docscm Documentation

version 0.1

Tony Garnock-Jones

docscm Documentation: version 0.1

by Tony Garnock-Jones

Published Tue Sep 24 16:56:30 BST 2002

Describes installation and use of the **docscm** Scheme documentation extraction and compilation tool.

Table of Contents

Chapter 1. User Guide

docscm is a collection of simple tools for extracting function-, variable-, and record-documentation from Scheme source files, and generating DocBook XML output suitable for further processing. It is written using the Chicken[1] scheme system, and draws inspiration from Javadoc, Doxygen, and Aubrey Jaffer's **schmooz**[2]. While it lends itself particularly to generating API documentation, it can also be used to help automate parts of the normal DocBook processing chain.

1.1. General Operation

Scheme files containing specially-formatted comments are read by **extract-doc**, which outputs a loose form of SXML[3]. This output is then converted to XML by **sxml2xml**. Finally, the XML output fragments can be included in DocBook documents using standard SGML entity inclusion. Some rules for GNU make are included which automate parts of the documentation compilation process.

```
;;@
;; Returns <varname>arg</varname> multiplied by two.
(define (double arg) (* arg 2))
```

docscm comments start on a line of their own, with any amount of leading whitespace, *n* leading semicolons, and an at-sign. They continue until the next line which starts with a different number of semicolons. **docscm** comments are associated with the S-expression immediately following them - in the example above, the comment is associated with the definition of `double`.

If the at-sign in the comment leader is followed by whitespace or newline, then the comment is associated with the following definition (if any). However, if the at-sign is immediately followed by an identifier, the rest of the line is parsed as a list of parameter S-expressions to the command named by the leading identifier.

1.2. @-commands

section.

```
;;@section <string> [<section-id>]
```

This command starts a new `<section>` in the generated (S)XML. The first parameter is the title of the new (sub)section, and should be a (double-quoted) string. The second parameter is optional - if present it should be a string or symbol to be used as the "id" attribute of the new section.

variable.

```
;;@variable <varname>
```

This command overrides the definition this comment is associated with, if any, and forces a variable-definition block to be output.

function.

```
;;@function (<function-name> [<arg> ...])
```

This command overrides the definition this comment is associated with, if any, and forces a function-definition block to be output. If there is a rest parameter in the list of *arg*s, it is output as a regular parameter followed by an ellipsis.

macro.

```
;;@macro (<macro-name> [<arg> ...])
```

Similar to `@function`, but documents a macro definition instead of a function definition.

sxml.

```
;;@sxml
```

Forces the rest of the comment to be interpreted as an SXML fragment to be literally inserted into the SXML output of **extract-doc**.

title.

```
;;@title <string>
```

Sets the title of the outermost generated SXML section to the string parameter given, and outputs the rest of the comment as initial paragraphs in the section. Should be used only as the first **docscm** command in a particular input file.

1.3. The extract-doc DSSSL stylesheet

docscm comes with a DSSSL stylesheet, `extract-doc.dsl`, which formats variable-, class- and method-description DocBook elements using Scheme external representation. The stylesheet extends the default DocBook Print or HTML stylesheet, so all the standard parameter definitions remain valid. `extract-doc.dsl` also defines some new parameters and conditional-inclusion variables.

1.3.1. Parameters

function `indent-docscm-synopsis`. This overridable function returns the amount of space to indent Scheme variable-, function-, macro- and class-definitions by. It may return a negative amount to outdent definitions.

1.3.2. Inclusion variables

a4paper. If defined, then *%paper-type%* is set to `"A4"`. (This is provided so that paper type can be easily controlled from the command-line without having to define an extension stylesheet.)

docscm-html. If defined, then `extract-doc.dsl` will behave overall as an HTML stylesheet, by including the standard DocBook HTML stylesheet.

docscm-print. If defined, then `extract-doc.dsl` will behave overall as a print stylesheet, by including the standard DocBook Print stylesheet.

1.4. Finding docscm tools

Since **docscm** may be configured and installed at any path prefix in the file system, there has to be some way of finding out where various **docscm** files are. The program **docscm-config** allows scripts and makefiles to find out where the tools they use are located.

Running **docscm-config --dslfile** will print out the full path to `extract-doc.dsl`; supplying **--makefile** causes it to print out the full path to `Makefile.docbook` (described below); and **--tooldir** prints out the path to the directory where **extract-doc** and **sxml2xml** live.

1.5. Makefiles

Included with **docscm** is a GNU Makefile fragment, `Makefile.docbook`, which provides rules for certain common Makefile productions. PDF, PostScript and HTML output can be produced from `.xml` files, which are automatically indexed. XML is produced from (the **docscm** dialect of) SXML using **sxml2xml**.

If *DOCSCM_STYLESHEET* is defined, it is used as the path to the DSSSL stylesheet to use when translating DocBook to Print or HTML formats. If *DOCSCM_PARAMS* is defined, the contents are put on the command-lines of **docbook2pdf**, **docbook2ps**, and/or **docbook2html**. This can be used to supply, for instance, `-i a4paper` to select A4 paper size.

To make use of `Makefile.docbook` in your own makefiles, GNU make's **include** command can be used along with **docscm-config**. The following example is the Makefile used to build this documentation:

```
1: TARGET=docscm
2: TARGET_INC=../extract-doc.xml ../sxml2xml.xml
3:
4: # Use A4 rather than US Letter
5: DOCSCM_PARAMS=-i a4paper
6:
7: # Set this to override the DSSSL stylesheet used by Makefile.docbook.
```

```
 8: #DOCSCM_STYLESHEET=$(CURDIR)/my-stylesheet.dsl
 9:
10: all: $(TARGET).pdf
11:
12: $(TARGET).xml: $(TARGET_INC)
13:
14: # The index is precious - expensive to build, but intermediary.
15: .SECONDARY: $(TARGET)-index.xml
16:
17: clean:
18:         rm -f $(TARGET_INC)
19:         rm -f $(patsubst %.xml,%.sxml,$(TARGET_INC))
20:         rm -f $(TARGET).xml $(TARGET).tex $(TARGET).out
21:         rm -f $(TARGET)-index.xml
22:
23: include $(shell docscm-config --makefile)
```

Notes

1. http://www.call-with-current-continuation.org
2. http://swissnet.ai.mit.edu/~jaffer/Docupage/schmooz.html
3. http://okmij.org/ftp/Scheme/SXML.html

Chapter 2. Tool documentation

This section describes each tool installed as part of **docscm** in the `docscm-config --tooldir` directory.

2.1. extract-doc: Scheme comments to SXML

extract-doc *filename...*

Extracts documentation from Scheme files and writes out SXML DocBook entries to stdout. SRFI-10 external representations of comment blocks are used as an intermediate processing step.

2.1.1. Internal datatypes and procedures

record: `(make-comment-block` *command lines code*`)`

Holds a comment-block between parsing and output.

procedure: `(->string` *x*`)`

Convert an arbitrary scheme object to a string.

procedure: `(split-paragraphs` *lines*`)`

Splits a list of strings into a list of paragraphs, representing each paragraph as a list of strings. Empty (zero-length) strings are interpreted as paragraph separators in the input list.

procedure: `(gen-body-text` *doc lines*`)`

Emits a block of comment text to the output SXML document, after splitting it into paragraphs using `split-paragraphs` and formatting it appropriately.

procedure: `(newsym)`

A version of gensym that generates (reasonably) unique symbols not only within an instance of the scheme system, but also between runs, by incorporating the current time in the symbol.

procedure: `(gen-variable-def` *doc lines varname initlist*`)`

Formats a block of documentation for a variable as DocBook markup.

procedure: `(gen-function-def` *doc lines fnname arglist*`)`

Formats a block of documentation for a function as DocBook markup.

procedure: `(gen-macro-def` *doc lines macroname arglist*`)`

Formats a block of documentation for a macro as DocBook markup.

procedure: `(gen-sxml` *`doc lines`*`)`

Appends raw SXML text directly to the output SXML document. Used by the **@sxml docscm** command.

procedure: `(dig-for-lambda-value` *`body`*`)`

Recurses into an expression, trying (via a fairly sloppy algorithm) to figure out if it returns a procedure or not.

procedure: `(gen-block` *`cblk doc`*`)`

Processes a single **docscm** comment block.

procedure: `(gen-doc` *`filename items`*`)`

Returns an SXML fragment of documentation for file `filename`, generated from the comment-blocks in `items`.

procedure: `(extract-doc` *`filename`*`)`

Extracts documentation from the file passed in as an argument. `filename` is the file to parse. The function returns a quasiquoted SXML DocBook expression.

2.2. sxml2xml: pseudo-SXML to XML text

sxml2xml *`filename`*...

Reads a file, an S-expression at a time, evaluating each expression. Each expression is expected to yield a pseudo-SXML fragment, which is then converted to plain XML text, and printed on stdout.

2.2.1. Internal datatypes and procedures

These subroutines are also available to code loaded as part of the SXML-to-XML conversion process.

procedure: `(include-sxml` *`filenames ...`*`)`

Reads a file, an S-expression at a time, evaluating each expression, and collecting the results into a list of SXML fragments.

Index