DSSSL Lite Specification Preliminary Draft

$Id: lite.html,v 1.1 1994/11/24 18:54:01 jjc Exp $

It is intended that this specification will eventually develop to the point where it is self-contained, but at the moment some of it will make sense only when read in conjunction with the DIS.

DSSSL Lite describes an application of DSSSL standardized by SGML Open. It is planned that a conforming implementation of DSSSL Lite should also conform to the DSSSL IS.

DSSSL Lite, like DSSSL, is structured as a required core, plus a number of optional features.

Conceptual model

The desired formatting is described in terms of a tree of flow objects. Each flow object describes a task to be performed by the formatter. The children of a flow object are flow objects that describe sub-tasks whose results are used by the formatter to perform the task described by the parent flow object. The flow object tree is constructed from the input SGML document using the formatting specification. This flow object tree constitutes the input to the formatter.

Flow objects of certain classes cannot have children associated with them. These are called atomic flow objects. Flow objects of other classes are called compound flow objects. These flow objects may have one or more sequences of children associated with them. Each sequence of children of a flow object is called a stream. There is no order defined between two flow objects which are children of the same parent flow object but which occur in different streams. Each stream of flow object is attached to a named port of the flow object. The set of ports that a flow object depends on its class and in some cases also its characteristics. One port of each flow object is distinguished as the principal port.

Each flow object has a class that specifies the type of task to be performed and a number of named characteristics that further specify the task.

The result of a formatting a flow object, that is performing the formatting task that the flow object describes, is a sequence of areas. An area is a rectangular box. An area may in turn contain sub-areas. An area can be imaged on a presentation medium. Imaging an area causes any sub-areas to be imaged and may also cause marks to be made on the presentation medium. Formatting a compound flow objects uses the areas produced by formatting its children flow objects to produce its own areas.

Areas are of two types inline areas and display areas. These areas differ in how they are placed.

The formatting specification can be thought of as describing a function that has two arguments, a node in the input tree and a characteristic set which associates values with certain characteristic names (the inherited characteristics), and returns a sequence of flow objects. This function is called the global constructor function. The flow object tree is constructed by applying the global constructor function to the root node of the input tree and a character set in which every inherited characteristic has its initial value; this must return a flow object sequence containing a single flow object.

The global constructor function is specified with a set of style specification. Each style specification consists of a pattern and a constructor function. When applied to a node the global constructor function finds the style specification with the most specific pattern which matches the node, and then applies the style specification's constructor function to the node. The characteristic set that was passed to the global constructor function is passed on as an implicit argument to the style specification's constructor function.

The constructor function can construct a flow object by specifying its class and characteristics. Each inherited characteristic gets its value from the implicit characteristic set argument. Characteristics that are not inherited can either be specified explicitly or, in most cases, can be defaulted. When constructing a compound flow object, a flow object sequence can be specified as the content. In many cases the content of the flow object will be the flow object sequence containing the results of recursively applying the global constructor function to the content of the node. This is also the default constructor function for any nodes in the input document that do not match the pattern in any style specification.

The flow objects in flow object sequences returned by the constructor function can be labelled with the name of a port. When a flow object occurs in a flow object sequence specified as the content of a compound flow object, then if the flow object is not labelled, it is added to the stream attached to the principal port of the

compound flow object; otherwise if the label is the same as the name of one of the ports of the compound flow object, the labelled flow object is added to that port's stream; otherwise the labelled flow object will float out of the content of the compound flow object. Thus a specification of the construction of a single flow object is treated as the specification of a flow object sequence: the constructed flow object is the first member of the sequence; the remaining members are labelled flow objects that have floated out of the content of the constructed flow object.

Style specifications

A style specification of the form

```
(element qualified-gi
         constructor)
```

specifies a pattern that matches elements that match the *qualified-gi*; this is either a generic identifier `P` or a list of generic identifiers `(UL LI)`. In the latter case an element matches if its generic identifier is equal to the last generic identifier in the list and the generic identifier of its parent is equal to the last but one generic identifier in the list and so on. Strings can be used instead of symbols; this is necessary if the generic identifier is not a legal identifier in the expression language (this can only happen with variant concrete syntaxes).

Note that it is possible for a *constructor* to give different treatment to elements based on their attributes or ancestry.

A longer *qualified-gi* is treated as more specific that a shorter one. If more than one identical *qualified-gi* are specified then only the first is used; this allows users to override style specifications for elements.

A style specification of the form

```
(id identifier
    constructor)
```

specifies a pattern that matches an element whose unique identifier is equal to *identifier*. The *identifier* is either a symbol or a string. This kind of style specification is treated as more specific than any specified using `element`.

A style specification of the form

```
(root constructor)
```

specifies a pattern that matches the SGML document node which contains the document element.

There is an implicit style specification for characters and sdata entities which constructs a character flow object.

Constructors

A constructor in a style–specification is an expression that results an object of type flow–object–sequence.

The function `(process-children)` returns the flow–object–sequence that results from appending the flow–object–sequences that result from applying the global constructor function to all the children of the current element in order. This is the default constructor for an element to which no style specification is applicable. This function shall not be called more than once for any node.

A flow–object–sequence can be constructed using an expression with the following form:

```
(flow-object-class-name keyword-argument-list)
```

The *flow-object-class-name* must be the name of a pre–defined flow object class, or must be have declared as an application–defined flow object class.

The *keyword-argument-list* consists of a sequence of pairs of keywords and expressions. The following keywords are allowed:

- A keyword which is the name of an inherited characteristic. In this case the argument is evaluated with respect to the current characteristic set to yield a value. A new characteristic set is constructed from the current characteristic set by replacing the value of the specified characteristic with this new value, and this characteristic set is used to evaluate the remaining keyword arguments. Keyword arguments corresponding to inherited characteristics are evaluated in the order specified before any other arguments.
- A keyword which is the name of a non–inherited characteristic applicable to flow objects of the class being constructed.
- `content:` specifies the content of the constructed flow object. The expression must evaluate to a flow object sequence. The default is `(process-children)`.
- `break-before:` specifying the type of break that is to occur before the constructed flow object. The argument must evaluate to one of symbols `line`, `paragraph`, `page`. This causes a flow object to be inserted before the constructed flow object which will produce a special display area, that will cause a break of the specified type.

  A `paragraph` break can be specified only within a paragraph. It cause the paragraph flow object to start a new paragraph. This differs from a line break in that the first line after a paragraph break will be indented as specified by the `first-line-start-indent:` characteristic. Furthermore the paragraph uses the characteristics associated with the inserted paragraph break flow object in formatting the portion of the content of the paragraph flow object after the paragraph break and before any subsequent paragraph breaks.
- `break-after:`
- `space-before:` specifies the length of a blank display area to be inserted before the constructed flow object. This argument causes a flow object to be inserted before the constructed flow object that will produce a single blank display area. The argument can be a length or a length–spec or a display–space.
- `space-after:` specifies the length of a blank display area to be inserted after the constructed flow object. This argument causes a flow object to be inserted after the constructed flow object that will produce a single blank display area. The argument can be a length or a length–spec or a display–space.
- `escapement-space-before:` specifies the length of a blank inline area to be inserted before the constructed flow object. This argument causes a flow object to be inserted before the constructed flow object that will produce a single blank inline area. The argument can be a length or a length–spec or an inline–space.
- `escapement-space-after:` specifies the length of a blank inline area to be inserted after the constructed flow object. This argument causes a flow object to be inserted after the constructed flow object that will produce a single blank inline area. The argument can be a length or a length–spec or an inline–space.
- `label:` specifies a symbol with which the constructed flow object should be labeled. Flow objects inserted by the break–before, break–after, space–before and space–after: keywords are also labeled with this.

A flow object sequence can also be constructed using the following form:

`(sequence` *`keyword-argument-list`*`)`

This returns the flow object sequence specified in the `content:` argument. The *`keyword-argument-list`* has the same form as with flow object class names. Since `sequence` is not a flow object no non–inherited characteristics can be specified. The `label:` argument has the effect of labelling all unlabelled flow objects in the content.

Public identifiers

DSSSL uses ISO 9070 public identifiers. I think we need to devise mappings between these and URNs or URLs.

Characters and SDATA entities

Describe model for handling characters and SDATA entities.

Characteristics

Inherited characteristics

Character level

- `font-family-name:`
- `font-weight:`
- `font-posture:`
- `font-proportionate-width:`
- `font-size:`
- `score:`
- `placement-offset:` Vertical offset from baseline. Applies to any inlined flow object.
- `color:` Applies also to rules.

Paragraph level

- `start-indent:` left indent (start means at the start in the writing-mode direction) Applies to lines produced by paragraphs. Also to tables, external graphics, displayed rules
- `first-line-start-indent:` Applies to first line produced by each paragraph.
- `end-indent:` (end means at the end in the writing-mode direction) Applies to lines produced by paragraphs. Also to tables, external graphics.
- `quadding:` Applies to lines produced by paragraphs.
- `display-alignment:` Similar to quadding, but justify not allowed. Applies to tables, external graphics, displayed rules
- `verbatim?:`
- `pre-line-spacing:` Component of line spacing before baseline
- `post-line-spacing:` Component of line spacing after baseline

Other

- `background-color:` Applies to root (specifies window background).

Common non-inherited characteristics

- `keep-with-previous?:` Applies to displayed flow objects.
- `keep-with-next?:` Applies to displayed flow objects.

Flow object classes

Root flow object class

Paragraph flow object class

Represents a logical paragraph. Can contain more than one block of text.

Labeled-item flow object class

Used for lists, where each list item is labelled (marked).

Character flow object class

`(literal` *`string`* `...)`

Rule flow object class

The rule flow object class is used to create horizontal and vertical rules.

Leader flow object class

The leader flow object class is used to create leaders. The content is repeated as many times as is necessary to fill the available space on the line.

External graphic flow object class

Flow object classes for tables

table table-part table-row inline-table-cell display-table-cell

Page layout

DSSSL Lite provides the following very simple page layout capabilities:

- single column only; no footnotes; no floats
- one line headers and footers
- ability to specify center/left/right/inside/outside components of headers as combination of literal text, source document content and page number
- specifiable top/bottom/left/right/header/footer margins on a per document basis
- ability to specify that content of element should start a new page
- ability to specify that a flow object should be kept with the preceding or following flow object and that group of flow objects should be kept on same page

In DSSSL terms there would be a simple variant of the page-sequence flow object with a fixed page model that would be parameterized by characteristics of the flow object.

Implementations that don't do pages would just ignore all of this.

Hypertext features

These features would not be standardized in DSSSL, although they would be specified in a way that is DSSSL conformant by using application defined flow objects and characteristics defined by the SGML Open application.

The `iconify` flow object would appear as an icon, which when clicked would display its content in a separate window.

Also need flow objects/characterics for linking.

Expression language

Lexical structure

Comments start with `;` and continue to end of line.

White-space can be used freely to separate tokens.

Language is case-sensitive.

Syntax

An expression is one of the following:

- boolean
- number
- character
- string
- keyword
- glyph-identifier
- quotation
- variable-reference
- function-call
- if-expression
- cond-expression
- case-expression
- and-expression
- or-expression

General purpose data types

- boolean
- number/quantity
- char
- string
- symbol
- keyword
- function (in this subset all functions are primitive or external) Need external-function function.
- list (no car, cdr, cons); just list and list-ref.

Special purpose data types

- color
- color–space
- display–space
- inline–space
- flow–object–sequence
- length–spec
- glyph–id

Querying

Counting

`(child-number)`

Returns child number of current element, that is, the number of siblings of the current element that are before or equal to the current element and that have the same gi as the current element.

`(ancestor-child-number` *e*`)`

Returns child number of nearest ancestor of current element whose gi is *e*.

`(hierarchical-number` *e1 e2 ... en*`)`

Returns a list of numbers; last member is child number of nearest ancestor of current element with gi *en*; next to last member is child number of nearest ancestor of that element with gi *en–1*.

`(hierarchical-number-recursive` *e*`)`

Returns a list of numbers; last member of list is child number of nearest ancestor of current element with gi *e*, next to last member is child number of nearest ancestor of that element with gi *e* and so on. Note that the length of this list is the level of nesting of *e*.

`(element-number)`

Returns the number of elements before or equal to the current element with the same gi as the current element. Note that this includes elements anywhere in the document hierarchy.

`(element-number-list` *e1 e2 ... eN*`)`

Returns a list of N numbers, where the *i*th number is the number of elements that:

- are before or equal to the current element,
- have gi *e(i)*,
- and, if *i* is greater than 1, are after the last element before the current element with gi *e(i–1)*

(Informally the counter for each argument is reset by the start of the previous argument.) Note that an element is considered to be after its parent. This could be used to number headings in HTML. It could also be used to number footnotes sequentially within a chapter (using the last number in the list). (For efficiency's sake it might be necessary to require that all arguments are constants, but I hope not.)

Querying for attributes

The value returned by these functions is either a string, giving the value of the attribute of #f.

`(attribute` *name*`)`

Returns the attribute *name* of the current element, or #f if the current element has no such attribute or the attribute is implied.

`(inherited-attribute` *name*`)`

Returns the attribute *name* of the current element or of the nearest ancestor of the current element for which this attribute is present with a non–implied value, or #f if there is no such element.

`(ancestor-attribute` *gi name*`)`