***Avertissement :*** *La seule raison d'être du texte qui suit est de remplir un espace supérieur à une page de texte, de manière à illustrer le fonctionnement des éléments **XSL-FO** et de certaines de leurs propriétés.*

### *3. Introduction to Formatting*

*The aim of this section is to describe the general process of formatting, enough to read the area model and the formatting object descriptions and properties and to understand the process of refinement.*

*Formatting is the process of turning the result of an XSL transformation into a tangible form for the reader or listener. This process comprises several steps, some of which depend on others in a non-sequential way. Our model for formatting will be the construction of an area tree, which is an ordered tree containing geometric information for the placement of every glyph, shape, and image in the document, together with information embodying spacing constraints and other rendering information; this information is referred to under the rubric of traits, which are to areas what properties are to formatting objects and attributes are to XML elements. [4 Area Model] will describe the area tree and define the default placement-constraints on stacked areas. However, this is an abstract model which need not be actually implemented in this way in a formatter, so long as the resulting tangible form obeys the implied constraints. Constraints might conflict to the point where it is impossible to satisfy them all. In that case, it is implementation-defined which constraints should be relaxed and in what order to satisfy the others.*

*Formatting objects are elements in the formatting object tree, whose names are from the XSL namespace; a formatting object belongs to a class of formatting objects identified by its element name. The formatting behavior of each class of formatting objects is described in terms of what areas are created by a formatting object of that class, how the traits of the areas are established, and how the areas are structured hierarchically with respect to areas created by other formatting objects. [6 Formatting Objects] and [7 Formatting Properties] describe formatting objects and their properties.*

*Some formatting objects are block-level and others are inline-level. This refers to the types of areas which they generate, which in turn refer to their default placement method. Inline-areas (for example, glyph-areas) are collected into lines and the direction in which they are stacked is the inline-progression-direction. Lines are a type of block-area and these are stacked in a direction perpendicular to the inline-progression-direction, called the block-progression-direction. See [4 Area Model] for detailed decriptions of these area types and directions.*

*In Western writing systems, the block-progression-direction is "top-to-bottom" and the inline-progression-direction is "left-to-right". This specification treats other writing systems as well and introduces the terms "block" and "inline" instead of using absolute indicators like "vertical" and "horizontal". Similarly this specification tries to give relatively-specified directions ("before" and "after" in the block-progression-direction, "start" and "end" in the inline-progression-direction) where appropriate, either in addition to or in place of absolutely-specified directions such as "top", "bottom", "left", and "right". These are interpreted according to the value of the writing-mode property.*

*Central to this model of formatting is refinement. This is a computational process which finalizes the specification of properties based on the attribute values in the XML result tree. Though the XML result tree and the formatting object tree have very similar structure, it is helpful to think of them as separate conceptual entities. Refinement involves*

- *propagating the various inherited values of properties (both implicitly and those with an attribute value of "inherit"),*
- *evaluating expressions in property value specifications into actual values, which are then used to determine the value of the properties,*
- *converting relative numerics to absolute numerics,*
- *constructing some composite properties from more than one attribute*

*Some of these operations (particularly evaluating expressions) depend on knowledge of the area tree. Thus refinement is not necessarily a straightforward, sequential procedure, but may involve look-ahead, back-tracking, or control-splicing with other processes in the formatter. Refinement is described more fully in [5 Property Refinement / Resolution].*

*To summarize, formatting proceeds by constructing an area tree (containing areas and their traits) which satisfies constraints based on information contained in the XML result tree (containing element nodes and their attributes). Conceptually, there are intermediate steps of constructing a formatting object tree (containing formatting objects and their properties) and refinement; these steps may proceed in an interleaved fashion during the construction of the area tree.*

#### *3.1 Conceptual Procedure*

*This subsection contains a conceptual description of how formatting could work. This conceptual*

*procedure does not mandate any particular algorithms or data structures as long as the result obeys the implied constraints.*

*The procedure works by processing formatting objects. Each object, while being processed, may initiate processing in other objects. While the objects are hierarchically structured, the processing is not; processing of a given object is rather like a co-routine which may pass control to other processes, but pick up again later where it left off. The procedure starts by initiating the processing of the fo:root formatting object.*

*Unless otherwise specified, processing a formatting object creates areas and returns them to its parent to be placed in the area tree. Like a co-routine, it resumes control later and initiates formatting of its own children (if any), or some subset of them. The formatting object supplies parameters to its children based on the traits of areas already in the area tree, possibly including areas generated by the formatting object or its ancestors. It then disposes of the areas returned by its formatting object children. It might simply return such an area to its parent (and will always do this if it does not generate areas itself), or alternatively it might arrange the area in the area tree according to the semantics of the formatting object; this may involve changing its geometric position. It terminates processing when all its children have terminated processing (if initiated) and it is finished generating areas.*

*Some formatting objects do not themselves generate areas; instead these formatting objects simply return the areas returned to them by their children. Alternatively, a formatting object may continue to generate (and return) areas based on information discovered while formatting its own children; for example, the fo:page-sequence formatting object will continue generating pages as long as it contains a flow with unprocessed descendants.*

*Areas returned to an fo:root formatting object are page-viewport-areas, and are simply placed as children of the area tree root in the order in which they are returned, with no geometrical implications.*

*As a general rule, the order of the area tree parallels the order of the formatting object tree. That is, if one formatting object precedes another in the depth-first traversal of the formatting object tree, with neither containing the other, then all the areas generated by the first will precede all the areas generated by the second in the depth-first traversal of the area tree, unless otherwise specified. Typical exceptions to this rule would be things like side floats, before floats, and footnotes.*

*At the end of the procedure, the areas and their traits have been constructed, and they are required to satisfy constraints described in the definitions of their associated formatting objects, and in the area model section. In particular, size and position of the areas will be subject to the placement and spacing constraints described in the area model, unless the formatting object definition indicates otherwise.*

*The formatting object definitions, property descriptions, and area model are not algorithms. Thus, the formatting object semantics do not specify how the line-breaking algorithm must work in collecting characters into words, positioning words within lines, shifting lines within a container, etc. Rather this specification assumes that the formatter has done these things and describes the constraints which the result is supposed to satisfy.*