that are nodes in the GUI layer's tree context. These classes apply to elements, contexts, and conditions (described below). It is an abstract base class and provides fundamental methods such as a Resolve Method that is shared by objects in Styler's main tree control. The IElement class is a subclass of IElementNodeBase. Both the GUI classes and the export classes use the IElement object as the interface between the styler GUI classes and the styler doc (typically the DOM and/or ACL). It persists in the multi-purpose stylesheet as the Element element whose name attribute specifies an element name in the user's document or doctype. It may contain a list of IContext objects.

[099] Figure 22 illustrates Property-related classes. The IPropertiedBase class is a subclass of IElementNodeBase for objects represented in the Styler

DTD as an element which contains BaseProperties. It is an abstract base class for element-based objects and provides fundamental methods and properties specific to elements which contain the BaseProperties element. The IOutputPropertiedBase class is a subclass of IPropertiedBase for objects which are represented in the Styler DTD as an element which has BaseProperties and a list of OutputProperties. It is an abstract base class for element-based objects and provides fundamental methods and properties specific to elements which may contain both the BaseProperties element and the OutputProperties element.

[0100] The IContext class is a subclass of IOutputPropertiedBase. Both the GUI classes and the export classes use the Icontextobject as the interface between the styler GUI classes and the styler doc (typically DOM and/or ACL). It persists in the stylesheet as the Context element whose selector attribute specifies an XML context to which the child properties apply. It may contain a list of ICondition objects. It contains two IProperties objects represented in the DTD as BaseProperties and OutputProperties.

[0101] The ICondition class is a subclass of IOutputPropertiedBase. Both the GUI classes and the export classes use the ICondition object as the interface between the styler GUI classes and the styler doc (typically the DOM and/or ACL). It persists in the multi-purpose stylesheet as the Condition element whose AttributeTest and XpathTest elements specify an XML condition which further define and refine the contexts to which the child properties apply. It contains two IProperties objects represented in the DTD as BaseProperties and OutputProperties.

[0102] Figure 23 illustrates IPropertyElementBase subclasses. This class represents an abstract base class for single Styler property objects which are DTD elements. As Portable Layer objects they exist as containers for individual attribute lists which are individual style properties. For the GUI, they typically exist simply as containers for attributes which exist in controls.

[0103] Users may modify the content of a multi-purpose stylesheet to include custom values, elements, styles, contexts, and conditions.

[0104] The Portable Layer 236 communicates with the DOM to retrieve and/or send data for display or manipulation in the GUI layer's various dialogs. The GUI Layer 234 contains a pointer to the Portable Layer's IStylesheet, thereby allowing each view access to the Portable Layer's functions. Direct access to these functions facilitates population and maintenance of data used in controls of the GUI Layer 234. The Styler architecture 230 permits the portable functionality of Styler to function on other platforms besides Windows.

[0105] The multi-purpose stylesheet editor, in a specific embodiment, is a Unicode application which supports multiple languages. In a specific embodiment, Styler is compiled as 16-bit (via a _UNICODE compile setting) only. In this embodiment, the Portable Layer 236 uses DOM strings (16-bit) almost exclusively. Also in this embodiment, the GUI Layer 234 uses TCHAR and Cstrings for string operations when required by MFC controls and/or classes. Or course, other settings and/or strings may be used as suitable.

[0106] The conversion of an input file to an output file, using Styler, may include, among other options, determining a context-specific, output device-specific, or user-specific format. That is, a multi-purpose stylesheet in accordance with the present invention may alter the format or style of the output file depending on the output source or device, the output file format, the context of data or files contained either within the input or within a hierarchical structure including the input file, or by user accessing the output file or for which the output file is intended.

[0107] Formatting properties ("properties") are style settings represented by attributed values on the elements defined in the PropertyElements entity. These elements are children of the BaseProperties and OutputProperties elements. The BaseProperties element is a child of Context, Condition, and PropertySet. The OutputProperties element is a child of Context and Condition.

[0108] For contexts and conditions, during the composition process, first the base properties are applied. Then, if there is an OutputProperties element

whose output attribute matches the output being composed, the output properties are overlaid.

[0109] For any given element of a document, composition follows a process to resolve what set of formatting properties to apply to that element. The composer determines which Context of the current element is the best match based on comparing its selector with where the element occurs in the document. That Context specifies the properties to use for the element. The properties may be specified directly via children of the Context element, or may be derived from PropertySets that are references by the children of the Context element. Additionally, the Contexts may contain Conditions which have Tests and properties. If a Condition's Tests are True, its properties are merged with the Context's. Of course, the Condition's properties may also be specified directly via children of the Condition element, or may be derived from PropertySets that are referenced by the children of the Condition element.

[0110] Extensible Markup Language (XML) is a meta-markup language that provides a format for describing structured data. XML is a standard language for storing, viewing, and manipulation Web-based data and provides a uniform method for describing and exchanging structured data in an open, text-based format. XML allows information content to be totally separated from the mechanisms for rendering/displaying that content. Compared to HTML, which is a display-oriented markup language, XML is a general purpose language for representing structured data without including information that describes how to format the data for display.

[0111] XML is similar to HTML in that it is a tag-based language. By virtue of its tag-based nature, XML defines a strict tree structure or hierarchy. XML allows the creation of custom tags to describe both proprietary data and business logic. XML "elements" are structural constructs that consist of a start tag, an end or close tag, and the information or content that is contained between the tags. A "start tag" is formatted as "<tagname>" and an "end tag" is formatted as "</tagname>". In an XML document, start and end tags can be nested within other start and end tags. All elements that occur within a

particular element must have their start and end tags occur before the end tag of that particular element. This defines a strict tree-like structure. Each element forms a node in this tree, and potentially has "child" or "branch" nodes. The child nodes represent any XML elements that occur between the start and end tags of the "parent" node.

[0112] An XML schema is a description of the structure of an XML document. More specifically, an XML schema is a method for specifying constraints on XML documents using an XML-based language. Typically an XML schema is written in XSL Schema Definition (XSD) and support namespaces and data types. Schemas are founded on XML and are, thus, hierarchical. Schemas enable easy creation of an unambiguous specification and determination of the scope over which a comment is meant to apply.

[0113] Within each schema, a "dictionary" of element names is defined. The dictionary of element names defined by a schema is referred to as a "namespace." Within an XML document, element names are qualified by namespace identifiers. When qualified by a namespace identifier, a tag name appears in the form "[namespace]:[tagname]". This model enables the same element name to appear in multiple schemas, or namespaces, and for instances of these duplicate element names to appear in the same XML document without colliding. Start tags can declare an arbitrary number of "attributes" which declare "property values" associated with the element being declared. Attributes are declared within the start tag using the form "<[tagname][attribute1][attribute2] . . . [attributeN]>", where an attribute1 through attributeN are declarations of an arbitrary number of tag attributes. Each attribute declaration is of the form "[attributeName]=["attributeValue"]" where each attribute is identified by a unique name followed by an "=" character, followed by the quoted value of the attribute.

[0114] The Styler schema comprises several main sections: top level element; definition of parameter entities; StyleSheetInfo; PropertySets; Elements; Styles; PrintedPageSets; TableOfContents; ReferenceFormats; and FosiRsrcDescSource.

[0115] StyleSheetInfo is the meta-data for the stylesheet. The Style Sheet Information section of the schema contains stylesheet meta-data. A "Stylesheet Identification Processing Instruction" at the top of each styler file specifies: "title," what is displayed for the style sheet in Arbortext Editor wherever style sheets are listed; and "composition types," the outputs for which the stylesheet can be used. By specifying this information, data is not replicated in the schema. A stylerVersion attribute contains the styler version at the time the stylesheet was created, or was last updated. A currentVersion attribute contains the current version of Styler. If the stylerVersion is < the currentVersion, styler automatically updates if necessary and changes the stylerVersion. MinorVersion and currentMinorVersion attributes keep in-house stylesheets up to date during development. Element sets in the StyleSheetInfo have no effect on what documents or document types the stylesheet may be used for. They enable the Styler User Interface to make editing easier by listing elements and attributes in the UI were appropriate.

[0116] PropertySets are building blocks which may be referenced from within the Elements section in order to modularize formatting properties values. A PropertySet is a set of properties that can be referenced from multiple places within the stylesheet (anywhere there is a propertySetReferences attribute). PropertySets allow stylesheets to be modularized. PropertySets can reference other PropertySets.

[0117] Formatting properties, or properties for short, are style settings represented by attribute values on the elements defined in the propertyElements entity. These elements are children of the "BaseProperties" and "OutputProperties" elements. The BaseProperties element is a child of "Context", "Condition", and "PropertySet". The OutputProperties element is a child of "Context" and "Condition". For contexts and conditions, during the composition process, first the base properties are applied, and then, if there is an OutputProperties element whose output attribute matches the output being composed, the output properties are overlaid. An individual formatting property can be specified or left unset. This is represented in the stylesheet file by whether or not the attribute representing the property is present.

[0118] For any given element of a document, composition follows a process to resolve what set of formatting properties to apply to that element. This process is as follows. The composer determines which Context of the current element is the best match based on comparing its selector with where the element occurs in the document. That Context specifies the properties to use for the element. The properties may be specified directly via children of the Context element, or may be derived from PropertySets that are referenced by the children the Context element. Additionally, the Contexts can contain Conditions which have Tests and properties. If a Condition's Tests are True, its properties are merged with the Context's. Note that the Condition's properties also can be specified directly via children of the Condition element, or may be derived from PropertySets that are referenced by the children of the Condition element. If any properties are still unset at the end of this merging, one of two things happens. 1. If the property is "inheritable", then the property is inherited from, i.e. will be the same as for, the parent element in the document instance; 2. If the property is not inheritable, then it is given a default value by Styler.

[0119] Elements contains formatting rules for elements, most commonly including rules for all elements of a given DTD or schema. The Elements section is the core of the stylesheet. The rules for elements may specify output specific formatting properties, and may vary depending on element context, attribute values, etc. Elements are mapped to styles. The element's style determines the available styles, what contexts are created initially for each style, and what properties those contexts have initially.

[0120] For some elements, it is desirable to use different properties depending on the context the element occurs in, so Styler associates formatting properties with element "contexts", rather than directly with elements. "Context" is the term used in Styler to specify the combination of an element selector and a set of formatting properties to use for elements that match that selector. The "selector" is specified using an XSL pattern. (Patterns are certain types of XPath expressions.) Only one context for a given instance of an element in a document, the one whose selector best matches the element's context in the

document, is processed. If selector is omitted, it implies an XSL pattern that is just the element name.

[0121] PrintPageSets defines page layout for print and PDF output.

[0122] TablesOfContents is a definition of tables of contents that may be referenced from the Elements section. For each Table of Contents defined by the user, there is a TableOfContents element. This element defines various properties of the Table of Contents by its attributes. Each TableOfContents element has five TocLevelIndent children, one for each TOC level. Those elements control indenting and number positioning of table of contents entries.

[0123] ReferenceFormats includes global definitions of formats used for cross-reference and references to divisions from headers and footers.

[0124] FosiRsrcDescSource contains optional FOSI rsrcdesc source markup that may be merged into an exported FOSI.

[0125] An exemplary schema for the present invention is attached as Appendix A.

[0126] Although the present invention has been described with reference to preferred embodiments, persons skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

APPENDIX A

An exemplary schema for Styler is disclosed below.

```
<!-- ** Top level Element -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Styler">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element minOccurs="0" ref="StyleSheetInfo"/>
    <xsd:element minOccurs="0" ref="PropertySets"/>
    <xsd:element ref="Elements"/>
    <xsd:element minOccurs="0" ref="UserFormattingElements"/>
    <xsd:element minOccurs="0" ref="StylerFormattingElements"/>
    <xsd:element minOccurs="0" ref="Styles"/>
    <xsd:element minOccurs="0" ref="Bullets"/>
    <xsd:element minOccurs="0" ref="ReferenceFormats"/>
    <xsd:element minOccurs="0" ref="Indexes"/>
    <xsd:element minOccurs="0" ref="PrintPageSets"/>
    <xsd:element minOccurs="0" ref="TablesOfContents"/>
    <xsd:element minOccurs="0" ref="Tables"/>
    <xsd:element minOccurs="0" ref="FosiRsrcDescSource"/>
   </xsd:sequence>
   <xsd:attribute name="singleElement"/>
   <xsd:attribute name="singlePageSet"/>
   <xsd:attribute name="webEditorOutput">
    <xsd:simpleType>
     <xsd:restriction base="xsd:token">
      <xsd:enumeration value="yes"/>
      <xsd:enumeration value="no"/>
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:attribute>
   <xsd:attribute name="wordStyleVersion"/>
  </xsd:complexType>
 </xsd:element>
 <!--
  If Styler.singleElement is set, only the Element of that name
  is transformed into the exported stylesheet
 -->
 <!--
  If Styler.singlePageSet is set, only the PrintPageSet of that name
  is transformed into the exported stylesheet
 -->
 <!--
  If Styler.webEditorOutput is 'yes', the exported stylesheet
```

```
 is designed to be used with the web editor
-->
<!--
 If Styler.wordStyleVersion is supplied, the exported stylesheet
 is valid ONLY for use with the ATI Word plugin
-->
<!-- ** Parameter Entities Used in Content Models -->
<!-- Elements whose attributes contain formatting properties -->
<xsd:complexType name="propertyElements">
 <xsd:sequence>
  <xsd:element minOccurs="0" ref="Hidden"/>
  <xsd:element minOccurs="0" ref="Font"/>
  <xsd:element minOccurs="0" ref="Highlight"/>
  <xsd:element minOccurs="0" ref="Indent"/>
  <xsd:element minOccurs="0" ref="Alignment"/>
  <xsd:element minOccurs="0" ref="LineSpacing"/>
  <xsd:element minOccurs="0" ref="SideBySide"/>
  <xsd:element minOccurs="0" ref="SpaceBefore"/>
  <xsd:element minOccurs="0" ref="SpaceAfter"/>
  <xsd:element minOccurs="0" ref="Breaks"/>
  <xsd:element minOccurs="0" ref="Hyphenation"/>
  <xsd:element minOccurs="0" ref="WidowOrphan"/>
  <xsd:element minOccurs="0" ref="Keep"/>
  <xsd:element minOccurs="0" ref="Float"/>
  <xsd:element minOccurs="0" ref="GeneratedText"/>
 </xsd:sequence>
</xsd:complexType>
<!-- ** Parameter Entities Used in Attribute Declarations -->
<xsd:simpleType name="yesorno">
 <xsd:restriction base="xsd:token">
  <xsd:enumeration value="yes"/>
  <xsd:enumeration value="no"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="character">
 <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:simpleType name="integer">
 <xsd:restriction base="xsd:NMTOKEN"/>
</xsd:simpleType>
<xsd:simpleType name="outputs">
 <xsd:restriction base="xsd:token">
  <xsd:enumeration value="editor"/>
  <xsd:enumeration value="printpdf"/>
  <xsd:enumeration value="htmlfile"/>
  <xsd:enumeration value="htmlhelp"/>
```

```
      <xsd:enumeration value="web"/>
      <xsd:enumeration value="wordxml"/>
      <xsd:enumeration value="contributor"/>
     </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="numberStyles">
     <xsd:restriction base="xsd:token">
      <xsd:enumeration value="1"/>
      <xsd:enumeration value="A"/>
      <xsd:enumeration value="a"/>
      <xsd:enumeration value="I"/>
      <xsd:enumeration value="i"/>
     </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="followWithValues">
     <xsd:restriction base="xsd:token">
      <xsd:enumeration value="tab"/>
      <xsd:enumeration value="space"/>
      <xsd:enumeration value="emspace"/>
      <xsd:enumeration value="nothing"/>
     </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="styleNames">
     <xsd:restriction base="xsd:token">
      <xsd:enumeration value="block"/>
      <xsd:enumeration value="crossReference"/>
      <xsd:enumeration value="definitionList"/>
      <xsd:enumeration value="definitionListItem"/>
      <xsd:enumeration value="division"/>
      <xsd:enumeration value="document"/>
      <xsd:enumeration value="formalBlock"/>
      <xsd:enumeration value="graphic"/>
      <xsd:enumeration value="hidden"/>
      <xsd:enumeration value="index"/>
      <xsd:enumeration value="indexTermElementModel"/>
      <xsd:enumeration value="indexTermAttributeModel"/>
      <xsd:enumeration value="inline"/>
      <xsd:enumeration value="link"/>
      <xsd:enumeration value="linkTarget"/>
      <xsd:enumeration value="bulletedList"/>
      <xsd:enumeration value="numberedList"/>
      <xsd:enumeration value="listItem"/>
      <xsd:enumeration value="paragraph"/>
      <xsd:enumeration value="preformatted"/>
      <xsd:enumeration value="table"/>
      <xsd:enumeration value="tableOfContents"/>
```

```
      <xsd:enumeration value="title"/>
      <xsd:enumeration value="nostyle"/>
      <xsd:enumeration value="unstyled"/>
     </xsd:restriction>
    </xsd:simpleType>
    <!--
     sizes are expressed in four parts:

      * an optional plus or minus sign; plus is the default

      * a number, with or without a decimal; up to two
       decimal places may be used

      * optional whitespace

      * a units abbreviation; allowable values are:
         pt, pc, in, mm, cm, em, px
       standing for
         points, picas, inches, millimeters, centimeters,
         ems, and pixels
       respectively

       an em is a distance equal to the current font size; (when used
       within the value of the font size property itself, it refers
       to the parent element's font size)

       there are 72.27 points to an inch, 12 points to a pica

     examples of valid sizes: 1pc  -10pt  +1.33 em
    -->
    <xsd:simpleType name="size">
     <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <!-- Styler stores colors as #rrggbb values for now. -->
    <xsd:simpleType name="color">
     <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <!--
     Left indent is a size as described above with an optional
     @ character prefix which means relative to parent element's
     left indent; otherwise, an absolute distance (from current
     left margin) is assumed.

     Firstline indent is a size as described above with an optional
     prefix that can be * or ^. * means relative to the context's
     left indent.  ^ means relative to the parent element's first
```

```
line indent. Otherwise, an absolute distance (from current
left margin) is assumed.

Right indent is a size as described above with an optional
prefix that can be * or @. * means relative to the context's
left indent. @ means relative to the parent element's right
indent. Otherwise, an absolute distance (from current
right margin) is assumed.
-->
<xsd:simpleType name="leftIndent">
 <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:simpleType name="firstLineIndent">
 <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:simpleType name="rightIndent">
 <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<!--
 ** Parameter entities for additional information
 added by Styler when "exporting"
-->
<xsd:simpleType name="added-foClass-names">
 <xsd:restriction base="xsd:token">
  <xsd:enumeration value="hidden-block"/>
  <xsd:enumeration value="hidden-inline"/>
  <xsd:enumeration value="graphic-block"/>
  <xsd:enumeration value="graphic-inline"/>
  <xsd:enumeration value="link-block"/>
  <xsd:enumeration value="link-inline"/>
  <xsd:enumeration value="preformatted-block"/>
  <xsd:enumeration value="preformatted-inline"/>
  <xsd:enumeration value="paragraph-block"/>
  <xsd:enumeration value="paragraph-inline"/>
  <xsd:enumeration value="page-sequence"/>
  <xsd:enumeration value="block-landscape"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="foClassNames">
 <xsd:union memberTypes="styleNames added-foClass-names"/>
</xsd:simpleType>
<xsd:attributeGroup name="added-attrs-for-exporting">
 <xsd:attribute name="foClass" type="foClassNames"/>
 <xsd:attribute name="blockOrInline">
  <xsd:simpleType>
   <xsd:restriction base="xsd:token">
```

```
            <xsd:enumeration value="block"/>
            <xsd:enumeration value="inline"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="hidden" type="yesorno"/>
      <xsd:attribute name="startNew">
        <xsd:simpleType>
          <xsd:restriction base="xsd:token">
            <xsd:enumeration value="column"/>
            <xsd:enumeration value="page"/>
            <xsd:enumeration value="oddPage"/>
            <xsd:enumeration value="evenPage"/>
            <xsd:enumeration value="none"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="newPageSet"/>
      <xsd:attribute name="startLandPage" type="yesorno"/>
      <xsd:attribute name="align">
        <xsd:simpleType>
          <xsd:restriction base="xsd:token">
            <xsd:enumeration value="left"/>
            <xsd:enumeration value="center"/>
            <xsd:enumeration value="right"/>
            <xsd:enumeration value="justify"/>
            <xsd:enumeration value="preformatted"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="bulletedListMarker"/>
      <xsd:attribute name="divisionLevel" type="integer"/>
    </xsd:attributeGroup>
    <!--
    ========================================================================
    ========== -->
    <!--
      ** schema.info: Style Sheet Information

      This section contains style sheet meta-data. Note that a
      "Stylesheet Identification Processing Instruction" at top
      of each styler file specifies:

        * Title - what is displayed for the style sheet in
          Arbortext Editor wherever style sheets are listed
        * Composition Types - The outputs for which the stylesheet
```

can be used.

so that data is not replicated in the schema.

The stylerVersion attribute contains the styler version at the time the stylesheet was created, or was last updated for.

The currentVersion attribute contains the current version of Styler. If the stylerVersion is < the currentVersion, styler will make automatic updates if necessary and change the stylerVersion.

The minorVersion and currentMinorVersion attributes play a similar role, though they're primarily intended for keeping in-house stylesheets up to date during development.

The element sets have no effect on what documents or document types the stylesheet can be used for, but they enable the Styler UI to make editing easier by listing elements and attributes in the UI where appropriate.

```
-->
<xsd:element name="StyleSheetInfo">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="ElementSet">
    <xsd:attribute name="stylerVersion" use="required"/>
    <xsd:attribute name="currentVersion" default="5">
     <xsd:simpleType>
      <xsd:restriction base="xsd:string">
       <xsd:enumeration value="5"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="minorVersion" default="0"/>
    <xsd:attribute name="currentMinorVersion" default="3">
     <xsd:simpleType>
      <xsd:restriction base="xsd:string">
       <xsd:enumeration value="3"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="printPdfEngine" use="required">
     <xsd:simpleType>
      <xsd:restriction base="xsd:token">
       <xsd:enumeration value="XSL-FO"/>
```

```
            <xsd:enumeration value="FOSI"/>
          </xsd:restriction>
         </xsd:simpleType>
        </xsd:attribute>
       </xsd:extension>
      </xsd:complexContent>
     </xsd:complexType>
    </xsd:element>
    <!--

    If stylesheet is created from a document with a DTD, then that
    DTD's public and system IDs are stored with the stylesheet,
    and the document path is not. If the document references a
    Schema, either in addition to, or instead of a DTD, then the
    Schema's location is stored with the stylesheet. If the
    stylesheet is created from a free-form document, its path is
    stored with the stylesheet. Currently these are just for
    informational purposes. In any given Styler session, it's the
    document that Styler was launched from, and its DTD or Schema
    if it has one that is used whenever we need document or DTD
    information.

    In the current release, only 1 ElementSet is allowed. It is
    anticipated that support for multiple element sets will be
    added in the future, along with better support for namespaces.

    When we create a stylesheet for a DTD or Schema that as a
    default namespace declared, save its URI so that XSL
    generation can use it.

    -->
    <xsd:complexType name="ElementSet">
     <xsd:sequence>
      <xsd:element ref="ElementSet"/>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="ElementSet">
     <xsd:complexType>
      <xsd:attribute name="dtdSystemId"/>
      <xsd:attribute name="dtdPublicId"/>
      <xsd:attribute name="schemaLocation"/>
      <xsd:attribute name="documentPath"/>
      <xsd:attribute name="namespaceURI"/>
     </xsd:complexType>
    </xsd:element>
```

```
<!--
=============================================================
========== -->
<!--
** schema.properties: Formatting Properties

Formatting properties, or properties for short, are style
settings represented by attribute values on the elements
defined in the propertyElements entity. These elements are
children of the "BaseProperties" and "OutputProperties"
elements. The BaseProperties element is a child of "Context",
"Condition", and "PropertySet". The OutputProperties element
is a childen of "Context" and "Condition".

For contexts and conditions, during the composition process,
first the base properties are applied, and then, if there is
an OutputProperties element whose output attribute matches
the output being composed, the output properties are overlaid.

An individual formatting property can be specified or left
unset. This is represented in the stylesheet file by whether
or not the attribute representing the property is present.

For any given element of a document, composition follows a
process to resolve what set of formatting properties to apply
to that element. This process is as follows:

The composer determines which Context of the current element
is the best match based on comparing its selector with where
the element occurs in the document. That Context specifies the
properties to use for the element. The properties may be
specified directly via children of the Context element, or may
be derived from PropertySets that are referenced by the
children the Context element.

Additionally, the Contexts can contain Conditions which have
Tests and properties. If a Condition's Tests are True, its
properties are merged with the Context's. Note that the
Condition's properties also can be specified directly via
children of the Condition element, or may be derived from
PropertySets that are referenced by the children of the
Condition element.

If any properties are still unset at the end of this merging,
one of two things happens.
```

1. If the property is "inheritable", then the property is inherited from, i.e. will be the same as for, the parent element in the document instance.

2. If the property is not inheritable, then it is given a default value by Styler.

Here is a slightly more detailed explanation of this.

The ordered steps to produce a merged property list from the context's properties, its referenced property sets, and the context's conditions are as follows:

* For starters, we start with a blank property set where every property is considered "unset" or "unspecified".

* A context is selected.

* All property sets referenced by the Context's BaseProperties's element are merged in the order referenced. For each property set being merged, any property sets that it's BaseProperties element references are first merged, in the order referenced. Then the properties specified by children of the BaseProperties element are merged. What merging means for a property is that any property that is specified, becomes the current value for that property in the property set, overriding any previous value.

* Then the Context's properties are merged into the result of the previous step - again any property that is specified, becomes the current value for that property in the property set, overriding any previous value. First the base properties for all ouputs are merged, then any overrides for the output currently being composed are merged.

* Then each condition is evaluated, and if true, its properties are merged into the result of the previous step. First the base properties for all ouputs are merged, then any overrides for the output currently being composed are merged.

* Then inheriting and defaulting occur as described above.

Whether or not a property is inheritable is inherent to the given property and is indicated in comments below.

Below, properties are grouped in a logical manner that

```
generally reflects the grouping in the UI. There is no
semantic to the grouping (such as there is for FOSI
categories).
-->
<!-- Container for properties that apply to all outputs -->
<xsd:element name="BaseProperties">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="propertyElements">
    <xsd:attribute name="propertySetReferences" type="xsd:IDREFS"/>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
<!--
Container for properties that apply to a single output -
properties specified under OutputProperties override
those specified for BaseProperties when there is a conflict
-->
<xsd:element name="OutputProperties">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="propertyElements">
    <xsd:attribute name="output" use="required" type="outputs"/>
    <xsd:attribute name="propertySetReferences" type="xsd:IDREFS"/>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
<!--
If an element is hidden, that means it does not show up or
take any space and all other properties are irrelevant except
for GeneratedText; GeneratedText has effect for the outermost
element that is hidden, but not for any children of that element;
The hidden property is NOT inherited, though it does affect
all descendants
-->
<xsd:element name="Hidden">
 <xsd:complexType>
  <xsd:attribute name="hidden" type="yesorno"/>
 </xsd:complexType>
</xsd:element>
<xsd:simpleType name="fontWeights">
 <xsd:restriction base="xsd:token">
  <xsd:enumeration value="normal"/>
  <xsd:enumeration value="bold"/>
```

```
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="Font">
     <xsd:complexType>
      <xsd:attribute name="family"/>
      <xsd:attribute name="size"/>
      <xsd:attribute name="weight" type="fontWeights"/>
      <xsd:attribute name="italic" type="yesorno"/>
     </xsd:complexType>
    </xsd:element>
    <!--
     All properties in the Font property group are inherited
     font family can be: Serif, Sanserif, Monospace, or a specific
     family name like Arial, Helvetica, etc.;  font size can be
     an ordinary %size; value, or one of the following:
      xx-small
      x-small
      small
      medium
      large
      x-large
      xx-large
      larger
      smaller
     or a number followed by the percent sign
    -->
    <xsd:element name="Highlight">
     <xsd:complexType>
      <xsd:attribute name="color" type="color"/>
      <xsd:attribute name="shading" type="color"/>
      <xsd:attribute name="underline" type="yesorno"/>
      <xsd:attribute name="strikeThrough" type="yesorno"/>
      <xsd:attribute name="overstrike" type="yesorno"/>
      <xsd:attribute name="overstrikeCharacter" type="character"/>
      <xsd:attribute name="applyEffects">
       <xsd:simpleType>
        <xsd:restriction base="xsd:token">
         <xsd:enumeration value="wordsOnly"/>
         <xsd:enumeration value="wordsAndSpaces"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="allcaps" type="yesorno"/>
      <xsd:attribute name="smallcaps" type="yesorno"/>
      <xsd:attribute name="script">
       <xsd:simpleType>
```

```
            <xsd:restriction base="xsd:token">
              <xsd:enumeration value="normal"/>
              <xsd:enumeration value="super"/>
              <xsd:enumeration value="sub"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
      </xsd:complexType>
    </xsd:element>
    <!--
      All properties in the Highlight property group are inherited

      NOTES:
        * in this release:
          there is no UI for overstrike and overstrikeCharacter
          but they can be edited in by hand; strikethrough is
          ignored if overstrike is specified.
        * overstrikeCharacter defaults to nothing
        * applyEffects controls how underline,
          strikeThrough, and overstrike work with regard to spaces
        * if script is super or sub, the font size is
          automatically adjusted and the font is raised
          or lowered
        * if both allcaps and smallcaps are specified, smallcaps
          is effectively ignored

    -->
    <xsd:element name="Indent">
      <xsd:complexType>
        <xsd:attribute name="leftIndent" type="leftIndent"/>
        <xsd:attribute name="firstLineIndent" type="firstLineIndent"/>
        <xsd:attribute name="rightIndent" type="rightIndent"/>
        <xsd:attribute name="lastLineRightIndent" type="size"/>
      </xsd:complexType>
    </xsd:element>
    <!--
      All properties in the Indent property group are inherited

      NOTE: lastLineRightIndent is not presented in the property tabs UI.
      It is set only for contexts of _sfe:TocEntryN elements and in the
      first release, is always set to 0pt.  It is relative to parent's
      right indent.

    -->
    <xsd:element name="Alignment">
      <xsd:complexType>
```

```
<xsd:attribute name="align">
 <xsd:simpleType>
  <xsd:restriction base="xsd:token">
   <xsd:enumeration value="left"/>
   <xsd:enumeration value="center"/>
   <xsd:enumeration value="right"/>
   <xsd:enumeration value="justify"/>
   <xsd:enumeration value="preformatted"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="lastLineAlign">
 <xsd:simpleType>
  <xsd:restriction base="xsd:token">
   <xsd:enumeration value="left"/>
   <xsd:enumeration value="center"/>
   <xsd:enumeration value="right"/>
   <xsd:enumeration value="justify"/>
   <xsd:enumeration value="relative"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:attribute>
</xsd:complexType>
</xsd:element>
<!--
 All properties in the Alignment property group are inherited.
 The semantic default (initial value) for lastLineAlign is 'relative".
-->
<xsd:element name="LineSpacing">
 <xsd:complexType>
  <xsd:attribute name="lineHeight" type="size"/>
 </xsd:complexType>
</xsd:element>
<!-- All properties in the LineSpacing property group are inherited -->
<xsd:element name="SideBySide">
 <xsd:complexType>
  <xsd:attribute name="definitionListRole">
   <xsd:simpleType>
    <xsd:restriction base="xsd:token">
     <xsd:enumeration value="list"/>
     <xsd:enumeration value="item"/>
     <xsd:enumeration value="term"/>
     <xsd:enumeration value="definition"/>
     <xsd:enumeration value="none"/>
    </xsd:restriction>
   </xsd:simpleType>
```

```
    </xsd:attribute>
   </xsd:complexType>
  </xsd:element>
  <!--
   All properties in the SideBySide property group are NOT inherited
   This property element says that for the given context or
  -->
  <xsd:simpleType name="precedence">
   <xsd:restriction base="xsd:token">
    <xsd:enumeration value="none"/>
    <xsd:enumeration value="low"/>
    <xsd:enumeration value="medium"/>
    <xsd:enumeration value="high"/>
    <xsd:enumeration value="force"/>
   </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="SpaceBefore">
   <xsd:complexType>
    <xsd:attribute name="minimum" type="size"/>
    <xsd:attribute name="optimum" type="size"/>
    <xsd:attribute name="maximum" type="size"/>
    <xsd:attribute name="conditionality">
     <xsd:simpleType>
      <xsd:restriction base="xsd:token">
       <xsd:enumeration value="keep"/>
       <xsd:enumeration value="discard"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="precedence" type="precedence"/>
   </xsd:complexType>
  </xsd:element>
  <!--
   All properties in the SpaceBefore property group are NOT inherited.
   If minimum or maximum is omitted, it is effectively the same as
   the optimum.
  -->
  <xsd:element name="SpaceAfter">
   <xsd:complexType>
    <xsd:attribute name="minimum" type="size"/>
    <xsd:attribute name="optimum" type="size"/>
    <xsd:attribute name="maximum" type="size"/>
    <xsd:attribute name="precedence" type="precedence"/>
   </xsd:complexType>
  </xsd:element>
  <!--
```

```
All properties in the SpaceAfter property group are NOT inherited .
If minimum or maximum is omitted, it is effectively the same as
the optimum.
-->
<xsd:simpleType name="numberOfColumns">
 <xsd:restriction base="xsd:token">
  <xsd:enumeration value="1"/>
  <xsd:enumeration value="2"/>
  <xsd:enumeration value="3"/>
  <xsd:enumeration value="4"/>
  <xsd:enumeration value="5"/>
  <xsd:enumeration value="6"/>
  <xsd:enumeration value="7"/>
  <xsd:enumeration value="8"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:element name="Breaks">
 <xsd:complexType>
  <xsd:attribute name="blockOrInline">
   <xsd:simpleType>
    <xsd:restriction base="xsd:token">
     <xsd:enumeration value="block"/>
     <xsd:enumeration value="inline"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="startNew">
   <xsd:simpleType>
    <xsd:restriction base="xsd:token">
     <xsd:enumeration value="column"/>
     <xsd:enumeration value="page"/>
     <xsd:enumeration value="oddPage"/>
     <xsd:enumeration value="evenPage"/>
     <xsd:enumeration value="none"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="spanInto">
   <xsd:simpleType>
    <xsd:union memberTypes="numberOfColumns">
     <xsd:simpleType>
      <xsd:restriction base="xsd:token">
       <xsd:enumeration value="noChange"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:union>
```

```
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="startLandPage" type="yesorno"/>
    <xsd:attribute name="newPageSet"/>
    <xsd:attribute name="pageNumber">
     <xsd:simpleType>
      <xsd:restriction base="xsd:token">
       <xsd:enumeration value="continue"/>
       <xsd:enumeration value="initial"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:attribute>
   </xsd:complexType>
  </xsd:element>
  <!--
   All properties in the Breaks property group are NOT inherited

   If blockOrInline is "inline", then no other properties will have any effect.

   The spanInto property corresponds to the Columns control. It should be
   blank (which implies noChange) if startNew is Column. noChange is
   the default.

   The startLandPage property (start landscape page region), can only be "yes"
   if startNew is page, oddPage, or evenPage.

   The newPageSet property can be the name of a page set defined in the
   stylesheet or #noChange.

   Default value for pageNumber is continue.
  -->
  <xsd:element name="Hyphenation">
   <xsd:complexType>
    <xsd:attribute name="hyphenate" type="yesorno"/>
    <xsd:attribute name="language"/>
   </xsd:complexType>
  </xsd:element>
  <!-- All properties in the Hyphenation property group are inherited -->
  <xsd:element name="WidowOrphan">
   <xsd:complexType>
    <xsd:attribute name="widowCount" type="integer"/>
    <xsd:attribute name="orphanCount" type="integer"/>
   </xsd:complexType>
  </xsd:element>
  <!-- All properties in the WidowOrphan property group are inherited -->
  <xsd:simpleType name="keepValue">
```

```
<xsd:restriction base="xsd:token">
 <xsd:enumeration value="0"/>
 <xsd:enumeration value="1"/>
 <xsd:enumeration value="2"/>
 <xsd:enumeration value="3"/>
 <xsd:enumeration value="4"/>
 <xsd:enumeration value="5"/>
 <xsd:enumeration value="6"/>
 <xsd:enumeration value="7"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="keepScopes">
 <xsd:restriction base="xsd:token">
  <xsd:enumeration value="withinLine"/>
  <xsd:enumeration value="withinColumn"/>
  <xsd:enumeration value="withinPage"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="keepFloatsOut">
 <xsd:restriction base="yesorno"/>
</xsd:simpleType>
<xsd:element name="Keep">
 <xsd:complexType>
  <xsd:attribute name="together" type="keepValue"/>
  <xsd:attribute name="withNext" type="keepValue"/>
  <xsd:attribute name="withPrevious" type="keepValue"/>
  <xsd:attribute name="keepScope" type="keepScopes"/>
  <xsd:attribute name="floatsOut" type="keepFloatsOut"/>
 </xsd:complexType>
</xsd:element>
<!--
 All properties in the Keep property group are NOT inherited,
 though the effect of keepTogether does affect all
 non-floating descendants
-->
<!-- Float is a future - not supported in first pass -->
<xsd:simpleType name="floatType">
 <xsd:restriction base="xsd:token">
  <xsd:enumeration value="none"/>
  <xsd:enumeration value="pageTop"/>
  <xsd:enumeration value="pageBottom"/>
  <xsd:enumeration value="pageTopOrBottom"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:element name="Float">
 <xsd:complexType>
```

```
      <xsd:attribute name="float" type="floatType"/>
      <xsd:attribute name="floatWidth">
       <xsd:simpleType>
        <xsd:restriction base="xsd:token">
         <xsd:enumeration value="column"/>
         <xsd:enumeration value="page"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:attribute>
     </xsd:complexType>
    </xsd:element>
    <!--
     All properties in the Float property group are NOT inherited,
     though the effect of floating does affect all decendants
    -->
    <xsd:element name="GeneratedText">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element minOccurs="0" ref="AddBefore"/>
       <xsd:element minOccurs="0" ref="AddAfter"/>
       <xsd:element minOccurs="0" ref="ReferenceFormats"/>
       <xsd:element minOccurs="0" ref="TocEntries"/>
       <xsd:element minOccurs="0" ref="RepeatingTitle"/>
       <xsd:element minOccurs="0" ref="Numbering"/>
       <xsd:element minOccurs="0" ref="Bulleting"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="AddBefore" type="any"/>
    <xsd:element name="AddAfter" type="any"/>
    <!--
     note that ReferenceFormats can appear in two places:

     1. at the top level within Styler it includes the global or default
     formats for each reference format; there must be a ReferenceFormat
     element for every reference format at this level; this collection
     determines the list that _gte:Xref elements are allowed to reference

     2. within the GeneratedText of a context or condition
     ReferenceFormats indicates the referencability of an element, and
     if any ReferenceFormat elements are present, they are local
     overrides; if ReferenceFormats in GeneratedText is not
     present, then references to that element are undefined; if
     ReferenceFormats is present, but for a particular format, there is
     no ReferenceFormat element, then that format uses the default
     format; (future) for formats that do have a ReferenceFormat element, the
```

```
      content specifies the format to use - this overrides the
      default value; if there is a ReferenceFormat element that is empty, it
      means that that format generates nothing for that context or
      condition
 5    -->
      <xsd:element name="ReferenceFormats">
       <xsd:complexType>
        <xsd:sequence>
         <xsd:element minOccurs="0" maxOccurs="unbounded"
10  ref="ReferenceFormat"/>
        </xsd:sequence>
       </xsd:complexType>
      </xsd:element>
      <xsd:element name="ReferenceFormat">
15     <xsd:complexType>
        <xsd:complexContent>
         <xsd:extension base="any">
          <xsd:attribute name="referenceFormatName" use="required"
    type="xsd:NMTOKEN"/>
20       </xsd:extension>
        </xsd:complexContent>
       </xsd:complexType>
      </xsd:element>
      <xsd:element name="TocEntries">
25     <xsd:complexType>
        <xsd:sequence>
         <xsd:element minOccurs="0" maxOccurs="unbounded" ref="TocEntry"/>
        </xsd:sequence>
       </xsd:complexType>
30    </xsd:element>
      <xsd:element name="TocEntry">
       <xsd:complexType>
        <xsd:sequence>
         <xsd:element ref="TocEntryContents"/>
35       <xsd:element minOccurs="0" ref="TocLevelIndent"/>
        </xsd:sequence>
        <xsd:attribute name="tocNameReference" use="required"
    type="xsd:NMTOKEN"/>
        <xsd:attribute name="tocLevel" use="required" type="integer"/>
40     </xsd:complexType>
      </xsd:element>
      <xsd:element name="TocEntryContents" type="any"/>
      <xsd:element name="RepeatingTitle" type="any"/>
      <!--
45     The Numbering property group is valid only for elements
      that are mapped to one of the following styles:
```

title
listItem

Its properties control the behavior and format of numbering, but there still has to be a number formatting-element in the contents of the element's GeneratedText for a number to appear, and for these properties to have any effect.

The Numbering element for a title context defines the number format and style to be used for all references to the title's parent, e.g. the context is chapter/title, defines the numbering for chapter, and the context section/section/title defines the numbering for section/section. So in the case of title contexts, the "current element" referred to in the comments below, is actually the parent element.

The "format" attribute consists of:
- element names, or the string "(PREVIOUS)", or "(CURRENT)" alternating with
- strings of punctuation characters delimited by single quotes.
A space separates each token.

Any non-alphanumberic character can be used as punctuation. The string ' is allowed as punctuation and means the straight single quote.

An element name signifies the (possibly compound) number defined for that element. Currently only 1 element name can appear in the format. The only elements valid in the format are divisions.

"(CURRENT)" signifies the current element. "(PREVIOUS)" is only allowed with division titles - it means the (possibly compound) number from the next higher level division title.

Examples: section '(' (CURRENT) ')'
(PREVIOUS) '.' (CURRENT)
(CURRENT) '.'
body-part '-' (CURRENT)
chapter ''' (CURRENT)

The "numberStyle" attribute determines the numbering style used:
1 means 1, 2, 3, 4 ...
A means A, B, ... Z, AA, AB, ...
a means a, b, ... z, aa, ab, ...
I means I, II, III, IV, ...
i means i, ii, iii, iv, ...

The "numberLabel" attribute is for the user-interface only. The value that actually gets used is propagated by styler to the BeforeText attribute.

The initialNumber attribute is the initial number value;

The "numberScope" attribute controls where the current element is counted from. Most commonly it is omitted, which means to count previous like-named siblings, i.e. the parent element is the scope. If present, it must specify a context selector of an ancestor element of the current element. The special value "(none)" means all occurrences of the element in the document are counted.

When an object is numbered, the first line indent and left indent controls on the Indent tab are read-only because the numbering UI controls the indenting. The values of those controls are determined by what's entered in the numbering UI. The left indent is set from the "Indent following lines:" field, adding the @ prefix. The first line indent is *-(leftind).

E.g., "Indent following lines" is 1in, then for Indent, we'd have:
<Indent leftIndent="@1in" firstLineIndent="*-1in">

numberAlignment says whether the number is to be right aligned at the alignAt point or left aligned there;

aligned at .....|
|
rightlleft
1.|1.
10.|10.

If followWith is "space", there is just a single space after the number, and the tabtoAfter is ignored. If followWith is "emspace", there is a 1em space after the number, and tabtoAfter is ignored. If followWith is "nothing", there is no space that follows the number and tabtoAfter is ignored. If followWith is "tab", then the content that follows the number starts at the position indicated by tabtoAfter.

The "Indent following lines" value in the UI corresponds with Indent.leftIndent,
so there is no attribute for it here.
-->

```
<xsd:element name="Numbering">
 <xsd:complexType>
  <xsd:attribute name="format"/>
  <xsd:attribute name="numberStyle" type="numberStyles"/>
  <xsd:attribute name="numberLabel"/>
  <xsd:attribute name="initialNumber" default="1" type="integer"/>
  <xsd:attribute name="numberScope"/>
  <xsd:attribute name="numberAlignment">
   <xsd:simpleType>
    <xsd:restriction base="xsd:token">
     <xsd:enumeration value="right"/>
     <xsd:enumeration value="left"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="alignAt"/>
  <xsd:attribute name="followWith" type="followWithValues"/>
  <xsd:attribute name="tabtoAfter"/>
 </xsd:complexType>
</xsd:element>
<!--
 The Bulleting properties are valid only for elements
 that are mapped to listItem style.

 Its properties control positioning of bullets. The bullet
 character itself, and its font data are in the contents
 of the element's AddBefore gentext.

 Bulleting follows the numbering model wrt indents and such, but
 the bullet is always left aligned since its size never varies
 between different list items.

-->
<xsd:element name="Bulleting">
 <xsd:complexType>
  <xsd:attribute name="alignAt"/>
  <xsd:attribute name="followWith" type="followWithValues"/>
  <xsd:attribute name="tabtoAfter"/>
 </xsd:complexType>
</xsd:element>
<!--
 ** schema.propertysets: PropertySets

 A "PropertySet" is a set of properties that can be referenced from
 multiple places within the stylesheet (anywhere there is a
 propertySetReferences attribute). PropertySets allow stylesheets
```

```
    to be modularized.  PropertySets can reference other PropertySets.
   -->
   <xsd:element name="PropertySets">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element minOccurs="0" maxOccurs="unbounded"
ref="PropertySet"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="PropertySet">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element ref="BaseProperties"/>
     </xsd:sequence>
     <xsd:attribute name="propertySetId" use="required" type="xsd:ID"/>
    </xsd:complexType>
   </xsd:element>
   <!--
   ============================================================
   ============ -->
   <!--
    ** schema.formattingelements: StylerFormattingElements and
UserFormattingElements

    StylerFormattingElements and UserFormattingElements are described
    in the .style file just like user DTD or schema elements.
   -->
   <xsd:element name="StylerFormattingElements">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element minOccurs="0" maxOccurs="unbounded" ref="Element"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="UserFormattingElements">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element minOccurs="0" maxOccurs="unbounded" ref="Element"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <!--
   ============================================================
   ============ -->
   <!--
```

```
** schema.elements: Elements

The "Elements" section is the heart of the stylesheet.

Elements are mapped to styles. The element's
style determines:

* what contexts are created for the element; this interacts
  with other elements' style mappings, e.g., for
  each division element, Styler automatically creates contexts
  for each title element inside the division.

* the initial formatting properties

* whether the element is listed in certain places in the UI

* a few styles imply properties that do not have
  corresponding markup in the Properties element

For some elements, it is desirable to use different properties
depending on the context the element occurs in, so Styler
associates formatting properties with element "contexts", rather
than directly with elements. "Context" is the term used in
Styler to specify the combination of an element selector and a
set of formatting properties to use for elements that match
that selector. The "selector" is specified using an XSL
pattern. (Patterns are certain types of XPath expressions.)
Only one context for a given instance of an element in a
document, the one whose selector best matches the element's
context in the document, is processed. If selector is omitted,
it implies an XSL pattern that is just the element name.

Finally, users may edit the stylesheet source code for one or
more outputs. We store their code as PCDATA, escaping < and &.

-->
<xsd:element name="Elements">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element minOccurs="0" maxOccurs="unbounded" ref="Element"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Element">
 <xsd:complexType>
  <xsd:sequence>
```

```
        <xsd:choice minOccurs="0">
          <xsd:element ref="GraphicDetails"/>
          <xsd:element ref="LinkDetails"/>
          <xsd:element ref="LinkTargetDetails"/>
          <xsd:element ref="IndexTermElementModelDetails"/>
          <xsd:element ref="IndexTermAttributeModelDetails"/>
          <xsd:element ref="DefinitionListDetails"/>
        </xsd:choice>
        <xsd:element maxOccurs="unbounded" ref="Context"/>
        <xsd:element minOccurs="0" maxOccurs="unbounded"
    ref="EditedSource"/>
      </xsd:sequence>
      <xsd:attribute name="name" use="required"/>
      <xsd:attribute name="style" use="required" type="styleNames"/>
      <xsd:attribute name="divisionLevel" type="integer"/>
      <xsd:attribute name="divisionNesting" type="integer"/>
      <xsd:attribute name="description"/>
    </xsd:complexType>
  </xsd:element>
  <!--
    Division Details

    The divisionLevel and divisionNesting attributes above are set only
    for elements mapped to the "division" style. They contain data
    entered in the Division Details dialog:

    divisionLevel - the division's level in the hierarchy (1-n); This
    attribute controls initial properties of title contexts created for
    titles of this division.

    divisionNesting - the total number of levels at which the division can
    occur. If the division element does not include itself in its content
    model, then nesting is 1 and cannot be changed

  -->
  <!--
    Graphic Details

    The GraphicDetails element appears only for elements mapped to the
    "graphic" style. It contains data entered in the Graphic Details
    dialog. Its attributes are explained below.

    primary - Is this the primary graphic element?

    The other attributes are used to specify which of the graphic element's
    attributes are to be used for the specific graphic traits. E.g., the
```

```
value for the filename attribute below should be the name of the
attribute for this element that contains the graphic's filename.
See the online help for a description of each of the graphic traits.
-->
<xsd:element name="GraphicDetails">
 <xsd:complexType>
  <xsd:attribute name="primary" default="no" type="yesorno"/>
  <xsd:attribute name="cropHeight" type="xsd:NMTOKEN"/>
  <xsd:attribute name="cropLowerLeftX" type="xsd:NMTOKEN"/>
  <xsd:attribute name="cropLowerLeftY" type="xsd:NMTOKEN"/>
  <xsd:attribute name="cropWidth" type="xsd:NMTOKEN"/>
  <xsd:attribute name="entity" type="xsd:NMTOKEN"/>
  <xsd:attribute name="filename" type="xsd:NMTOKEN"/>
  <xsd:attribute name="horizOffsetAmount" type="xsd:NMTOKEN"/>
  <xsd:attribute name="horizOffsetPercent" type="xsd:NMTOKEN"/>
  <xsd:attribute name="horizScale" type="xsd:NMTOKEN"/>
  <xsd:attribute name="notation" type="xsd:NMTOKEN"/>
  <xsd:attribute name="processor" type="xsd:NMTOKEN"/>
  <xsd:attribute name="reproDepth" type="xsd:NMTOKEN"/>
  <xsd:attribute name="reproWidth" type="xsd:NMTOKEN"/>
  <xsd:attribute name="resolution" type="xsd:NMTOKEN"/>
  <xsd:attribute name="scaleToFit" type="xsd:NMTOKEN"/>
  <xsd:attribute name="vertOffsetAmount" type="xsd:NMTOKEN"/>
  <xsd:attribute name="vertOffsetPercent" type="xsd:NMTOKEN"/>
  <xsd:attribute name="vertScale" type="xsd:NMTOKEN"/>
 </xsd:complexType>
</xsd:element>
<!--
 Link Details

 The LinkDetails element appears only for elements mapped to the
 "link" style. It contains data entered in the Link Details
 dialog. Its attributes are explained below.

 primary - Is this the primary link element?
 uriAttrname - Name of the element's attribute which contains Web-style links.
 idrefAttrname - Name of the element's attribute which contains document
links.
 -->
<xsd:element name="LinkDetails">
 <xsd:complexType>
  <xsd:attribute name="primary" default="no" type="yesorno"/>
  <xsd:attribute name="uriAttrname" type="xsd:NMTOKEN"/>
  <xsd:attribute name="idrefAttrname" type="xsd:NMTOKEN"/>
 </xsd:complexType>
</xsd:element>
```

```
<!--
 Link Target Details

 The LinkTargetDetails element appears only for elements mapped to the
 "linkTarget" style.  It contains data entered in the Link Target Details
 dialog.  Its attributes are explained below.

 primary - Is this the primary link target element?
 id - Name of the element's attribute which contains the target id.
-->
<xsd:element name="LinkTargetDetails">
 <xsd:complexType>
  <xsd:attribute name="primary" default="no" type="yesorno"/>
  <xsd:attribute name="id" type="xsd:NMTOKEN"/>
 </xsd:complexType>
</xsd:element>
<!--
 Index Term Details

 In the first release, Styler supports two models for index term
 markup:

 - the docbook model, or "element model", where a wrapper element has
 children for each of the 4 levels, each of which may optionally be
 followed by 1 "see" element, or multiple "see also" elements; it
 allows for the user of a "sort as" attribute

 - the cals model, or "attribute" model, which is also used by our
 Article and Book doctypes.  It has a single empty indexterm element,
 and that element has 4 attributes, 1 for each level; for styler, we've
 extended this model to allow a single See and See also attribute;
 these are assumed to apply to the lowest level whose term is specified
 in the same instance of the index term element

 The element names and attribute names can be different of course - users
 will tell us what role a given element or attribute plays in the
 appropriate details dialog, and the information is stored here as attributes
 of IndexTermElementModelDetails or IndexTermAttributeModelDetails


 The IndexTermElementModelDetails element appears only for elements
 mapped to the "indexTermElementModel" style.  The elementRole attribute
 says what index term role the element plays.  The sortAsAttribute gives
 the name of the attribute that contains the optional manual sort key.

 The attribute model case is parallel but without the sortas
```

attribute. Its attributes contain the names of the attributes that the user maps to index term roles in the Index Term (Attribute Model) Details dialog.

```
-->
<xsd:element name="IndexTermElementModelDetails">
 <xsd:complexType>
  <xsd:attribute name="elementRole">
   <xsd:simpleType>
    <xsd:restriction base="xsd:token">
     <xsd:enumeration value="wrapper"/>
     <xsd:enumeration value="1"/>
     <xsd:enumeration value="2"/>
     <xsd:enumeration value="3"/>
     <xsd:enumeration value="4"/>
     <xsd:enumeration value="see"/>
     <xsd:enumeration value="seealso"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="sortAsAttribute" type="xsd:NMTOKEN"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="IndexTermAttributeModelDetails">
 <xsd:complexType>
  <xsd:attribute name="level1Attribute" type="xsd:NMTOKEN"/>
  <xsd:attribute name="level2Attribute" type="xsd:NMTOKEN"/>
  <xsd:attribute name="level3Attribute" type="xsd:NMTOKEN"/>
  <xsd:attribute name="level4Attribute" type="xsd:NMTOKEN"/>
  <xsd:attribute name="seeAttribute" type="xsd:NMTOKEN"/>
  <xsd:attribute name="seeAlsoAttribute" type="xsd:NMTOKEN"/>
 </xsd:complexType>
</xsd:element>
<!--
 Definition List Details

-->
<xsd:element name="DefinitionListDetails">
 <xsd:complexType>
  <xsd:attribute name="definitionListItemElement" type="xsd:NMTOKEN"/>
  <xsd:attribute name="definitionListTermElement" use="required"
type="xsd:NMTOKEN"/>
  <xsd:attribute name="definitionListDefinitionElement" use="required"
type="xsd:NMTOKEN"/>
  <xsd:attribute name="termWidth" default="9pc" type="size"/>
```

```
      <xsd:attribute name="gutterWidth" default="1pc" type="size"/>
    </xsd:complexType>
  </xsd:element>
  <!--
    user edited stylesheet source code; each instance of this
    applies to one output
  -->
  <xsd:element name="EditedSource">
    <xsd:complexType mixed="true">
      <xsd:attribute name="sourceType" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:token">
            <xsd:enumeration value="xslHtml"/>
            <xsd:enumeration value="xslFo"/>
            <xsd:enumeration value="xslHtmlHelp"/>
            <xsd:enumeration value="xslWeb"/>
            <xsd:enumeration value="xslContributor"/>
            <xsd:enumeration value="fosi"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
  <!--
    BaseProperties are always present in .style files, but are
    optional here to so that the stylesheet will be valid when we strip it
    down to export a .wstyle file for use by the Word plug in
  -->
  <xsd:element name="Context">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0" ref="BaseProperties"/>
        <xsd:element minOccurs="0" maxOccurs="unbounded"
ref="OutputProperties"/>
        <xsd:element minOccurs="0" ref="Conditions"/>
      </xsd:sequence>
      <xsd:attribute name="selector" use="required"/>
      <xsd:attribute name="priority"/>
      <xsd:attributeGroup ref="added-attrs-for-exporting"/>
    </xsd:complexType>
  </xsd:element>
  <!--
  ============================================================
  ============ -->
  <!--
    ** schema.conditions: Conditions
```

```
When it is desired to change one or more formatting properties
of a context based on a conditional test, a "condition" is
used. Conditions consist of one or more boolean tests and a
set of formatting properties. If the condition is true, the
specified formatting properties are overlaid on the context's
formatting properties.

A condition is true if:

* "logic" = "and" and all its tests are true

* "logic" = "or" and any one of its tests is true

Contexts can have any number of conditions. When a context
is used, every condition is tested, and the formatting properties
are overlaid for every condition that is true.
-->
<xsd:element name="Conditions">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element minOccurs="0" maxOccurs="unbounded" ref="Condition"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<!--
 BaseProperties are always present in .style files, but are
 optional here to so that the stylesheet will be valid when we strip it
 down to export a .wstyle file for use by the Word plug in
-->
<xsd:element name="Condition">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="Tests"/>
   <xsd:element minOccurs="0" ref="BaseProperties"/>
   <xsd:element minOccurs="0" maxOccurs="unbounded"
ref="OutputProperties"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="added-attrs-for-exporting"/>
 </xsd:complexType>
</xsd:element>
<!--
 XPath tests are not in the 5.0 release, but are projected to be
 in a future release
-->
<xsd:element name="Tests">
```

```
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded"
    ref="AttributeTest"/>
        <xsd:element minOccurs="0" ref="XPathTest"/>
       </xsd:sequence>
       <xsd:attribute name="logic" default="and">
        <xsd:simpleType>
         <xsd:restriction base="xsd:token">
          <xsd:enumeration value="and"/>
          <xsd:enumeration value="or"/>
         </xsd:restriction>
        </xsd:simpleType>
       </xsd:attribute>
      </xsd:complexType>
     </xsd:element>
     <!--

      An attribute test can test an attribute of the current element
      or of any ancestor - the elementName attribute specifies the
      element whose attribute to test. If omitted, it's the current
      element whose attribute is tested. If present, it's the
      current element OR an ancestor element. There is no way to
      specify an ancestor element with the same element name.

      testType meanings:
      "any" - the attribute has any value at all (even an empty string)
      "none" - the attribute does not have any value at all
      "includes" - the attribute includes a space delimited token that
                   matches testValue
      "equals" - the attribute's value matches testValue exactly
      and so on

      testValue is ignored when testType is "any" or "none"
     -->
     <xsd:element name="AttributeTest">
      <xsd:complexType>
       <xsd:attribute name="elementName"/>
       <xsd:attribute name="attributeName" use="required"/>
       <xsd:attribute name="testType" default="any">
        <xsd:simpleType>
         <xsd:restriction base="xsd:token">
          <xsd:enumeration value="any"/>
          <xsd:enumeration value="none"/>
          <xsd:enumeration value="includes"/>
          <xsd:enumeration value="equals"/>
```

```
            <xsd:enumeration value="lessThan"/>
            <xsd:enumeration value="lessThanOrEqual"/>
            <xsd:enumeration value="greaterThan"/>
            <xsd:enumeration value="greaterThanOrEqual"/>
            <xsd:enumeration value="notEqual"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="testValue"/>
    </xsd:complexType>
  </xsd:element>
  <!-- Missing elementName implies the current element (self) -->
  <!-- A present elementName implies a self or ancestor element -->
  <!--
    XPath tests are not in the 5.0 release, but are projected to be
    in a future release
  -->
  <xsd:element name="XPathTest">
    <xsd:complexType>
      <xsd:attribute name="expression" use="required"/>
    </xsd:complexType>
  </xsd:element>
  <!--
  ============================================================
  ============ -->
  <!--
    ** schema.styles: Styles

    This section controls Styler's behavior when an element is mapped
    to a style.  There must be a Style element for each style the user
    can map an element to.  The Contexts in this section do not use
    actual element names in selectors, but use either:
      "$elemname" - should be replaced with current element name
      "*<stylename>" - where <stylename> is the name of any style
    defined in this section. Contexts with this construct are copied
    for every element of the named style, substituting those elements'
    names for "*<stylename"; See $aptpath/styler/new/new.style for examples.
  -->
  <xsd:element name="Styles">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" ref="Style"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Style">
```

```
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element maxOccurs="unbounded" ref="Context"/>
       </xsd:sequence>
       <xsd:attribute name="styleName" use="required"/>
       <xsd:attribute name="baseStyleName"/>
      </xsd:complexType>
     </xsd:element>
     <!--
================================================================
============ -->
     <!--
      ** schema.bullets: Bullets

      This section stores data on the bullet characters most recently
      used in the Styler UI.
     -->
     <xsd:element name="Bullets">
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element maxOccurs="unbounded" ref="Bullet"/>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>
     <xsd:element name="Bullet" type="any"/>
     <!--
================================================================
============ -->
     <!--
      ** schema.printpagesets: PrintPageSets

     -->
     <xsd:element name="PrintPageSets">
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded"
ref="PrintPageSet"/>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>
     <xsd:element name="PrintPageSet">
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element ref="Margins"/>
        <xsd:element ref="Headers"/>
        <xsd:element ref="Footers"/>
```

```
        <xsd:element ref="Columns"/>
        <xsd:element ref="PageNumbers"/>
        <xsd:element minOccurs="0" maxOccurs="unbounded"
    ref="EditedSource"/>
      </xsd:sequence>
      <xsd:attribute name="pageSetName" use="required"
    type="xsd:NMTOKEN"/>
      <xsd:attribute name="doubleSided" default="no" type="yesorno"/>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="Margins">
     <xsd:complexType>
      <xsd:attribute name="paperSize" default="Letter"/>
      <xsd:attribute name="paperHeight" default="11in" type="size"/>
      <xsd:attribute name="paperWidth" default="8.5in" type="size"/>
      <xsd:attribute name="orientation" default="portrait">
       <xsd:simpleType>
        <xsd:restriction base="xsd:token">
         <xsd:enumeration value="portrait"/>
         <xsd:enumeration value="landscape"/>
         <xsd:enumeration value="portland"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="top" default="1.5in" type="size"/>
      <xsd:attribute name="bottom" default="1.5in" type="size"/>
      <xsd:attribute name="evenLeft" default="1in" type="size"/>
      <xsd:attribute name="evenRight" default="1in" type="size"/>
      <xsd:attribute name="oddLeft" default="1in" type="size"/>
      <xsd:attribute name="oddRight" default="1in" type="size"/>
     </xsd:complexType>
    </xsd:element>
    <!--

     For all headers and footers, there's an "empty" attribute
     that is used to signify "no header".

     If single-sided, the OddHeaders determines the headers for
     both odd and even pages, and there are no blank pages.
     If OddHeaders is missing, there are no headers in the document.

     If double-sided and OddHeaders is missing, there are no headers on
     odd pages. If OddHeaders is present, Header represents the normal
     header to use. If FirstPageHeader is missing, the first page, if
     odd, uses the regular Header. If FirstPageHeader is present, it
     can be empty (its "empty" attribute is "yes"), or else its content
```

```
specifies the first page header.

If double-sided and EvenHeaders is missing, then even page headers
are the same as odd. If present, even page headers follow the
same model as odd.

If double-sided and BlankpageHeader is missing, then odd blank
pages use the normal odd header and even blank pages use the
normal even header. If BlankPageHeader is present, it can be
empty (its "empty" attribute is "yes"), or else its content
specifies the header to use on blank pages (whether even or odd).
-->
<xsd:element name="Headers">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element minOccurs="0" ref="OddHeaders"/>
   <xsd:element minOccurs="0" ref="EvenHeaders"/>
   <xsd:element minOccurs="0" ref="BlankPageHeader"/>
  </xsd:sequence>
  <xsd:attribute name="fromEdge" default="1in" type="size"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Footers">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element minOccurs="0" ref="OddFooters"/>
   <xsd:element minOccurs="0" ref="EvenFooters"/>
   <xsd:element minOccurs="0" ref="BlankPageFooter"/>
  </xsd:sequence>
  <xsd:attribute name="fromEdge" default="1in" type="size"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="OddHeaders">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element minOccurs="0" ref="FirstPageHeader"/>
   <xsd:element ref="Header"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name="OddFooters">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element minOccurs="0" ref="FirstPageFooter"/>
   <xsd:element ref="Footer"/>
  </xsd:sequence>
```

```
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="EvenHeaders">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element minOccurs="0" ref="FirstPageHeader"/>
    <xsd:element ref="Header"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="EvenFooters">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element minOccurs="0" ref="FirstPageFooter"/>
    <xsd:element ref="Footer"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="FirstPageHeader">
  <xsd:complexType>
   <xsd:complexContent>
    <xsd:extension base="any">
     <xsd:attribute name="empty" default="no" type="yesorno"/>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="Header">
  <xsd:complexType>
   <xsd:complexContent>
    <xsd:extension base="any">
     <xsd:attribute name="empty" default="no" type="yesorno"/>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="BlankPageHeader">
  <xsd:complexType>
   <xsd:complexContent>
    <xsd:extension base="any">
     <xsd:attribute name="empty" default="no" type="yesorno"/>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="FirstPageFooter">
```

```
<xsd:complexType>
 <xsd:complexContent>
  <xsd:extension base="any">
   <xsd:attribute name="empty" default="no" type="yesorno"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
</xsd:element>
<xsd:element name="Footer">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="any">
    <xsd:attribute name="empty" default="no" type="yesorno"/>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
<xsd:element name="BlankPageFooter">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="any">
    <xsd:attribute name="empty" default="no" type="yesorno"/>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Columns">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element minOccurs="0" maxOccurs="unbounded" ref="Column"/>
  </xsd:sequence>
  <xsd:attribute name="initialColumnCount" default="1"
type="numberOfColumns"/>
  <xsd:attribute name="balance" default="no" type="yesorno"/>
  <xsd:attribute name="raggedBottom" default="yes" type="yesorno"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Column">
 <xsd:complexType>
  <xsd:attribute name="columnCount" use="required"
type="numberOfColumns"/>
  <xsd:attribute name="columnWidth" type="size"/>
  <xsd:attribute name="gutter" default=".25in" type="size"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="PageNumbers">
```

```
    <xsd:complexType>
     <xsd:complexContent>
      <xsd:extension base="any">
       <xsd:attribute name="pageNumberStyle" default="1"
5   type="numberStyles"/>
      </xsd:extension>
     </xsd:complexContent>
    </xsd:complexType>
   </xsd:element>
10  <!--
    ========================================================================
    ============ -->
    <!--
     ** schema.Tables: Tables
15
     For each table model used by this doctype, there should be a
     TableModel element whose "model" attribute is set appropriately.
     Only one TableModel element should set "primary" to yes.

20   -->
    <xsd:element name="Tables">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element minOccurs="0" maxOccurs="unbounded"
25  ref="TableModel"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="TableModel">
30   <xsd:complexType>
      <xsd:attribute name="primary" default="no" type="yesorno"/>
      <xsd:attribute name="model">
       <xsd:simpleType>
        <xsd:restriction base="xsd:token">
35        <xsd:enumeration value="cals"/>
          <xsd:enumeration value="html"/>
          <xsd:enumeration value="arbortext"/>
        </xsd:restriction>
       </xsd:simpleType>
40    </xsd:attribute>
     </xsd:complexType>
    </xsd:element>
    <!--
    ========================================================================
45  ============ -->
    <!--
```

```
** schema.TablesOfContents: Tables of Contents

For each Table of Contents defined by the user, there is a
TableOfContents element. This element defines various
properties of the Table of Contents by its attributes.
Non-obvious ones are explained here:

- tocName acts like an ID.
- tocScope if empty, means the TOC is not scoped, i.e. it
  is for the entire document; otherwise, it must be an
  element name;
- tocDivLevels refers to the number of division levels to
  include in the TOC, but it does not actually determine
  that; it is used in the user interface, and if changed
  by the user via the UI, it affects what division title
  contexts are included in the table of contents, but it
  is the TocEntry elements inside the GeneratedText of
  title contexts that determines which titles are included
  the table of contents.
- textRightIndent controls how much space is reserved for
  page numbers; it keeps leader dots from abutting page
  numbers and it keeps wrapped titles from extruding over
  page numbers
- if the size available for leaderMinLength is not available,
  then the TOC entry text is wrapped to the next line;

Each TableOfContents element has five TocLevelIndent children,
one for each TOC level. Those elements control indenting and
number positioning of table of contents entries.

Whether and where a TableOfContents appears in the document
is determined by the presence of a _gte:TOC element in the
AddBefore or AddAfter child of GeneratedText of any element.
-->
<xsd:element name="TablesOfContents">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element minOccurs="0" maxOccurs="unbounded"
ref="TableOfContents"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name="TableOfContents">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element maxOccurs="unbounded" ref="TocLevelIndent"/>
```

```
      </xsd:sequence>
      <xsd:attribute name="tocName" use="required" type="xsd:NMTOKEN"/>
      <xsd:attribute name="tocScope" type="xsd:NMTOKEN"/>
      <xsd:attribute name="tocDivLevels" default="3" type="integer"/>
      <xsd:attribute name="formalBlockElement"/>
      <xsd:attribute name="tocIndented" default="yes" type="yesorno"/>
      <xsd:attribute name="printPageNumbers" default="yes" type="yesorno"/>
      <xsd:attribute name="rightAlignPageNumbers" default="yes"
type="yesorno"/>
      <xsd:attribute name="textRightIndent" default="2pc" type="size"/>
      <xsd:attribute name="leaderDots" default="yes" type="yesorno"/>
      <xsd:attribute name="leaderSpacing" default="3pt" type="size"/>
      <xsd:attribute name="leaderMinLength" default="3pc" type="size"/>
    </xsd:complexType>
  </xsd:element>
  <!--
    There is one TocLevelIndent element for each of the 5 possible
    levels of a table of contents

    They also control whether the toc entry includes the title's
    label and number, number only, or neither.

    They control the indents and number and text positioning for
    table of contents entries at the level specified by the
    tocLevel attribute.

    The attributes correspond obviously with the group of controls
    under the Numbered radio button in the dialog.  If the user
    selects Not numbered, then we emit:
      numberAlignment=left
      alignAt=0em
      followWith=tab
      tabtoAfter={IndentFirstLine}
      indentFollowing={IndentFollowingLines}
    where {...} means the value entered in the control named ...

  -->
  <xsd:element name="TocLevelIndent">
    <xsd:complexType>
      <xsd:attribute name="tocLevel" type="integer"/>
      <xsd:attribute name="numbered" type="yesorno"/>
      <xsd:attribute name="includeLabel" type="yesorno"/>
      <xsd:attribute name="numberAlignment">
        <xsd:simpleType>
          <xsd:restriction base="xsd:token">
            <xsd:enumeration value="right"/>
```

```
            <xsd:enumeration value="left"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute name="alignAt"/>
         <xsd:attribute name="followWith" type="followWithValues"/>
         <xsd:attribute name="tabtoAfter"/>
         <xsd:attribute name="indentFollowing" type="size"/>
        </xsd:complexType>
       </xsd:element>
       <!--
        Indexing

        In the first release, only one index can be configured. Most of
        the attributes are related to the index format. The settings the
        user makes in the index format dialog, are stored here, but more
        importantly, they are propagated to the styler formatting
        elements that mark-up the final index. The user can directly
        style those elements to get finer control than index format
        dialog allows
       -->
       <xsd:element name="Indexes" type="Index"/>
       <xsd:complexType name="Index">
        <xsd:sequence>
         <xsd:element ref="Index"/>
        </xsd:sequence>
       </xsd:complexType>
       <xsd:element name="Index">
        <xsd:complexType>
         <xsd:attribute name="indexName" use="required" type="xsd:NMTOKEN"/>
         <xsd:attribute name="numberPosition" default="rightAligned">
          <xsd:simpleType>
           <xsd:restriction base="xsd:token">
            <xsd:enumeration value="rightAligned"/>
            <xsd:enumeration value="spaceSeparated"/>
            <xsd:enumeration value="commaSeparated"/>
            <xsd:enumeration value="bulletSeparated"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute name="leaderDots" default="no" type="yesorno"/>
         <xsd:attribute name="leaderSpacing" default="3pt" type="size"/>
         <xsd:attribute name="leaderMinLength" default="3em" type="size"/>
         <xsd:attribute name="showHeadings" default="yes" type="yesorno"/>
         <xsd:attribute name="headingsAlignment" default="left">
          <xsd:simpleType>
```

```
            <xsd:restriction base="xsd:token">
              <xsd:enumeration value="left"/>
              <xsd:enumeration value="center"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="headingsBold" default="yes" type="yesorno"/>
        <xsd:attribute name="headingsItalic" default="no" type="yesorno"/>
        <xsd:attribute name="headingsAllcaps" default="yes" type="yesorno"/>
      </xsd:complexType>
    </xsd:element>
    <!--
    ============================================================
    ============ -->
    <!--
      ** schema.FosiRsrcDescSource: FosiRsrcDescSource

      Contains source edits for the FOSI rsrcdesc. These are output when
      a FOSI is exported. Advanced users can override rsrcdesc items that
      Arbortext Styler emits.

    -->
    <xsd:element name="FosiRsrcDescSource">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element minOccurs="0" maxOccurs="unbounded"
ref="FosiRsrcDescItem"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <!--
    ============================================================
    ============ -->
    <!--
      ** schema.FosiRsrcDescItem: FosiRsrcDescItem

      Contains source for one possible child of <rsrcdesc>. The source is stored
      as escaped PCDATA. The attributes have the following meaning:

        tagName  - name of the <rsrcdesc> child element.
          keyValue - value of the ID attribute for this child element

      These attributes are used to simplify the FOSI export process in which this
      data must be merged along with similar data being exported from Styler.

    -->
```

```
<xsd:element name="FosiRsrcDescItem">
 <xsd:complexType mixed="true">
  <xsd:attribute name="tagName" use="required">
   <xsd:simpleType>
    <xsd:restriction base="xsd:token">
     <xsd:enumeration value="hyphrule"/>
     <xsd:enumeration value="charfill"/>
     <xsd:enumeration value="counter"/>
     <xsd:enumeration value="stringdecl"/>
     <xsd:enumeration value="floatloc"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="keyValue"/>
 </xsd:complexType>
</xsd:element>
<xsd:complexType name="any" mixed="true">
 <xsd:sequence>
  <xsd:any minOccurs="0" maxOccurs="unbounded"
processContents="strict"/>
 </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

CLAIMS

We claim:

1. A single stylesheet defining specifications for outputting content in multiple outputs, comprising:

instructions within the single stylesheet identifying information to be output; and

instructions within the single stylesheet defining at least one format for at least a portion of the information for each of the multiple outputs, whereby, when the information is output in a particular format, the defined format for the selected output will be applied.

2. The stylesheet of claim 1, wherein the instructions within the stylesheet identifying information to be output comprise instructions identifying elements of an input file to be output.

3. The stylesheet of claim 1, further comprising instructions to identify a location of the information identified to be output.

4. The stylesheet of claim 2, further comprising instructions to identify type of element.

5. The stylesheet of claim 4, wherein the instructions include tags to identify the type of element.

6. The stylesheet of claim 1, wherein the instructions that define a format for at least a portion of the identified information to be output include at least one style archetype.

7. The stylesheet of claim 1, wherein the instructions that define a format for at least a portion of the identified information to be output include defining a font

for the identified information for each of the multiple outputs in which the information may be output.

8. The stylesheet of claim 1, wherein the instructions that define a format for at least a portion of the identified information to be output include defining indent properties for the identified information for each of the multiple outputs in which the information may be output.

9. The stylesheet of claim 1, wherein the instructions that define a format for at least a portion of the identified information to be output include defining spacing properties for the identified information for each of the multiple outputs in which the information may be output.

10. The stylesheet of claim 1, wherein the instructions that define a format for at least a portion of the identified information to be output include defining break properties for the identified information for each of the multiple outputs in which the information may be output.

11. The stylesheet of claim 1, wherein the instructions that define a format for at least a portion of the identified information to be output include defining an addition of automatically generated text in conjunction with identified information for each of the multiple outputs in which the information may be output.

12. A computer program product comprising:

a computer useable medium and computer readable code embodied on said computer useable medium for creating one stylesheet for formatting information to be output in multiple formats , the computer readable code comprising:

computer readable program code devices configured to cause the computer to effect the receiving of the input file;

computer readable program code devices configured to cause the computer to determine at least one element within the input file;

computer readable program code devices configured to determine a formatting rule applicable to the at least one element;

computer readable program code devices configured to apply the formatting rule to the at least one element;

computer readable program code devices configured for storing the at least one element having the formatting rule applied thereto as the output file of the second file type.

13. A computer data signal embodied in a transmission medium, comprising:

a code segment including instructions for receiving an input file;

a code segment including instructions for determining at least one element within the input file;

a code segment including instructions for determining a formatting rule applicable to the at least one element, wherein a formatting rule is determined for each of at least two output formats;

a code segment including instructions for applying a appropriate formatting rule to the at least one element for outputting the at least two output formats;

a code segment including instructions for storing the at least one element having the formatting rule applied thereto as an output file of a second file type.

14. A computer data signal embodied in a transmission medium, comprising:

a code segment including instructions for receiving and reading an input file; and

a code segment including instructions for applying at least one formatting rule to at least one element within the input file for each of at least two output formats, resulting in a formatted element consistent

with the formatting rule when output in one of the at least two output formats.

15. A method for outputting information in one of several output formats using a single stylesheet, the method comprising:

providing for determining an element that is to be output;

providing for determining a formatting rule applicable to the element for each of several output formats; and

providing for applying the formatting rule to the element consistent with the output format.

16. A method for outputting information in multiple formats using a single stylesheet, the method comprising:

determining an element to be output; determining at least one formatting rule for each element based on the form of output;

applying the formatting rule consistent with the output format selected for the element.

17. The method of claim 16, wherein determining a formatting rule applicable to the element and applying the formatting rule to the element comprise determining a style archetype applicable to the element and applying the style archetype to the element.

18. A method for publishing information, comprising:

receiving information in an input file;

publishing the information in multi-channels using a single stylesheet;

wherein the stylesheet applies formatting rules to elements within the input file, the formatting rules being adapted for multi-channels.

19. A method for creating a stylesheet capable of outputting information in one of multiple output file types using a single stylesheet, the method comprising:

within a single stylesheet, identifying and tagging at least one element within the input file;

within a single stylesheet, associating a style with the at least one element; and

within a single stylesheet, selecting an output file type, wherein the style may vary according to the output file type selected.

20. The method of claim 19, wherein the act of associating a style with the at least one element is done based on the location of the element within the input style.

21. The method of claim 19, wherein the act of associating a style with the at least one element is done based on a default assignment policy.

22. The method of claim 19, wherein the act of associating a style with the at least one element is done based on a user-specified assignment policy.

23. The method of claim 19, wherein the style comprises at least one formatting rule.

24. The method of claim 19, wherein the at least one element appears in one or more contexts and wherein the act of associating a style with the at least one element comprises associating the style with one of the element contexts.

25. The method of claim 24, further comprising applying a condition and associated formatting rule with at least one of the element contexts.

26. The method of claim 25, wherein the condition comprises a set of attribute tests plus the associated formatting rule.

27 The method of claim 19, wherein the act of associating a style with the at least one element is done using a property set.

28. The method of claim 19, wherein the selected output file type is context-specific.

29. The method of claim 19, wherein the selected output file type is user-specific.

30. The method of claim 19, wherein the selected output file type is device-specific.

31. The method of claim 19, further including associating a baseline characteristic to at least one element prior to associating a style with the at least one element.

32. A system for outputting content to various output media formats, the system comprising:

means for identifying information to be output;

means for associating at least one formatting rule with the information to be output, wherein the means further comprises associating a formatting rule for the information to be output for outputting in multiple output forms,

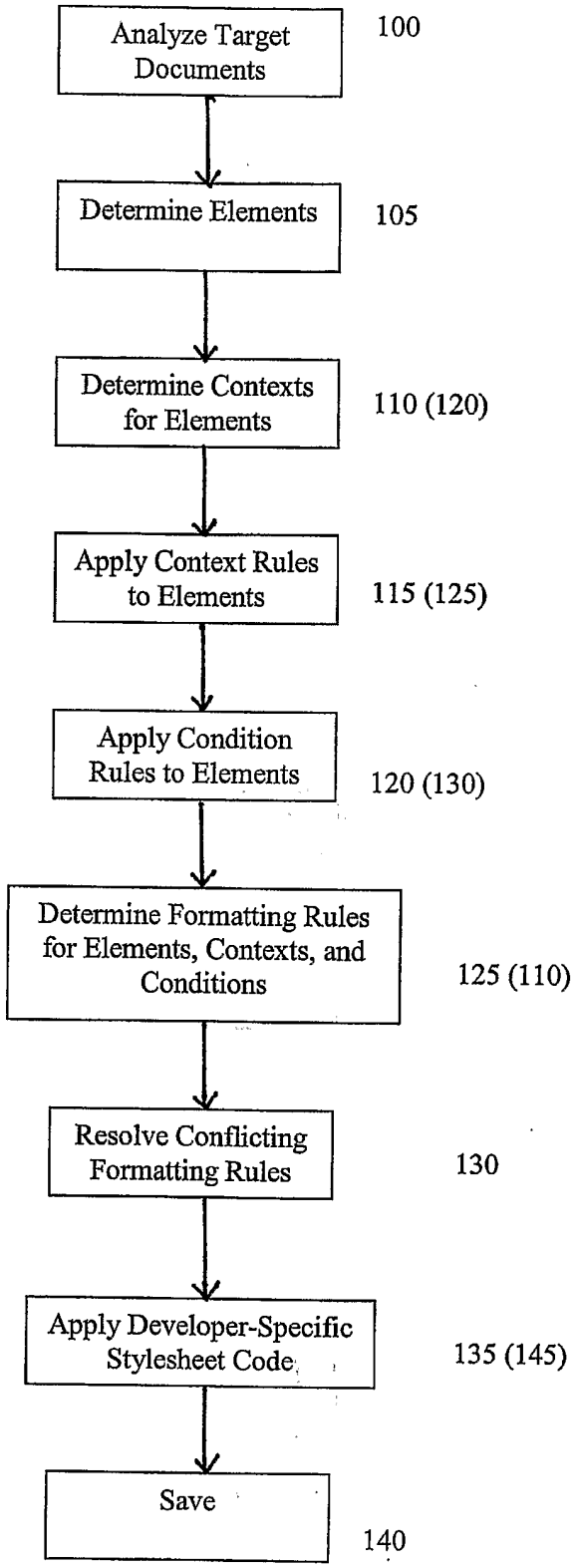
33. The system of claim 32, further comprising means for enabling a user to write code for an output format.

34. The system of claims 33, further comprising means for saving the code.

35. The system of claim 34, wherein the saved code may be managed and reused.

36. The system of claim 33, wherein the saved code is part of a library of code that may be managed and reused.

FIGURE 1


Analyze Target Documents
100
Determine Elements
105
Determine Contexts for Elements
110 (120)
Apply Context Rules to Elements
115 (125)
Apply Condition Rules to Elements
120 (130)
Determine Formatting Rules for Elements, Contexts, and Conditions
125 (110)
Resolve Conflicting Formatting Rules
130
Apply Developer-Specific Stylesheet Code
135 (145)
Save
140

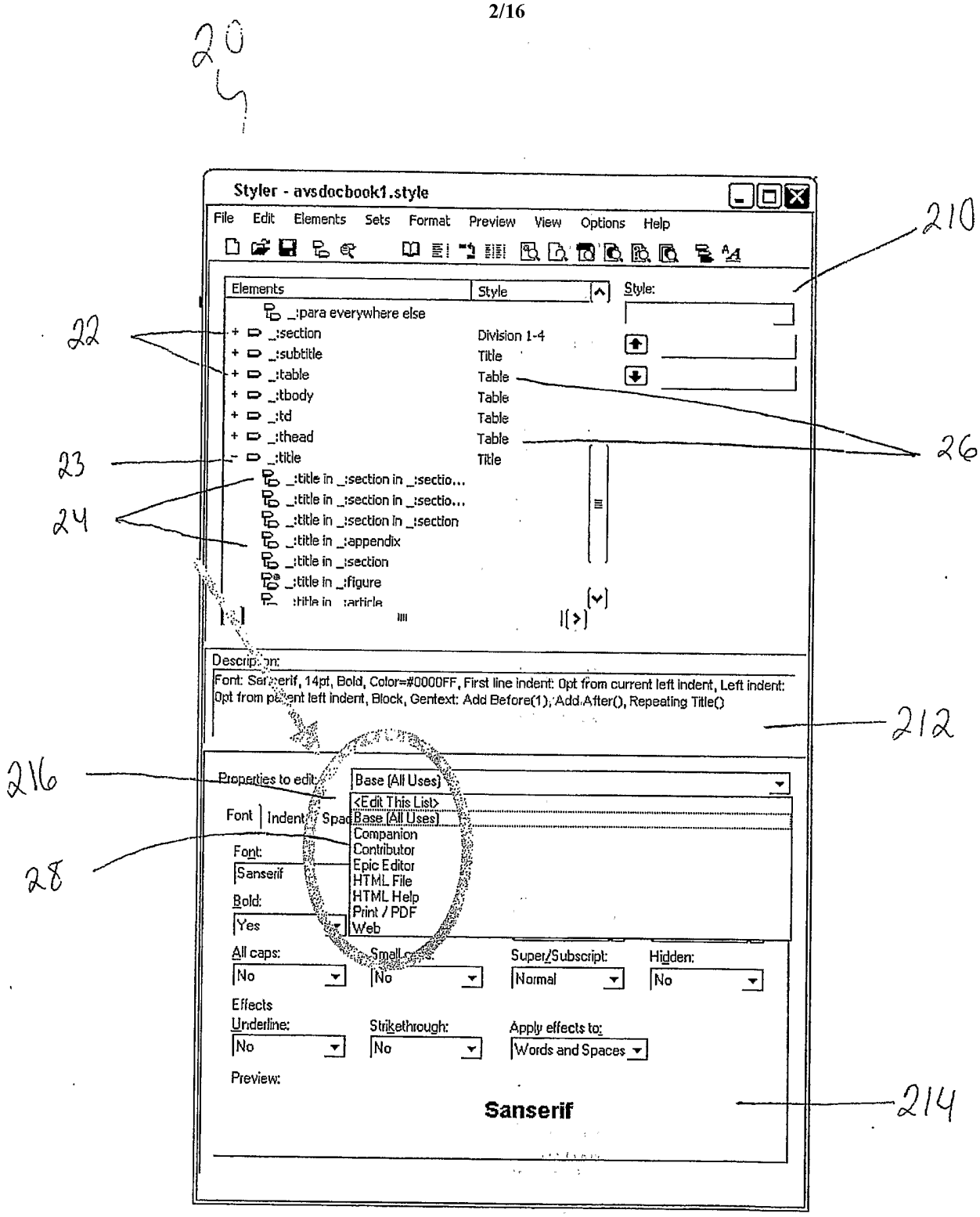

20
Styler - avsdocbook1.style
File Edit Elements Sets Format Preview View Options Help
210
Elements
Style
Style:
22
_:para everywhere else
_:section
Division 1-4
_:subtitle
Title
_:table
Table
_:tbody
Table
_:td
Table
_:thead
Table
23
_:title
Title
24
_:title in _:section in _:sectio...
_:title in _:section in _:sectio...
_:title in _:section in _:section
_:title in _:appendix
_:title in _:section
_:title in _:figure
26
Description:
Font: Sanserif, 14pt, Bold, Color=#0000FF, First line indent: 0pt from current left indent, Left indent: 0pt from parent left indent, Block, Gentext: Add Before(1), Add.After(), Repeating Title()
212
216
Properties to edit:
Base (All Uses)
<Edit This List>
Base (All Uses)
Companion
Contributor
Epic Editor
HTML File
HTML Help
Print / PDF
Web
Font
Indents
Spac
28
Font:
Sanserif
Bold:
Yes
All caps:
No
Small c...
No
Super/Subscript:
Normal
Hidden:
No
Effects
Underline:
No
Strikethrough:
No
Apply effects to:
Words and Spaces
Preview:
Sanserif
214

FIGURE 2

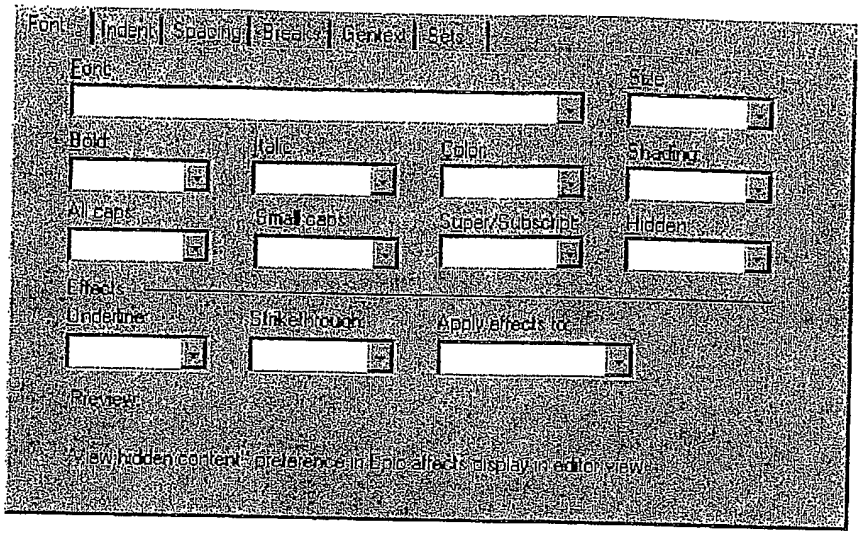


FIGURE 3

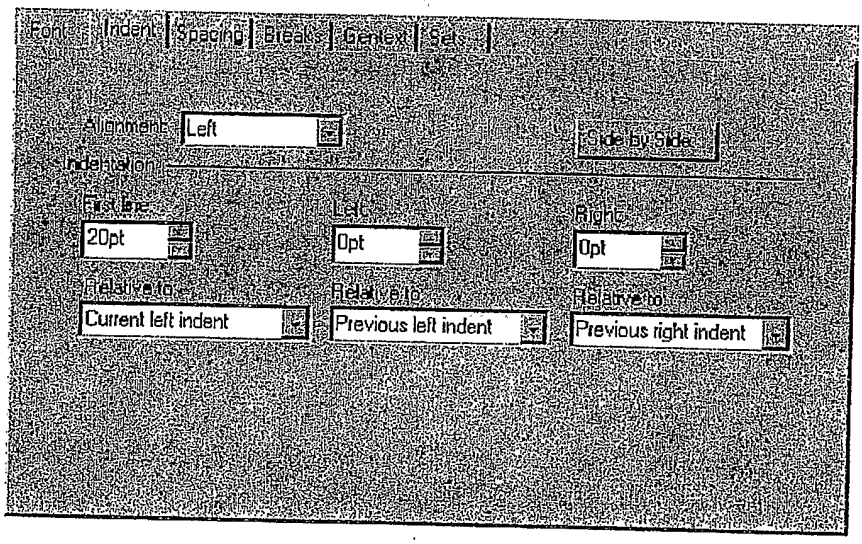
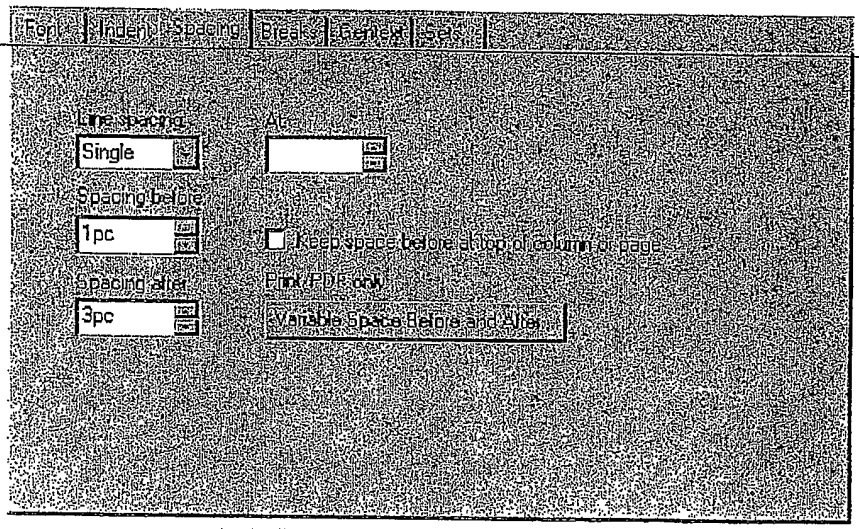


FIGURE 4

FIGURE 5



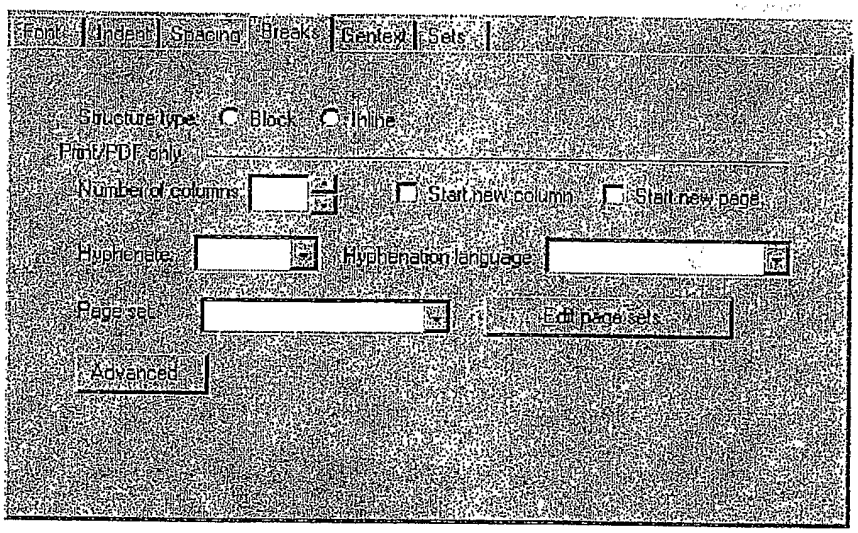


FIGURE 6

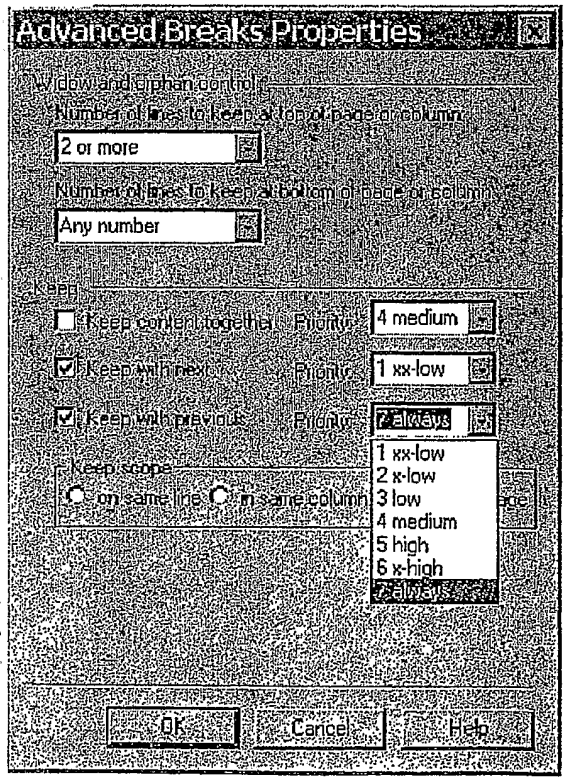


FIGURE 7

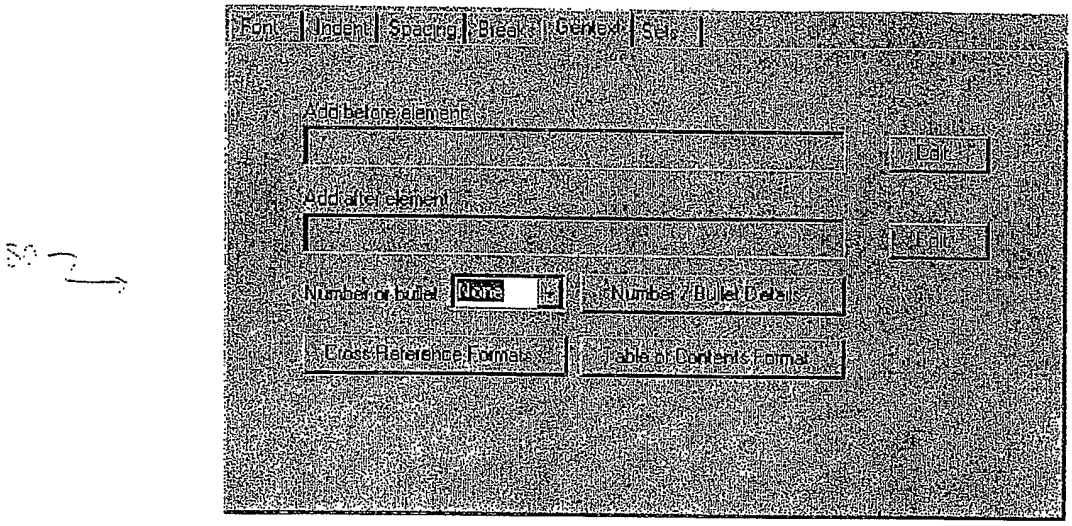


FIGURE 8

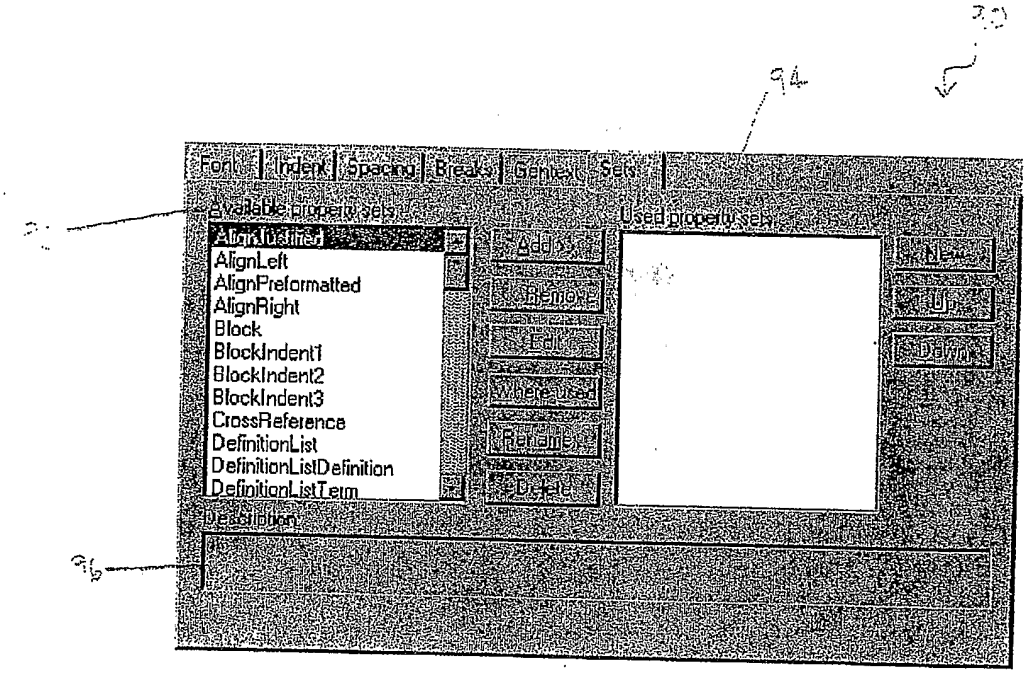
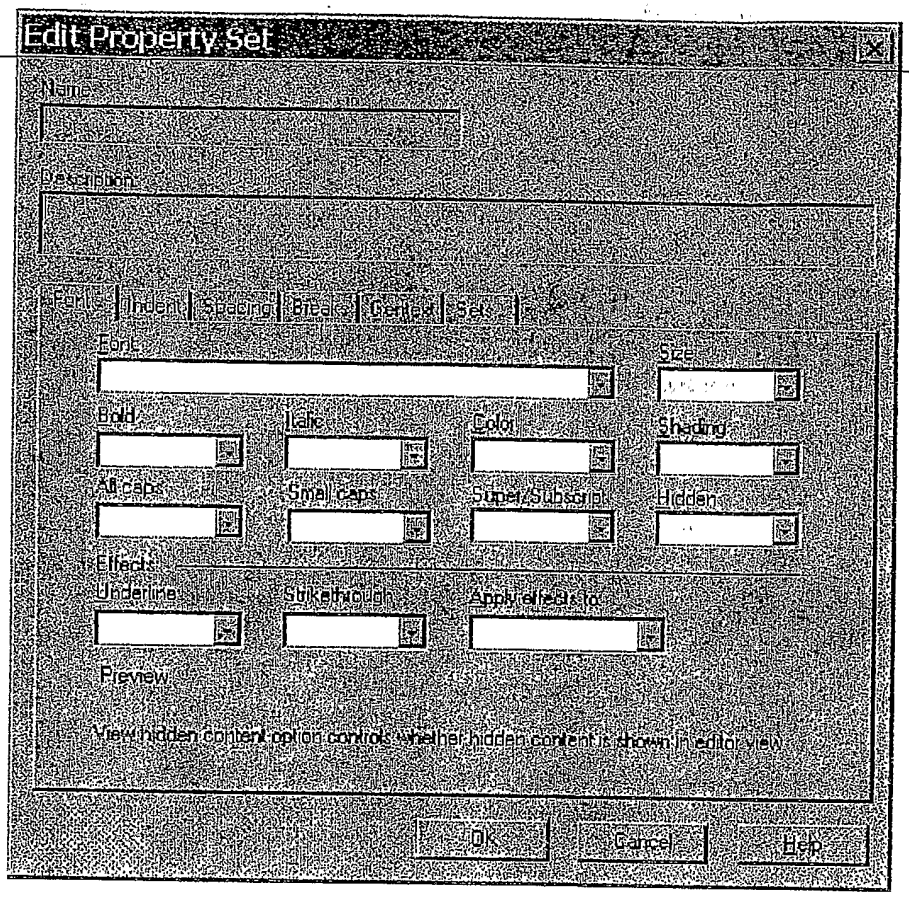


FIGURE 9



150

FIGURE 10

160

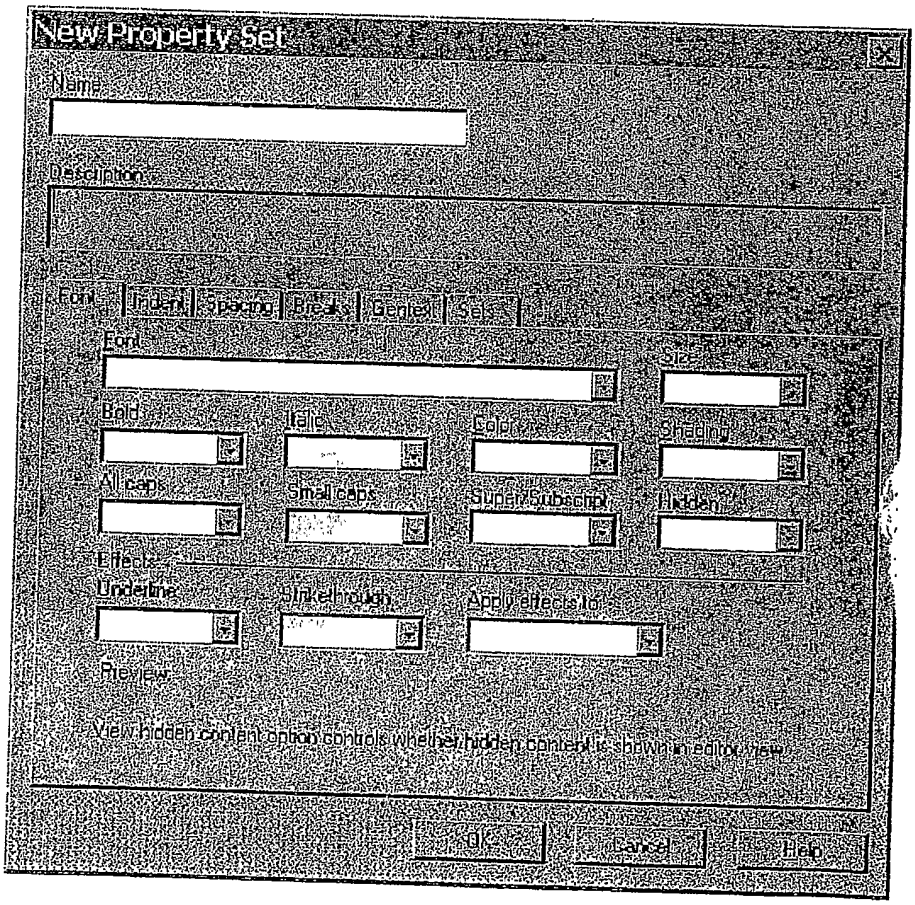


FIGURE 11

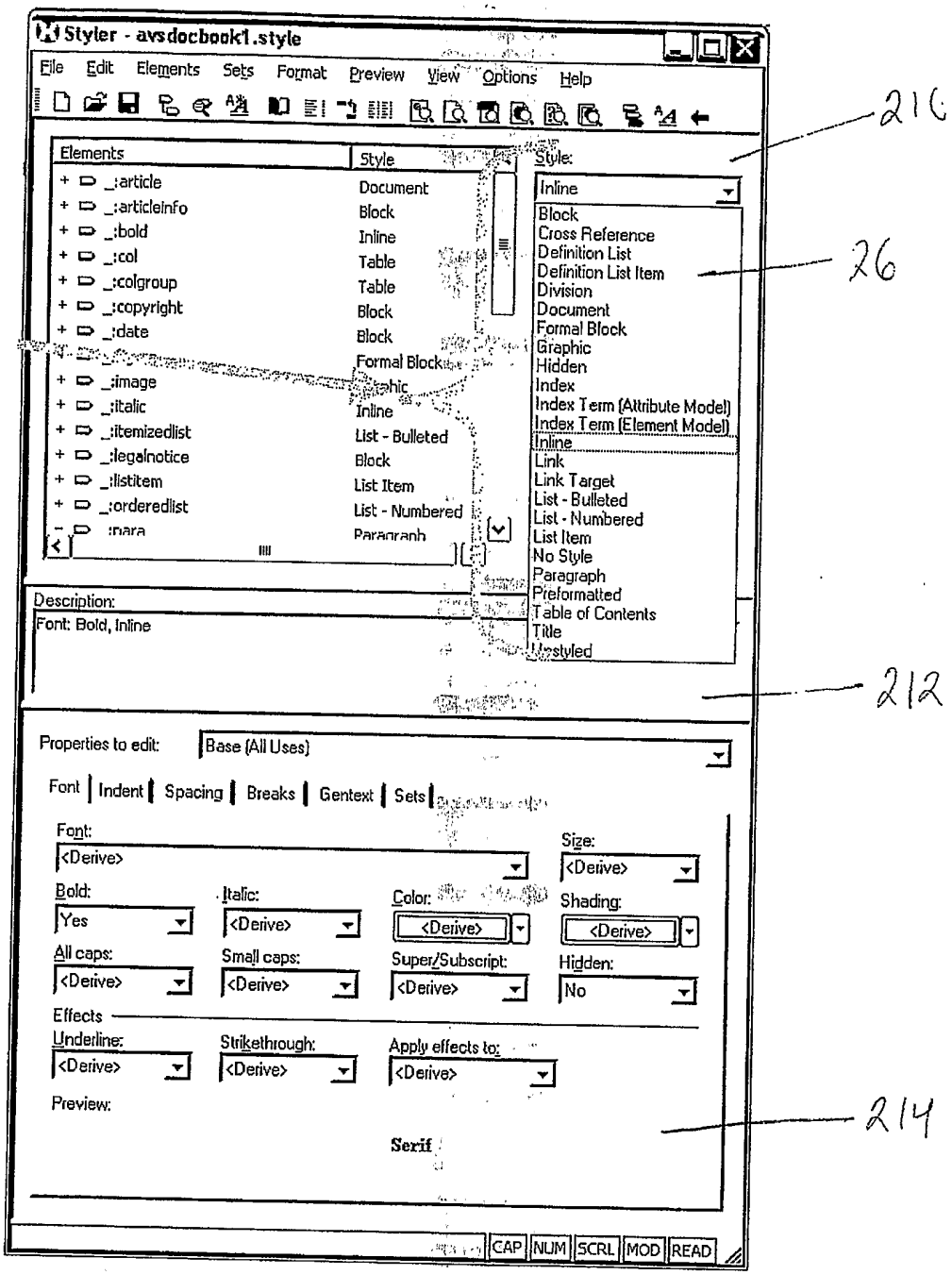

Styler - avsdocbook1.style
File Edit Elements Sets Format Preview View Options Help
Elements
Style
Style:
Inline
Block
Cross Reference
Definition List
Definition List Item
Division
Document
Formal Block
Graphic
Hidden
Index
Index Term (Attribute Model)
Index Term (Element Model)
Inline
Link
Link Target
List - Bulleted
List - Numbered
List Item
No Style
Paragraph
Preformatted
Table of Contents
Title
Unstyled
Description:
Font: Bold, Inline
Properties to edit:
Base (All Uses)
Font Indent Spacing Breaks Gentext Sets
Font:
Size:
Bold:
Italic:
Color:
Shading:
All caps:
Small caps:
Super/Subscript:
Hidden:
Effects
Underline:
Strikethrough:
Apply effects to:
Preview:
Serif
CAP NUM SCRL MOD READ
216
26
212
214

FIGURE 12

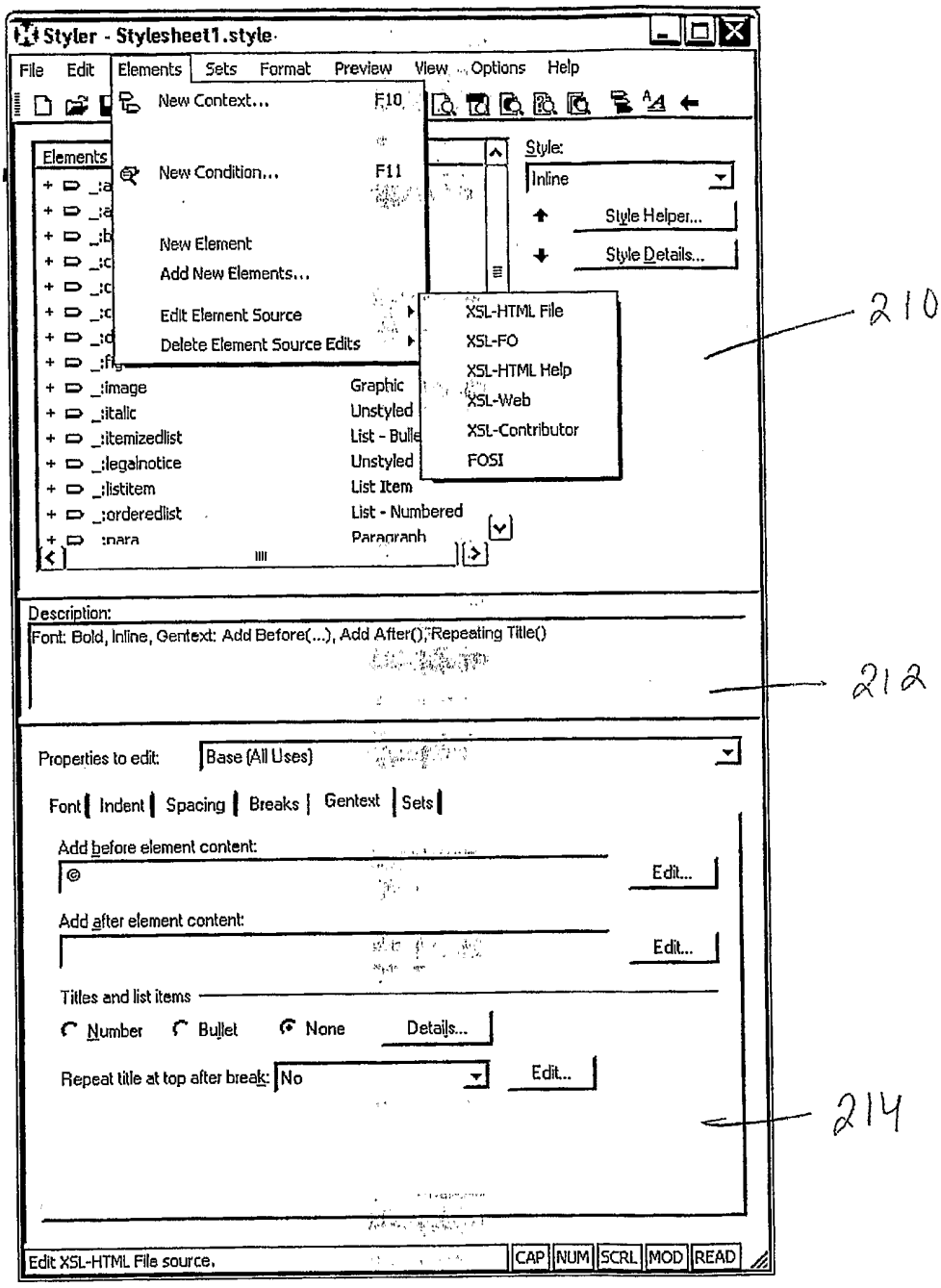

Styler - Stylesheet1.style
File Edit Elements Sets Format Preview View Options Help
New Context... F10
New Condition... F11
New Element
Add New Elements...
Edit Element Source
Delete Element Source Edits
XSL-HTML File
XSL-FO
XSL-HTML Help
XSL-Web
XSL-Contributor
FOSI
Style:
Inline
Style Helper...
Style Details...
Elements
_:image Graphic
_:italic Unstyled
_:itemizedlist List - Bulle
_:legalnotice Unstyled
_:listitem List Item
_:orderedlist List - Numbered
Description:
Font: Bold, Inline, Gentext: Add Before(...), Add After(), Repeating Title()
Properties to edit: Base (All Uses)
Font Indent Spacing Breaks Gentext Sets
Add before element content:
Edit...
Add after element content:
Edit...
Titles and list items
Number Bullet None Details...
Repeat title at top after break: No
Edit...
Edit XSL-HTML File source.
CAP NUM SCRL MOD READ
210
212
214

FIGURE 13

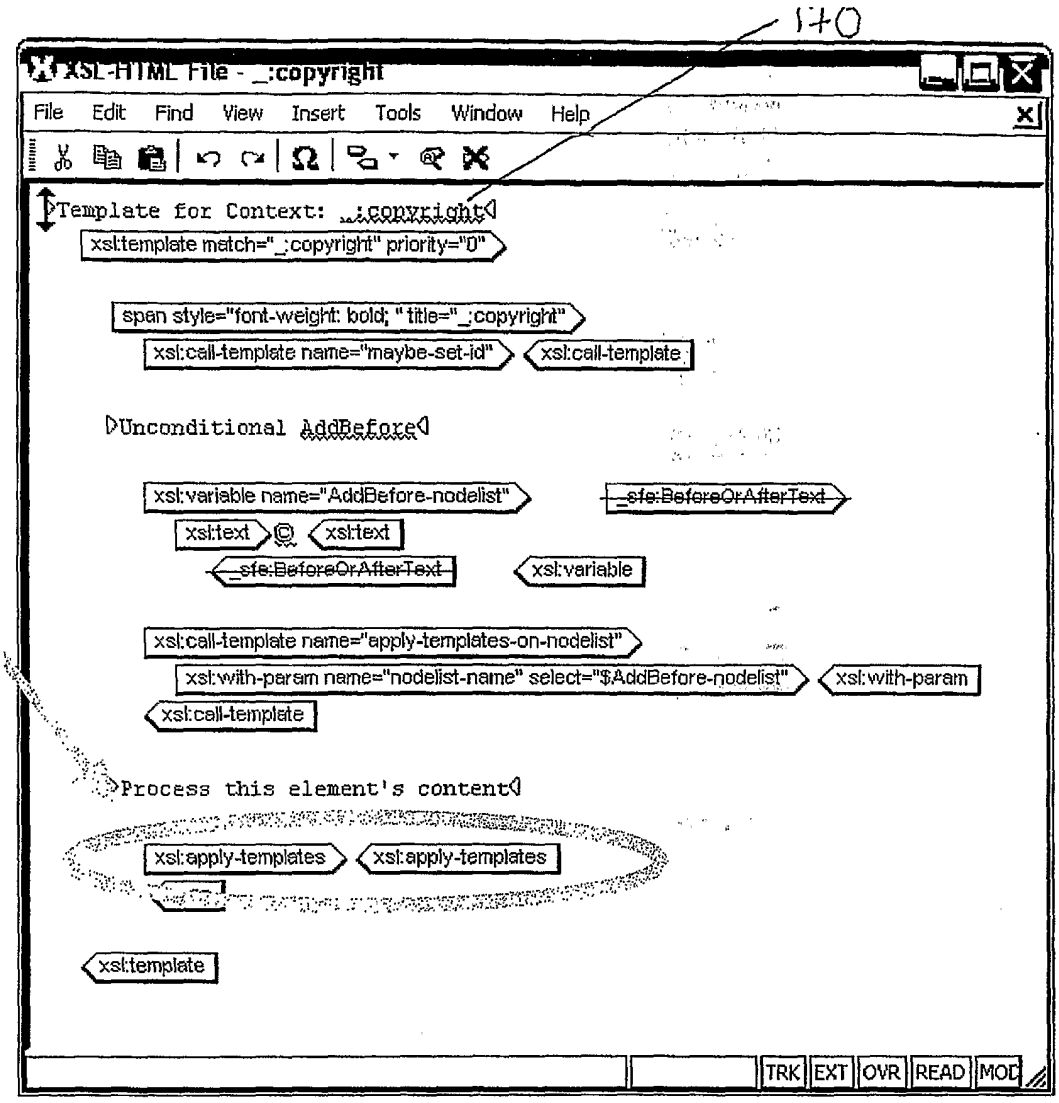


FIGURE 14

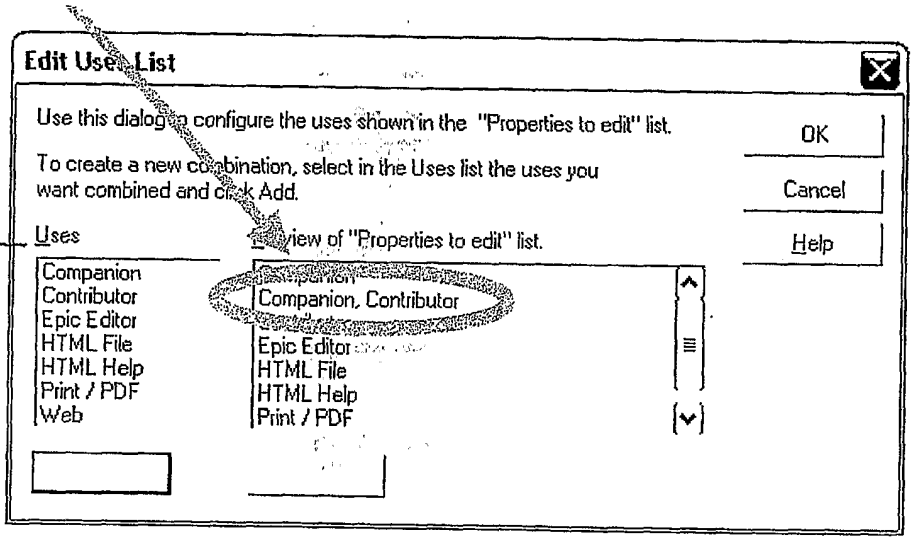


FIGURE 15

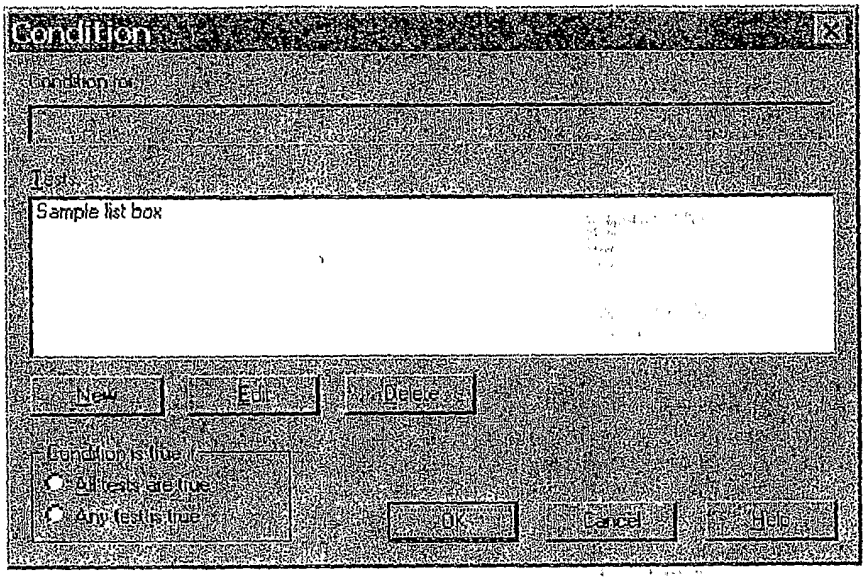


Figure 16

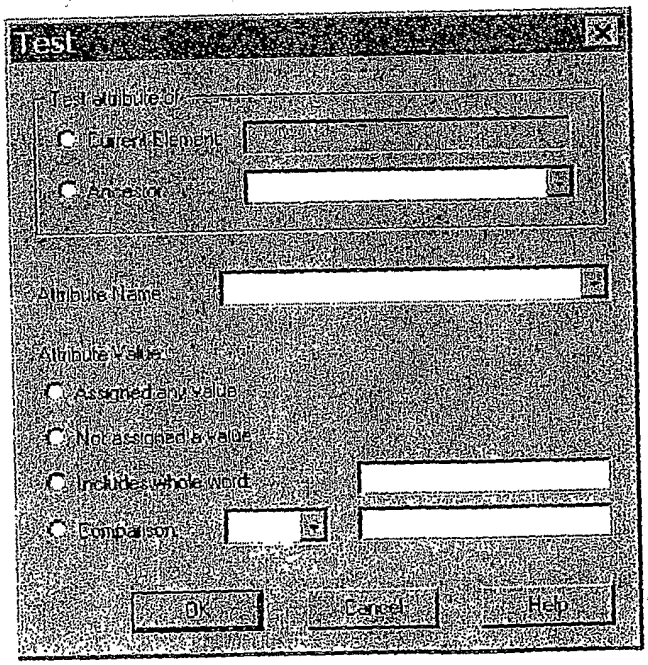


Figure 17

200

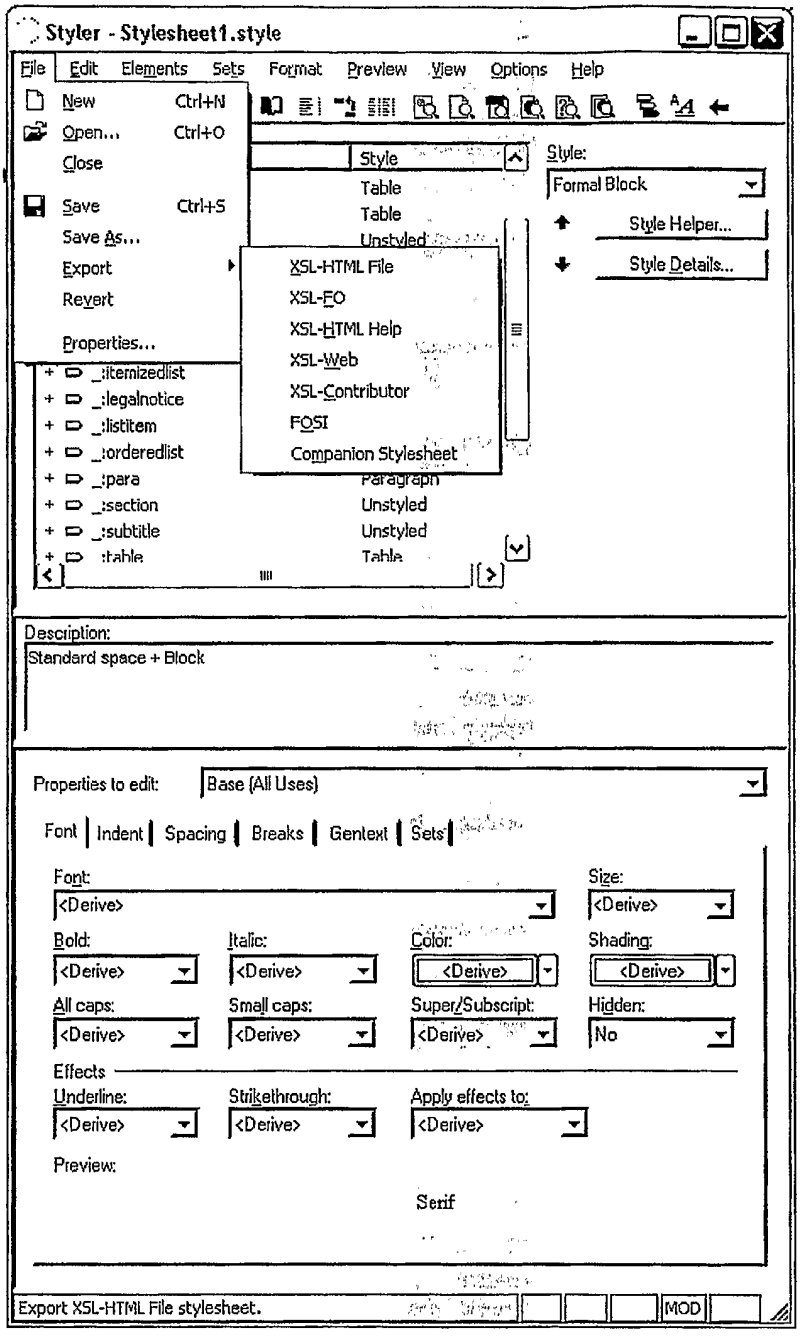


FIGURE 18

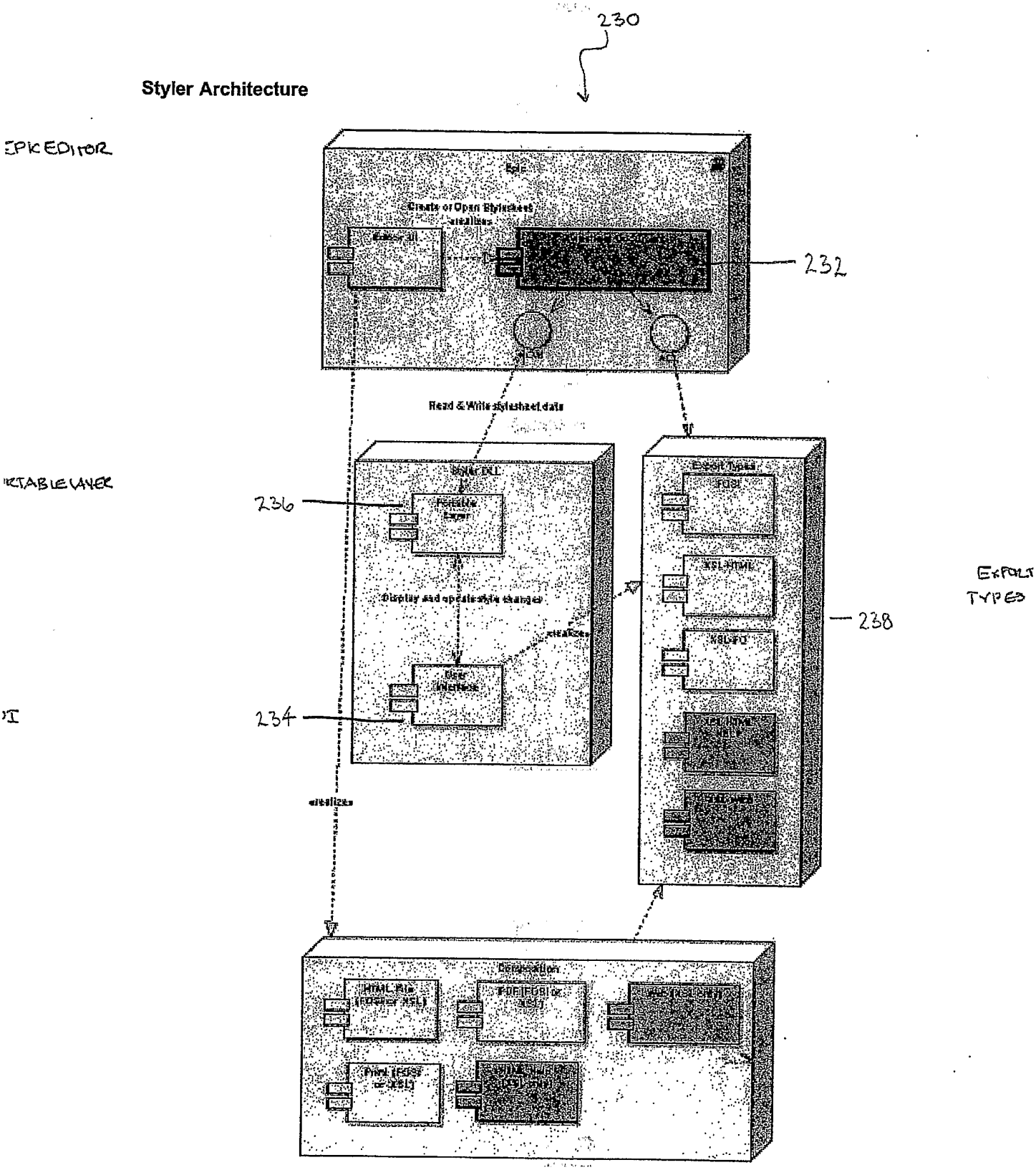


FIGURE 19

Base Classes and Stylesheet-level Classes

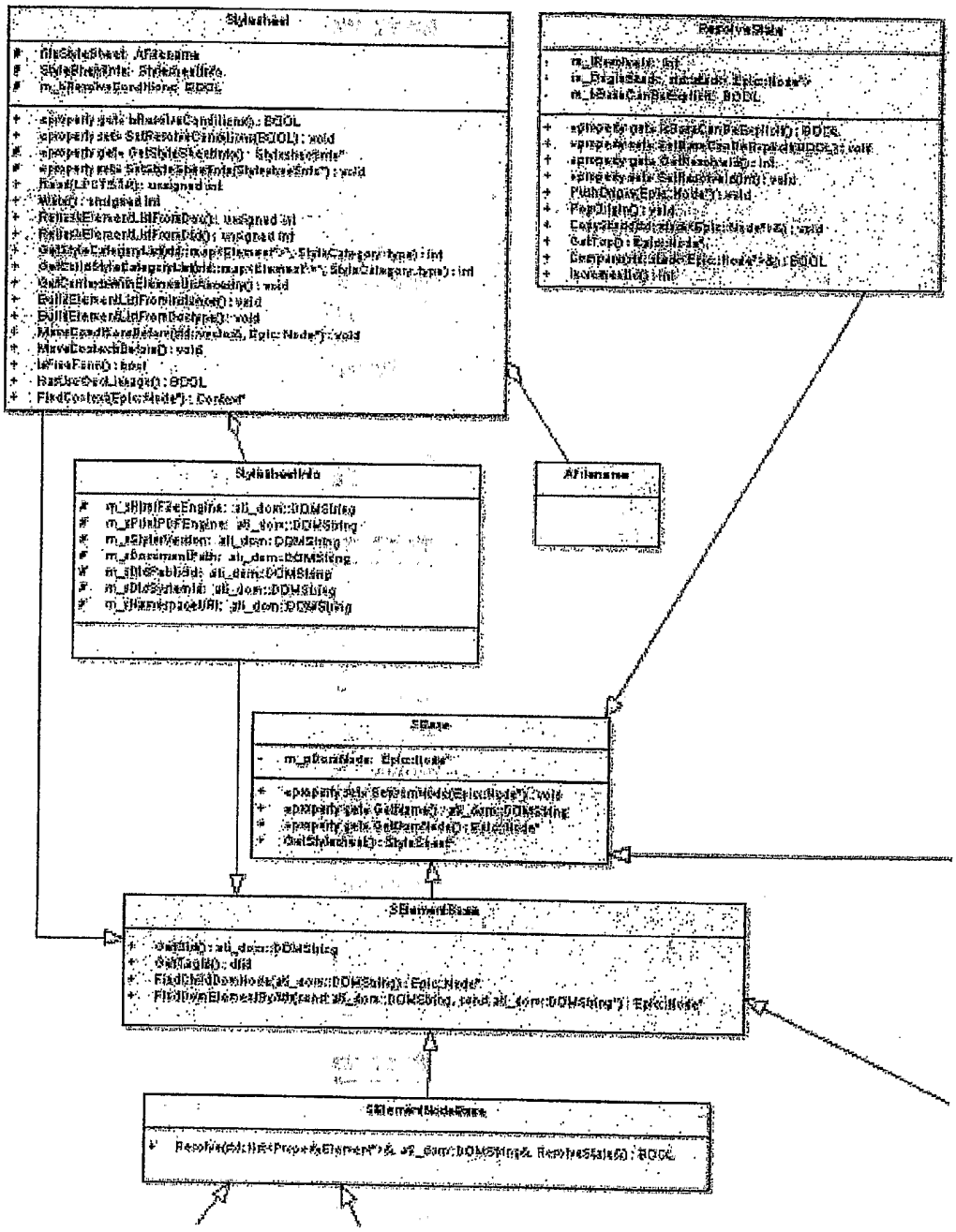
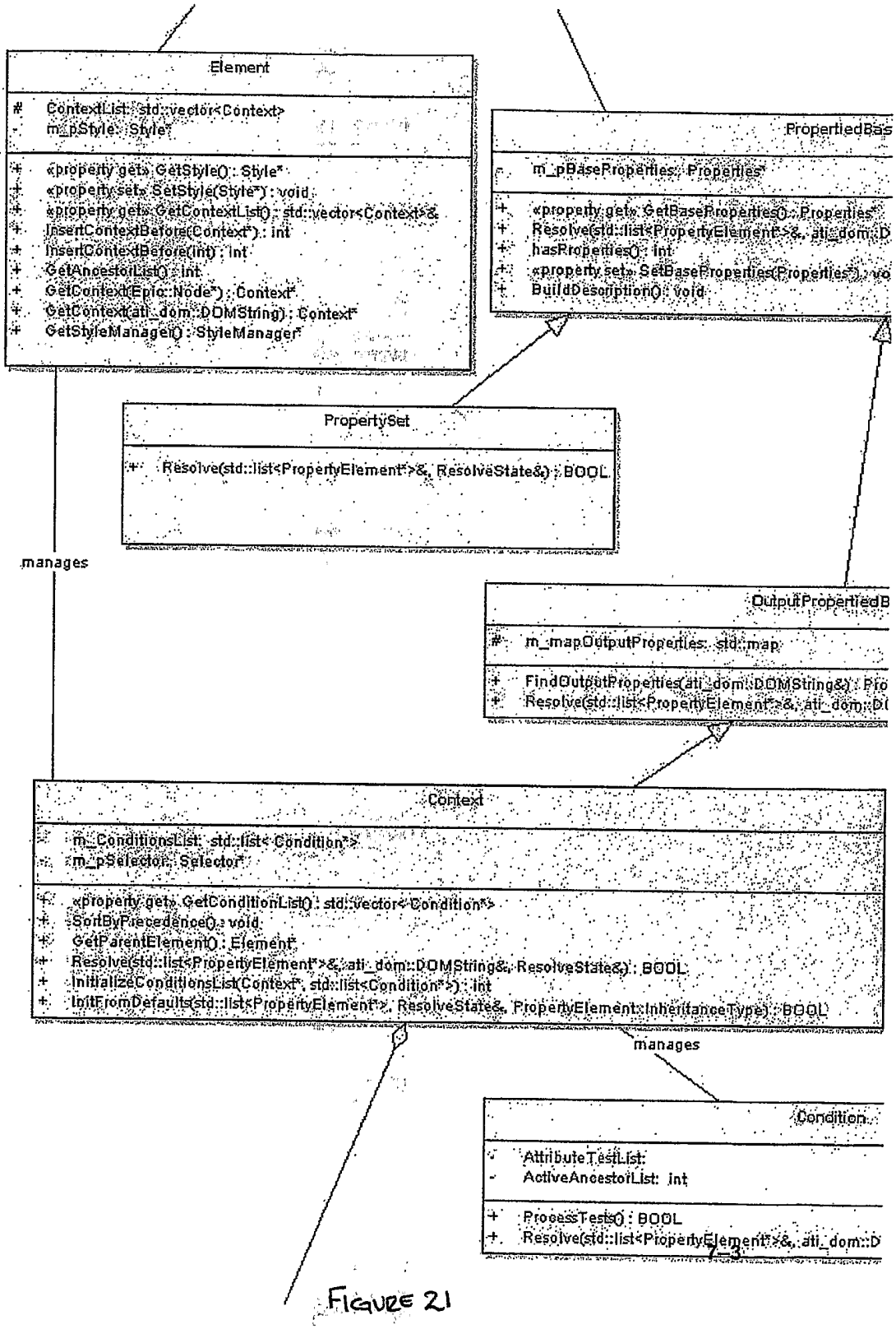


FIGURE 20

IElementNode Subclasses



7-3

FIGURE 21

Property-related Classes

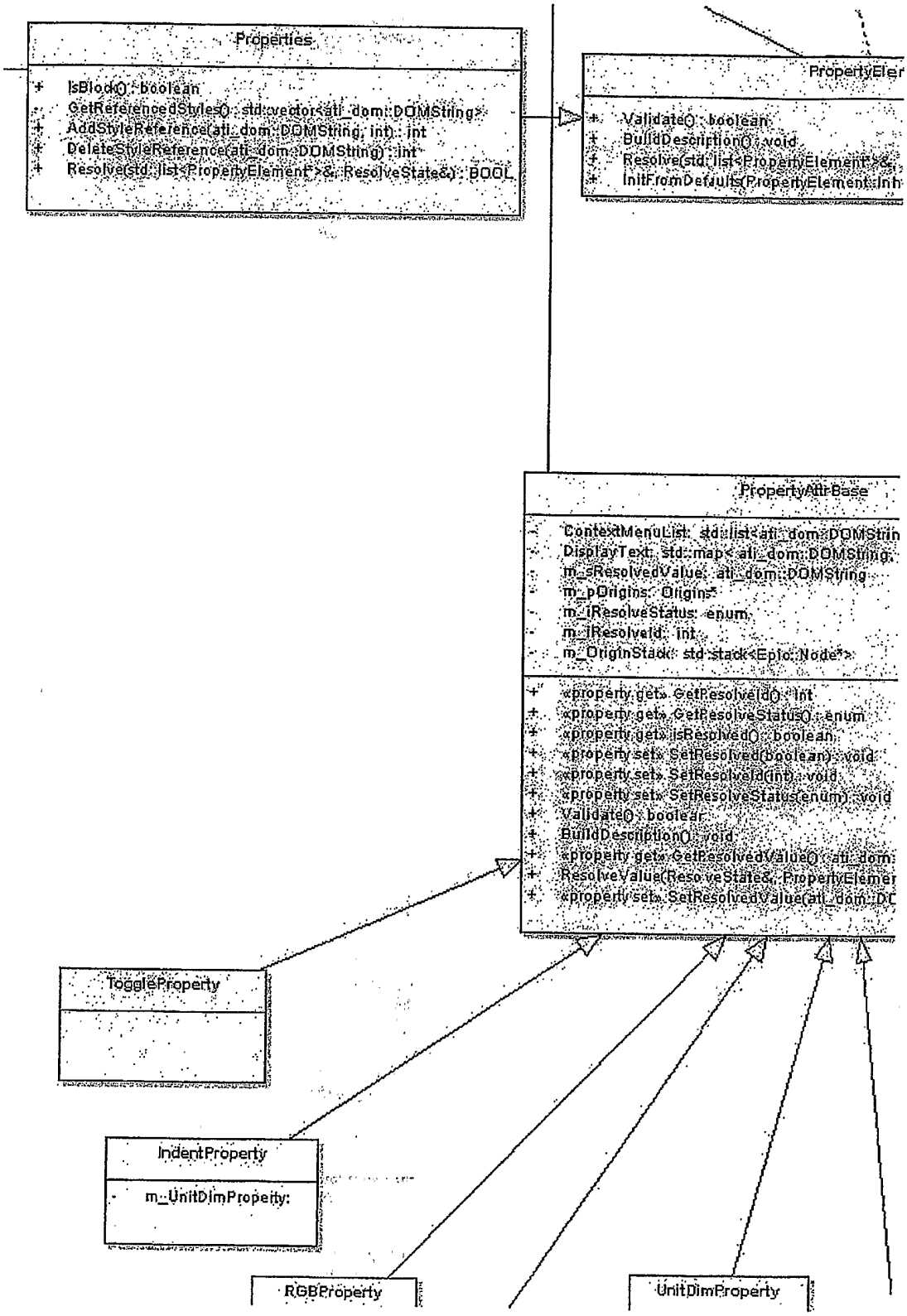


FIGURE 22

IPropertyElementBase Subclasses

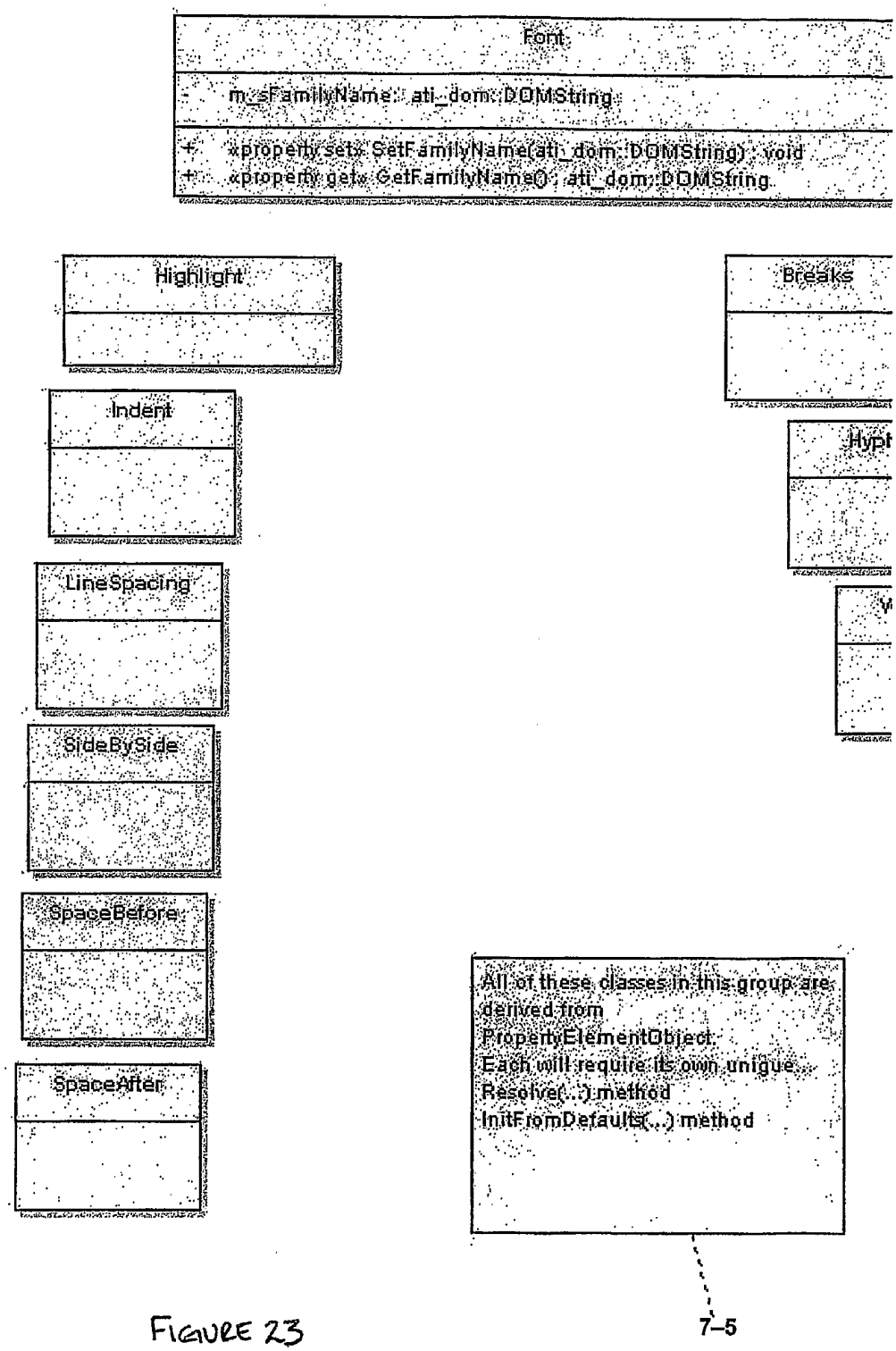


FIGURE 23