Scriblib: Extra Scribble Libraries

Version 6.5

April 22, 2016

Contents

1 Examples Using the GUI Toolbox **3**

2 Figures **5**
2.1 Configuring Output 7

3 Bibliographies **8**

4 BibTeX Bibliographies **13**

5 Footnotes **14**

6 Conditional Content **15**

1 Examples Using the GUI Toolbox

```
(require scriblib/gui-eval)
```
package: scribble-lib

The scriblib/gui-eval library support example evaluations that use racket/gui facilities (as opposed to just racket/draw) to generate text and image results.

The trick is that racket/gui is not generally available when rendering documentation, because it requires a GUI context. Text and image output is rendered to an image file when the MREVAL environment variable is set, so run the enclosing document once with the environment varibale to generate the images. Future runs (with the environment variable unset) use the generated image.

```
(gui-interaction datum ...)
(gui-interaction
 #:eval+opts the-eval get-predicate? get-render
             get-get-width get-get-height
 datum ...)
(gui-interaction-eval datum ...)
(gui-interaction-eval
 #:eval+opts the-eval get-predicate? get-render
             get-get-width get-get-height
 datum ...)
(gui-interaction-eval-show datum ...)
(gui-interaction-eval-show
 #:eval+opts the-eval get-predicate? get-render
             get-get-width get-get-height
 datum ...)
(gui-racketblock+eval datum ...)
(gui-racketblock+eval
 #:eval+opts the-eval get-predicate? get-render
             get-get-width get-get-height
 datum ...)
(gui-racketmod+eval datum ...)
(gui-racketmod+eval
 #:eval+opts the-eval get-predicate? get-render
             get-get-width get-get-height
 datum ...)
(gui-def+int datum ...)
(gui-def+int
 #:eval+opts the-eval get-predicate? get-render
             get-get-width get-get-height
 datum ...)
```

```
(gui-defs+int datum ...)
(gui-defs+int
 #:eval+opts the-eval get-predicate? get-render
             get-get-width get-get-height
 datum ...)
```

The first option of each of the above is like `interaction`, etc., but actually evaluating the forms only when the `MREVAL` environment variable is set, and then in an evaluator that is initialized with `racket/gui/base` and `slideshow`.

The second option of each allows you to specify your own evaluator via the *the-eval* argument and then to specify four thunks that return functions for finding and rendering graphical objects:

- *get-predicate?* : `(-> (-> any/c boolean?))` Determines if a value is a graphical object (and thus handled by the other operations)
- *get-render* : `(-> (-> any/c (is-a?/c dc<%>) number? number? void?))` Draws a graphical object (only called if the predicate returned `#t`; the first argument will be the value for which the predicate holds).
- *get-get-width* : `(-> (-> any/c number?))` Gets the width of a graphical object (only called if the predicate returned `#t`; the first argument will be the value for which the predicate holds).
- *get-get-height* : `(-> (-> any/c number?))` Gets the height of a graphical object (only called if the predicate returned `#t`; the first argument will be the value for which the predicate holds).

2 Figures

```
(require scriblib/figure)
```
package: scribble-lib

```
(figure  tag
         caption
         p ...
        [#:style style
         #:continue? continue?]) → block?
  tag : string?
  caption : content?
  p : pre-flow?
  style : style? = center-figure-style
  continue? : any/c = #f
(figure*  tag
          caption
          p ...
         [#:style style
          #:continue? continue?]) → block?
  tag : string?
  caption : content?
  p : pre-flow?
  style : style? = center-figure-style
  continue? : any/c = #f
(figure**  tag
           caption
           p ...
          [#:style style
           #:continue? continue?]) → block?
  tag : string?
  caption : content?
  p : pre-flow?
  style : style? = center-figure-style
  continue? : any/c = #f
(figure-here  tag
              caption
              pre-flow ...
             [#:style style
              #:continue? continue?]) → block?
  tag : string?
  caption : content?
  pre-flow : pre-flow?
  style : style? = center-figure-style
  continue? : any/c = #f
```

Creates a figure. The given *tag* is for use with figure-ref or Figure-ref. The *caption* is an element. The *pre-flow* is decoded as a flow.

For HTML output, the figure and figure* functions are the same, while figure** allows the content to be wider than the document body. For two-column Latex output, figure* and figure** generate a figure that spans columns.

For Latex output, figure-here generates a figure to be included at the position in the output text where the figure-here occurs in the source text. For HTML output, all figure variants place the figure where the use appears in the source text.

By default, *style* is set so that the content of the figure is centered. Use left-figure-style, center-figure-style, or right-figure-style to specify the alignment.

If *continue?* is a true value, then the figure counter is not incremented.

```
left-figure-style : style?
center-figure-style : style?
right-figure-style : style?
left : style?
```

Implements figure alignments.

The left binding is a synonym for left-figure-style, provided for backward compatibility.

```
(figure-ref tag ...+) → element?
  tag : string?
```

Generates a reference to one or more figures, using a lowercase word “figure”.

```
(Figure-ref tag ...+) → element?
  tag : string?
```

Generates a reference to one or more figures, capitalizing the word “Figure”.

```
(Figure-target tag [#:continue? continue?]) → element?
  tag : string?
  continue? : any/c = #f
```

Generates a new figure label. This function is normally not used directly, since it is used by figure.

```
(suppress-floats) → element?
```

Produces an empty element that renders in Latex as \suppressfloats, which discourages the placement of figures in the column or page of the surrounding text.

2.1 Configuring Output

Output uses the following style names, which can be adjusted in an overriding `".css"` or `".tex"` specification:

- `"Figure"`, `"FigureMulti"`, `"FigureMultiWide"`, or `"HereFigure"` — used for the outer of three `nested-flow`s for a figure, depending on whether `figure`, `figure*`, `figure**`, or `figure-here` is used to generate the figure.
- `"Leftfigure"`, `"Centerfigure"`, or `"Rightfigure"` — used for the middle of three `nested-flow`s for a figure, depending on the specified style.
- `"FigureInside"` — used for the inner of three `nested-flow`s for a figure.
- `"Legend"` — Wraps the caption for a figure.
- `"LegendContinued"` — Wraps the caption for a figure that does not increment the figure counter.
- `"FigureTarget"` — Wraps the label anchor and text within a figure's caption. For Latex output, the corresponding command is given a second argument, which is just the generated label (used with `\label` in the command's first argument).
- `"FigureRef"` — Wraps a reference to a figure. For Latex output, the corresponding command is given a second argument, which is just the target label.

3 Bibliographies

```
(require scriblib/autobib)
```
package: scribble-lib

This library provides support for bibliography management in a Scribble document. The `define-cite` form is used to bind procedures that create in-line citations and generate the bibilography in the document.

Individual bibliography entries are created with the `make-bib` function. See below for an example.

```
#lang scribble/base

@(require scriblib/autobib)

@(define-cite ~cite citet generate-bibliography)

@(define plt-tr1
   (make-bib
    #:title    "Reference: Racket"
    #:author   (authors "Matthew Flatt" "PLT")
    #:date     "2010"
    #:location (techrpt-location #:institution "PLT Inc."
                                 #:number "PLT-TR-2010-1")
    #:url      "http://racket-lang.org/tr1/"))

Racket is fun@~cite[plt-tr1].

@(generate-bibliography)
```

For citations that reference a page number or section, the `in-bib` function can be used. For example, the following snippet:

```
Racket has a contract library.@~cite[(in-bib plt-tr1 ", §8")]
```

includes a citation to section 8 of the Racket reference.

```
(define-cite ~cite-id citet-id generate-bibliography-id
             option ...)

option = #:style style-expr
       | #:disambiguate disambiguator-expr
       | #:spaces spaces-expr
       | #:render-date-bib render-date-expr
       | #:render-date-cite render-date-expr
       | #:date<? date-compare-expr
       | #:date=? date-compare-expr
```

```
style-expr : (or/c author+date-style number-style)
spaces-expr : number
disambiguator-expr : (or/c #f (-> exact-nonnegative-integer? element?))
render-date-expr : (or/c #f (-> date? element?))
date-compare-expr : (or/c #f (-> date? date? boolean?))
```

Binds *~cite-id*, *citet-id*, and *generate-bibliography-id*, which share state to accumulate and render citations.

The function bound to *~cite-id* produces a citation referring to one or more bibliography entries with a preceding non-breaking space, by default sorting the entries to match the bibliography order. It has the contract

```
(->* (bib?) (#:sort? any/c) #:rest (listof bib?) element?)
```

The function bound to *citet-id* generates an element suitable for use as a noun—referring to a document or its author—for one or more bibliography entries which have the same authors. It has the contract

```
(->* (bib?) () #:rest (listof bib?) element?)
```

The function bound to *generate-bibliography-id* generates the section for the bibliography. It has the contract

```
(->* () (#:tag string? #:sec-title string?) part?)
```

The default value for the `#:tag` argument is `"doc-bibliography"` and for `#:sec-title` is `"Bibliography"`.

The optional *spaces-expr* determines the number of blank lines that appear between citations. The default number of lines is 1.

The optional *style-expr* determines the way that citations and the bibliography are rendered. Currently, two built-in style are provided, and `author+date-style` is the default.

Programmer-defined styles may be supported in the future.

For `author+date-style`, if two citations' references would render the same (as judged by equal authors and dates that are considered the same) but are different, the optionally provided function from *disambiguator-expr* is used to add an extra element after the date; the default disambiguator adds `a`, `b`, etc. until `z`, and anything more ambiguous raises an exception. Date comparison is controlled by *date-compare-expr*s. Dates in citations and dates in the bibliography may be rendered differently, as specified by the optionally given *render-date-expr* functions.

```
author+date-style : any/c
number-style : any/c
```

Styles for use with `define-cite`.

```
(bib? v) → boolean?
  v : any/c
```

Returns `#t` if *v* is a value produced by `make-bib` or `in-bib`, `#f` otherwise.

```
(make-bib  #:title title
           [#:author author
            #:is-book? is-book?
            #:location location
            #:date date
            #:url url
            #:note note])        → bib?
  title : any/c
  author : any/c = #f
  is-book? : any/c = #f
  location : any/c = #f
  date : (or/c #f date? exact-nonnegative-integer? string?) = #f
  url : string? = #f
  note : any/c = #f
```

Produces a value that represents a document to cite. Except for *is-book?* and *url*, the arguments are used as content, except that `#f` means that the information is not supplied. Functions like `proceedings-location`, `author-name`, and `authors` help produce elements in a standard format.

Dates are internally represented as *date* values, so a *date* may be given, or a number or string that represent the year.

An element produced by a function like `author-name` tracks first, last names, and name suffixes separately, so that names can be ordered and rendered correctly. When a string is provided as an author name, the last non-empty sequence of alphabetic characters or `-` after a space is treated as the author name, and the rest is treated as the first name.

```
(in-bib orig where) → bib?
  orig : bib?
  where : string?
```

Extends a bib value so that the rendered citation is suffixed with *where*, which might be a page or chapter number.

```
(proceedings-location  location
                       [#:pages pages
                        #:series series
                        #:volume volume]) → element?
  location : any/c
  pages : (or (list/c any/c any/c) #f) = #f
  series : any/c = #f
  volume : any/c = #f
```

Combines elements to generate an element that is suitable for describing a paper's location within a conference or workshop proceedings.

```
(journal-location  title
                   [#:pages pages
                    #:number number
                    #:volume volume]) → element?
  title : any/c
  pages : (or (list/c any/c any/c) #f) = #f
  number : any/c = #f
  volume : any/c = #f
```

Combines elements to generate an element that is suitable for describing a paper's location within a journal.

```
(book-location [#:edition edition
                #:publisher publisher]) → element?
  edition : any/c = #f
  publisher : any/c = #f
```

Combines elements to generate an element that is suitable for describing a book's location.

```
(techrpt-location [#:institution institution]
                  #:number number)           → element?
  institution : edition = any/c
  number : any/c
```

Combines elements to generate an element that is suitable for describing a technical report's location.

```
(dissertation-location [#:institution institution
                        #:degree degree])          → element?
  institution : edition = any/c
  degree : any/c = "PhD"
```

Combines elements to generate an element that is suitable for describing a dissertation.

```
(author-name first last [#:suffix suffix]) → element?
  first : any/c
  last : any/c
  suffix : any/c = #f
```

Combines elements to generate an element that is suitable for describing an author's name, especially where the last name is not merely a sequence of ASCII alphabet letters or where the name has a suffix (such as "Jr.").

```
(authors name names ...) → element?
  name : content?
  names : content?
```

Combines multiple author elements into one, so that it is rendered and alphabetized appropriately. Any of *name* or *names* that are strings are parsed in the same way as by make-bib.

```
(org-author-name name) → element?
  name : any/c
```

Converts an element for an organization name to one suitable for use as a bib-value author.

```
(other-authors) → element?
```

Generates an element that is suitable for use as a "others" author. When combined with another author element via authors, the one created by other-authors renders as "et al."

```
(editor name) → element?
  name : name/c
```

Takes an author-name element and create one that represents the editor of a collection. If a *name* is a string, it is parsed in the same way as by make-bib.

```
(abbreviate-given-names) → any/c
(abbreviate-given-names abbreviate?) → void?
  abbreviate? : any/c
```

Shortens given names in calls to author and make-bib to just the first initial when the parameter value is not #f. Otherwise, does not change the author names.

Defaults to #f.

Added in version 1.5 of package scribble-lib.

4 BibTeX Bibliographies

```
(require scriblib/bibtex)
```
package: scribble-lib

```
(define-bibtex-cite bib-pth ~cite-id citet-id generate-bibliography-id
  option ...)
```

Expands into:

```
(begin
  (define-cite autobib-cite autobib-citet generate-bibliography-id
    option ...)
  (define-bibtex-cite* bib-pth
    autobib-cite autobib-citet
    ~cite-id citet-id))
```

```
(define-bibtex-cite* bib-pth autobib-cite autobib-citet
                     ~cite-id citet-id)
```

Parses *bib-pth* as a BibTeX database, and augments *autobib-cite* and *autobib-citet* into ~*cite-id* and *citet-id* functions so that rather than accepting bib? structures, they accept citation key strings.

Each string is broken along spaces into citations keys that are looked up in the BibTeX database and turned into bib? structures.

The only BibTeX entries that are supported are: misc, book, article, inproceedings, webpage, masterthesis, and techreport.

```
(struct bibdb (raw bibs))
  raw : (hash/c string? (hash/c string? string?))
  bibs : (hash/c string? bib?)
```

Represents a BibTeX database. The *raw* hash table maps the labels in the file to hash tables of the attributes and their values. The *bibs* hash table maps the same labels to Scribble data-structures representing the same information.

```
(path->bibdb path) → bibdb?
  path : path-string?
```

Parses a path into a BibTeX database.

```
(bibtex-parse ip) → bibdb?
  ip : input-port?
```

Parses an input port into a BibTeX database.

5 Footnotes

```
(require scriblib/footnote)
```
package: scribble-lib

```
(note pre-content ...) → element?
  pre-content : pre-content?
```

Creates a margin note for HTML and a footnote for Latex/PDF output.

```
(define-footnote footnote-id footnote-part-id)
```

Binds *footnote-id* to a form like note that generates a footnote in HTML output as well as Latex/PDF output. To trigger the HTML output of the footnotes that are registered through *footnote-id*, the function bound to *footnote-part-id* must be called at a position that corresponds the bottom of the HTML page. (The generated section will not show a title or appear in a table of contents; it will look like a footnote area.)

Beware that any content passed to *footnote-id* will occur twice in at least an intermediate form of the document, and perhaps also in the rendered form of the document. Consequently, the content passed to *footnote-id* should not bind link targets or include other one-time declarations.

6 Conditional Content

```
(require scriblib/render-cond)
```
package: scribble-lib

As much as possible, Scribble documents should be independent of the target format for rendering the document. To customize generated output, use styles plus "back end" configurations for each target format (see §6.11 "Extending and Configuring Scribble Output" in *Scribble: The Racket Documentation Tool*).

As a last resort, the `cond-element` and `cond-block` forms support varying the document content depending on the target format. More precisely, they generate parts of a document where content is delayed until the traverse pass of document rendering. Format detection relies on the `'scribble:current-render-mode` registration that is accessible through a `traverse-element` or `traverse-block`.

The syntax of `cond-element` and `cond-block` is based on SRFI-0.

```
(cond-element [feature-requirement body ...+])
(cond-element [feature-requirement body ...+] [else body ...+])

feature-requirement = identifier
                    | (not feature-requirement)
                    | (and feature-requirement ...)
                    | (or feature-requirement ...)
```

Generates a `traverse-element` whose replacement content is produced by the *body* of one of the first matching `cond-element` clause.

A *feature-requirement* can be any identifier; a useful identifier is one whose symbol form can appear in a `'scribble:current-render-mode` list. The identifier matches when its symbol form is in the `'scribble:current-render-mode` list. Typically, the identifier is `html`, `latex`, or `text` to indicate the corresponding rendering target.

A `(not` *feature-requirement*`)` test matches when *feature-requirement* does not match, and so on. An `else` clause always matches. If no `else` clause is present and no clause matches, then the `exn:fail:contract` exception is raised. Similarly, if the result of the selected *body* is not content according to `content?`, then the `exn:fail:contract` exception is raised.

```
(cond-block [feature-requirement body ...+])
(cond-block [feature-requirement body ...+] [else body ...+])
```

Like `cond-element`, but generates a `traverse-block` where the selected *body* must produce a block according to `block?`.