CSLDSSSL - An Annotatable DSSSL Stylesheet

Table of Contents

1. Introduction

This environment models and supports an annotatable DSSSL stylesheet. Annotations take the form of formal documentation, formatted as a printed manual, split into sections, sub-sections and sub-sub-sections. The public identifier for the document model (available at `http://www.CraneSoftwrights.com`) is one of either of the two following values supported in the catalogue:

```
+//IDN CraneSoftwrights.com::CSL//DTD An Annotatable DSSSL Stylesheet//EN"
```

```
+//ISBN 1-894049::CSL//DTD An Annotatable DSSSL Stylesheet//EN
```

When one writes DSSSL scripts using the Annotatable DSSSL Stylesheet document model, this environment provides a DSSSL stylesheet to use to document that script in three renderings: print, HTML, and an unannotated DSSSL distillation.

This environment's documentation stylesheet uses standardized DSSSL print semantics for the print rendering, and custom SGML syntax semantics as defined in the DSSSL engine named JADE (`http://www.jclark.com/jade`) for the HTML and raw DSSSL renderings. Refer to the command-line documentation 2.1. Command-line Manipulated Variables for details on how the invocation uses the setting of different variables to engage the different renderings from the same script.

1.1. Assumptions

1.1.1. Print Rendering

This version of the stylesheet assumes imperial measurements for the page dimensions. All values can be changed to metric measurements in the root construction rule 4. Root Construction, or dynamically at invocation with a command line option documented in 2.1. Command-line Manipulated Variables.

1.1.2. HTML Rendering

This version of the stylesheet assumes the HTML 3.2 Recommendation for markup, with the exception of including by default a colour background attribute for a table which is not defined in the recommendation. Refer to the command-line documentation 2.1. Command-line Manipulated Variables for details on how to keep strictly to the HTML 3.2 recommendation.

1.2. Sample Windows Environment

1.2.1. Support Files

The file `csldsssl.soc` is an SGML-Open standard format catalogue of the public identifiers for the environment.

The file `sample.sgm` is a simple example stylesheet.

The file `csldsssl.sgm` is the DSSSL script used to produce the documentation for an annotated stylesheet.

The file `dsssl.dtd` is architectural meta-DTD for DSSSL.

Two entity files `iso-lat1.ent` and `iso-num.ent` include the SDATA declarations for entities helpful in writing the prose of the documentation in an annotated stylesheet.

1.2.2. Sample Script Use

Consider a DSSSL stylesheet that is an instance of the CSLDSSSL document model is named `sample.sgm` (the extension chosen because of the rich element structure), and the document being processed is `mydoc.sgm`, then the JADE invocation is (in part):

```
jade -d sample.sgm -c csldsssl.soc mydoc.sgm
```

1.2.3. Documentation of the Sample Script

The batch file `csldsssl.bat` is a representative invocation of this documentation environment in Windows, using JADE as an example DSSSL engine. It assumes the DSSSL script is in an SGML instance with the filename extension of `.SGM`.

There is one mandatory argument to the invocation batch file: the name (without `.SGM` suffix) of the instance to be processed. Up to 8 other arguments for JADE may be included after the mandatory argument.

For the one `.SGM` input, three outputs are created: an `.HTM` file for browsing, an `.RTF` file for printing, and a `.DSL` file for information. Note the `.DSL` file is only a distillation of the DSSSL code fragments from the annotated stylesheet, and itself can be used with the unannotated version of the stylesheet. It is not necessary to create the `.DSL` file to use with a DSSSL-conforming engine, because the `.SGM` file itself conforms to the DSSSL architecture.

To create the three documentation files of the `sample.sgm` DSSSL stylesheet, one would execute the one command:

```
csldsssl sample
```

To limit the HTML documentation produced to strictly follow the HTML 3.2 document model, add the setting of a switch:

```
csldsssl sample -V strict
```

1.2.4. Environment Variables

The supplied example invocation batch file has one environment variable dependency, the directory in which the CSLDSSSL documentation script file is found. For example:

```
SET csldsssl=c:\csldsssl\
```

Note the directory name is suffixed with a subdirectory separator character (backslash for Windows).

The invocation also assumes the JADE executable is on the path.

1.3. Creation of CSLDSSSL Documentation

This documentation file is, itself, the output of the environment's stylesheet applied to itself.

The invocation used of the sample batch file to produce this documentation is:

```
csldsssl csldsssl
```

2. Definitions

2.1. Command-line Manipulated Variables

When using JADE, these variable are turned to true with the following command line option:

```
-V variable-name
```

The `noscript` variable defaults to false which will not suppress the printing of script text in the documentation. When true, the script text is suppressed from the printing of the documentation.

```
(define noscript #f)    ;set to true to suppress script text in output docs
```

The `html` variable defaults to false which will produce standard DSSSL flow objects. When true, the script will produce non-standard SGML flow objects supported by JADE.

```
(define html #f)    ;set to true to produce SGML flow objects of HTML
```

The `strict` variable defaults to false allowing non-standard HTML 3.2 syntax to be generated.

```
(define strict #f)  ;set to true to produce strict HTML 3.2 syntax
```

The `metric` variable defaults to false thereby engaging imperial measurements when using print semantics.

```
(define metric #f)  ;set to true to use metric measurements
```

2.2. Dimensions

```
                                    ;metric ;imperial
(define page-width          (if metric  210mm   8.5in))
(define page-height         (if metric  297mm   11in))
(define left-margin         (if metric  13mm    .5in))
(define right-margin        (if metric  13mm    .5in))
(define header-margin       (if metric  13mm    .5in))
(define footer-margin       (if metric  13mm    .5in))
(define top-margin          (if metric  25mm    1in))
(define bottom-margin       (if metric  25mm    1in))
(define box-start-indent    (if metric  6mm     .25in))
(define box-end-indent      (if metric  6mm     .25in))
```

```
(define toc-start-indent    (if metric  13mm    .5in))
(define toc-end-indent      (if metric  20mm    .75in))
(define toc-item-indent     (if metric  20mm    .75in))
(define index-col-sep       (if metric  6mm     .25in))
```

2.3. Common Strings

```
                                        ;metric ;imperial
(define toc-string          "Table of Contents")
(define defn-string         "Index of Definitions")
```

2.4. Print Semantics Font Related Variables

```
(define default-font-size    12pt)          ;main body of text
(define title-font-sizes     '(18pt 17pt 16pt 15pt 14pt 13pt))
(define main-title-font-size 16pt)
(define part-title-font-size 14pt)
(define serif-font           "Times New Roman") ;should be "iso-serif"
(define monospaced-font-size 10pt)
(define monospaced-font      "Courier New")     ;should be "iso-monospc"
```

2.5. Non-standard Function Definitions

Note these function may not be supported by all DSSSL engines; the declarations used are those recognized by JADE.

The `(debug)` function is available to be used for diagnostic purposes. It may be that the function isn't actually called anywhere for a particular revision of the script.

```
(define debug
  (external-procedure "UNREGISTERED::James Clark//Procedure::debug"))
```

The `(if-first-page)` function is used to remove the redundant page header on the first page of the printed output.

```
(define if-first-page
  (external-procedure "UNREGISTERED::James Clark//Procedure::if-first-page"))
```

These `JADE SGML Syntax Flow Objects` declare the non-DSSSL standard flow objects for emitting SGML syntax as supported by JADE.

```
(declare-flow-object-class element
  "UNREGISTERED::James Clark//Flow Object Class::element")
(declare-flow-object-class empty-element
  "UNREGISTERED::James Clark//Flow Object Class::empty-element")
(declare-flow-object-class document-type
  "UNREGISTERED::James Clark//Flow Object Class::document-type")
(declare-flow-object-class processing-instruction
  "UNREGISTERED::James Clark//Flow Object Class::processing-instruction")
```

```
(declare-flow-object-class entity
  "UNREGISTERED::James Clark//Flow Object Class::entity")
(declare-flow-object-class entity-ref
  "UNREGISTERED::James Clark//Flow Object Class::entity-ref")
(declare-flow-object-class formatting-instruction
  "UNREGISTERED::James Clark//Flow Object Class::formatting-instruction")
(declare-characteristic preserve-sdata?
  "UNREGISTERED::James Clark//Characteristic::preserve-sdata?"
  #t)
```

2.6. Inherited Characteristic Initial Values

```
(declare-initial-value font-size        default-font-size)
(declare-initial-value font-family-name serif-font)
(declare-initial-value min-leading      0pt) ;stretch line height to fit
(declare-initial-value quadding         'justify)
```

3. Support Procedures

The `construct` function captures the construction rule for both print and SGML semantics without replicating the condition test in the syntax of every construction rule.

```
(define (construct print sgml)
    (if html sgml print))       ;use SGML semantics if HTML requested.
```

The `make-elem` and `make-emptelem` functions create SGML flow objects with the optional supply of attributes.

```
(define (make-elem gi #!optional (a '()) (content (process-children)))
    (make element
        gi:         gi
        attributes: a
        content))

(define (make-emptelem gi #!optional (a '()))
    (make empty-element
        gi:         gi
        attributes: a))
```

The `part-depth`, `part-list` and `part-list-name` functions calculate the depth of parts and subparts of the given node, and the formatted string.

```
(define (part-depth nd)
    (length (hierarchical-number-recursive "part" nd)))

(define (part-list nd delim suffix)
    (string-append
        (format-number-list (hierarchical-number-recursive
                                "PART"
                                (if (equal? (gi nd) "PART")
```

```
                                    (node-list-first (children nd))
                                    nd))
                            "1" delim)
        (if suffix delim "")))

(define (part-list-name nd prefix)
    (string-append (if prefix "#" "")
                   "P"
                   (part-list (current-node) "-" #f)))
```

The `first-descendant-sosofo` function builds on the assumed lazy evaluation of lists to efficiently return the content of the node of the given generic identifier as a sosofo.

```
(define (first-descendant-sosofo giFind)
    (literal                                            ;make a sosofo
        (data                                           ;of the data
            (node-list-first                            ;of the first
                (select-elements
                    (descendants                        ;of the descendants
                        (if (gi (current-node))
                            (current-node)
                            (node-property 'docelem
                                (current-node))))
                    giFind)))))                         ;with the given name
```

4. Root Construction

```
(root
    (let* ((title-sosofo (first-descendant-sosofo "TITLE")))
        (construct
            (make simple-page-sequence
                page-width:     page-width
                page-height:    page-height
                left-margin:    left-margin
                right-margin:   right-margin
                header-margin:  header-margin
                footer-margin:  footer-margin
                top-margin:     top-margin
                bottom-margin:  bottom-margin
                center-header:  (if-first-page (empty-sosofo) title-sosofo)
                left-footer:    (first-descendant-sosofo "INFO")
                right-footer:   (first-descendant-sosofo "DATE")
                center-footer:  (page-number-sosofo)
                (process-children))
            (sosofo-append              ;assume HTML 3.2
                (make document-type
                    name:      "html"
                    public-id: "-//W3C//DTD HTML 3.2 Final//EN")
                (make formatting-instruction
                    data:      (string-append "<"
                                 "!-- CSL DSSSL Stylesheet - "
```

```
                                "http://www.CraneSoftwrights.com -->
<"
                                "!-- " (time->string (time)) " -->
"))
                (make-elem "html" '()
                    (sosofo-append
                        (make-elem "head" '()       ;HTML meta-data
                            (make-elem "title" '()  ;set to script title
                                title-sosofo))
                        (make-elem "body" '()       ;add documentation
                                (process-children))))))))
```

5. High-level Constructs

The `csldsssl` and `spec` and `part` elements comprise the high-level constructs other than the `dsssl` element.

```
(element csldsssl
    (sosofo-append      ;change the rendered order of the authored information
        (process-first-descendant "title")
        (toc)                                       ;injected table of contents
        (process-first-descendant "spec")
        (index)                                     ;injected index
        (process-first-descendant "meta")))

(element spec
    (process-children))

(element part
    (process-children))
```

6. Display of Raw DSSSL Script

The text of the DSSSL script itself is displayed if the invocation is not engaged to suppress the display, as defined in 2.1. Command-line Manipulated Variables.

```
(element dsssl
    (if (not noscript)          ;if body text is being displayed
        (construct              ;render verbatim
            (make box
                display?: #t
                space-before:        default-font-size
                start-indent:        box-start-indent
                end-indent:          box-end-indent
                keep-with-previous?: #t
                (make paragraph
                    start-indent:     0pt
                    end-indent:       0pt
                    lines:            'asis
                    font-family-name: monospaced-font
```

```
                    font-size:        monospaced-font-size
                    (process-children)))
            (make-elem "table" (if strict   ;then don't set background
                                   '(("border" "1"))
                                   '(("border" "1") ("bgcolor" "#d0ffd0")))
                (make-elem "tr" '()         ;single row and cell
                    (make-elem "td" '()
                        (make-elem "pre")))))
        (empty-sosofo)))        ;suppress it
```

7. Low-level Constructs

The `title`, `para`, `emph`, `partref`, `defn` and `samp` elements comprise the low-level constructs.

```
(element (csldsssl title)                   ;the title of the entire stylesheet
    (construct
        (make paragraph
            font-size:      main-title-font-size
            font-weight:    'bold
            quadding:       'center
            (make score
                type:       'after
                (process-children)))
        (make-elem "center" '()             ;use HTML heading levels
            (make-elem "h1" '()
                (make-elem "u")))))

(element title                              ;the title of a part
    (let* ((depth      (part-depth (current-node)))
           (report     (if (> depth 5)
                           (error (string-append "Part depth is too deep: "
                                                 (number->string depth)
                                                 " (maximum is 5 levels)."))
                           ""))
           (depth-list (part-list (current-node) "." #t))
           (title-content (sosofo-append (literal depth-list)
                                         (literal " ")
                                         (process-children)))
           (part-id    (attribute-string "ID" (parent (current-node)))))
        (construct
            (let* ((pts (list-ref title-font-sizes depth)))
                (make paragraph
                    space-before:    pts
                    font-size:       pts
                    font-weight:     'bold
                    keep-with-next?: #t
                    title-content))
            (make-elem (string-append "h" (number->string (+ depth 1)))
                       '()
                (make-elem "a"
                           (list (list "name"
                                       (part-list-name (current-node) #f)))
```

```
                    title-content)))))

(element para                               ;a typical paragraph
    (construct
        (make paragraph
            space-before:           default-font-size
            keep-with-previous?:    (if (equal? (child-number) 1)
                                        #t
                                        #f)
            (process-children))
        (make-elem "p")))

(element emph                               ;emphasized content
    (construct
        (make sequence
            font-weight: 'bold
            (process-children))
        (make-elem "b")))

(element defn                               ;the name of something being
    (construct                              ;defined in the part
        (make sequence
            font-family-name: monospaced-font
            (process-children))
        (make-elem "samp")))

(element (para pre)                         ;pre-formatted text in a para
    (construct
        (make sequence                      ;don't break the line
            font-family-name: monospaced-font
            (process-children))
        (make-elem "samp")))

(element pre                                ;some pre-formatted text alone
    (construct
        (make paragraph                     ;lines of text on their own
            space-before:           default-font-size
            font-family-name:       monospaced-font
            lines:                  'asis
            keep-with-previous?:    (if (equal? (child-number) 1)
                                        #t
                                        #f)
            (process-children))
        (make-elem "pre")))

(element extref                             ;a reference possibly outside doc
    (construct
        (make sequence
            font-family-name: monospaced-font
            (process-children))
        (make-elem "a" `(("href" ,(attribute-string "HREF"))))))
```

7.1. Part Referencing

A part of the documentation can be referenced, and in so doing, the referenced part number and title is rendered in a hyper-link to the part itself.

```
(element partref
    (with-mode partref                      ;process the referenced element
        (process-element-with-id (attribute-string "IDREF"))))

(mode partref
    (element part
        (process-first-descendant "TITLE")) ;only process the title child

    (element title                          ;display part number information
        (let* ((part-id (attribute-string "ID" (parent (current-node))))
               (content (sosofo-append
                           (literal (part-list (element-with-id part-id)
                                               "." #t))
                           (literal " ")
                           (process-children))))
            (construct                          ;as a hyperlink
                (make score
                    type: 'after
                    (make link
                        destination: (current-node-address)
                        content))
                (make-elem "a"
                           (list (list "href"
                                       (part-list-name
                                            (element-with-id part-id) #t)))
                    content))))
)
```

8. Meta-data Information

Render the information about the script in a mono-spaced font in order to distinguish the information from the body of the document.

```
(element meta               ;separate meta-data from rest with a rule
    (construct
        (empty-sosofo)      ;already placed in the footer
        (sosofo-append
            (make-emptelem "hr" '(("noshade" "noshade")))
            (make-elem "pre" '()
                (process-children)))))

(element date               ;display without ornamentation
    (construct
        (make paragraph
            font-family-name: monospaced-font
            (process-children))
```

```
        (sosofo-append
            (process-children)
            (literal "
")))))

(element info               ;display without ornamentation
    (construct
        (make paragraph
            font-family-name: monospaced-font
            (process-children))
        (sosofo-append
            (process-children)
            (literal "
"))))
```

9. Default Construction Rule

Handle all elements not explicitly handled by reporting an error that there exists no construction rule for the given element.

```
(default                        ;handle all elements not explicitly handled
    (let* ((report              ;dummy needed for assignment, but not used
            (if (equal? (element-number (current-node)) 1)
                (error          ;trigger the DSSSL-defined error function
                    (string-append "A construction rule for elements named '"
                                   (gi (current-node))
                                   "' isn't defined."))
                "")))           ;don't repeat error for every occurence
        (process-children)))
```

10. Table of Contents Processing

The TOC is composed of entries to each of the parts, to all depth levels.

```
(define (toc)
    (let* ((contents
                (with-mode toc
                    (process-node-list
                        (select-elements
                            (descendants
                                (node-property
                                    'docelem
                                    (node-property
                                        'grove-root (current-node))))
                            "PART")))))
        (construct
            (sosofo-append
                (make paragraph
                    space-before:    part-title-font-size
                    space-after:     default-font-size
                    font-size:       part-title-font-size
```

```
                    font-weight:     'bold
                    keep-with-next?: #t
                    (literal toc-string))
                contents)
            (sosofo-append
                (make-elem "h3" '()
                    (make-elem "a" '(("name" "toc"))
                        (literal toc-string)))
                (make-elem "p" '()
                    contents)))))

(mode toc
    (element part
        (process-first-descendant "TITLE"))

    (element title
        (let* ((ref (literal (part-list (current-node) "." #t))))
            (construct
                (make paragraph
                    start-indent:               (+ toc-start-indent
                                                   toc-item-indent)
                    first-line-start-indent:    (- toc-item-indent)
                    end-indent:                 toc-end-indent
                    (make line-field
                        field-width:            toc-item-indent
                        (make link
                            destination: (current-node-address)
                            ref))
                    (make link
                        destination: (current-node-address)
                        (process-children))
                    (make leader)
                    (make link
                        destination: (current-node-address)
                        (current-node-page-number-sosofo)))
                (let* ((name (part-list-name (current-node) #t)))
                    (sosofo-append
                        (let loop ((countdown (* 4
                                                 (part-depth (current-node)))))
                            (if (equal? countdown 0)
                                (empty-sosofo)
                                (sosofo-append
                                    (make entity-ref
                                        name: "nbsp")
                                    (loop (- countdown 1)))))
                        (make-elem "a" `(("href" ,name))
                            (sosofo-append
                                ref
                                (literal " ")
                                (process-children)))
                        (make-emptelem "br"))))))
)
```

11. Index of Definitions Processing

For those scripts choosing to use the `defn` element, an index of definitions is added to the end of the report. Note that this version of the script does not sort the index.

First, be able to determine the precence of any `defn` element in the script.

```
(define (defn-present?)             ;boolean presence of any defn element
    (> (node-list-length
            (node-list-first        ;hopefully save time with lazy evaluation
                (select-elements
                    (descendants
                        (node-property
                            'docelem
                            (node-property
                                'grove-root (current-node))))
                    "DEFN")))
       0))
```

Sort the definitions alphabetically (not supported at this time).

```
(define (sort-entries unsorted)
    unsorted)
```

Format in a similar fashion to the TOC.

```
(define (index)
    (if (defn-present?)
        (let* ((entries-unsorted                            ;get all entries
                    (select-elements
                        (descendants
                            (node-property
                                'docelem
                                (node-property
                                    'grove-root (current-node))))
                        "DEFN"))
               (entries (sort-entries entries-unsorted))    ;determine order
               (contents                                    ;calculate contents
                    (with-mode index
                        (process-node-list entries))))
            (construct
                (sosofo-append
                    (make paragraph
                        space-before:    part-title-font-size
                        space-after:     default-font-size
                        font-size:       part-title-font-size
                        font-weight:     'bold
                        keep-with-next?: #t
                        (literal defn-string))
                    (make table
                        (make table-column width: (table-unit 1))
                        (make table-column width: (table-unit 3))
```

```
                        contents))
                (sosofo-append
                    (make-elem "h3" '()
                        (make-elem "a" '(("name" "index"))
                            (literal defn-string)))
                    (make-elem "table" '()
                        contents))))
        (empty-sosofo)))

(mode index                         ;present a node formatted for an index
    (element defn
        (let* ((part     (ancestor "PART"))
               (part-ref (process-node-list part))
               (addr     (node-list-address part))
               (name     (part-list-name (current-node) #t)))
            (construct
                (make table-row
                    (make table-cell
                        (make paragraph
                            (make link
                                destination: addr
                                (process-children))))
                    (make table-cell
                        (make paragraph
                            start-indent:   index-col-sep
                            (make link
                                destination: addr
                                part-ref))))
                (make-elem "tr" '()
                    (sosofo-append
                        (make-elem "td" '()
                            (make-elem "a" `(("href" ,name))
                                (process-children)))
                        (make-elem "td" '()
                            (make-elem "a" `(("href" ,name))
                                part-ref)))))))

    (element part
        (process-first-descendant "TITLE"))

    (element title
        (sosofo-append
            (literal (part-list (current-node) "." #t))
            (literal " ")
            (process-children)))
)
```

12. Distilling of the DSSSL Script

These rules will process the DSSSL architectural version of the input document. For JADE, this is engaged with the command line option:

```
-A dsssl
```

The end result is a DSSSL stylesheet conforming to the unannotated version of this document model, without any documentation (only the concatenation of all body portions).

Note that code this will not preserve general entities for "<" and "&" characters, or any named general entities such as "RE".

```
(element dsssl-specification
    (sosofo-append
        (make document-type             ;declare the document type
            name:      "style"
            public-id:
              "+//ISBN 1-894049::CSL//DTD An Annotatable DSSSL Stylesheet//EN")
        (process-children)))            ;the body of the specification

(element style-specification            ;satisfy heritage to single body
    (process-children))

(element style-specification-body       ;accommodate missing newlines
    (make formatting-instruction
        data: (string-append "
" (data (current-node)) "
")))

; end of DSSSL
```

Index of Definitions

csldsssl	5. High-level Constructs
spec	5. High-level Constructs
part	5. High-level Constructs
title	7. Low-level Constructs
para	7. Low-level Constructs
emph	7. Low-level Constructs
partref	7. Low-level Constructs
defn	7. Low-level Constructs
samp	7. Low-level Constructs