Scriblib: Extra Scribble Libraries

Version 9.4.0.5

November 10, 2026

Contents

1 Examples Using the GUI Toolbox **3**

2 Figures **5**

2.1 Configuring Output 7

3 Bibliographies **9**

4 BibTeX Bibliographies **19**

5 Footnotes **23**

6 Conditional Content **25**

7 Book-Style Indexing **27**

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

```
(figure-here  tag
              caption
              pre-flow ...
             [#:style style
              #:label-sep label-sep
              #:label-style label-style
              #:continue? continue?])    → block?
  tag : string?
  caption : content?
  pre-flow : pre-flow?
  style : style? = center-figure-style
  label-sep : pre-content? = ": "
  label-style : element-style? = #f
  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.

The *label-sep* and *label-style* arguments adjust the way that the caption's label is shown. By default, the label is the word "Figure" followed by a space, the figure number, ":", and a space, but *label-sep* can specify an alternative to the ":" and ending space. The composed label is given the style specified by *label-style*.

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

Changed in version 1.24 of package `scribble-lib`: Added the `#:label-sep` and `#:label-style` arguments.

```
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
             ...+
            [#:and-word and-word
             #:link-render-style link-style]) → element?
  tag : string?
  and-word : string? = "and"
  link-style : (or/c link-render-style? #f) = #f
```

Generates a reference to one or more figures, using a lowercase word “figure”. When two tags are given, the *and-word* is used to connect the two.

If *link-style* or `(current-link-render-style)` at the time of rendering indicates the `'number` style mode, then the word “figure” itself is not hyperlinked. Otherwise, the word `figure` is hyperlinked together with the referenced figure’s number.

Changed in version 1.26 of package `scribble-lib`: Added the `#:link-render-style` argument.

```
(Figure-ref tag
            ...+
            #:link-render-style link-style) → element?
  tag : string?
  link-style : (or/c link-render-style? #f)
```

Like `figure-ref`, but capitalizes the word “Figure”.

Changed in version 1.26 of package `scribble-lib`: Added the `#:link-render-style` argument.

```
(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 bibliography 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.

Changed in version 1.61 of package `scribble-lib`: Added fields and location types for better bibtex support. Changed in version 1.68: Improved bibliography layout, added support for multi-paragraph notes and extended support for structured content in bibliography fields.

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

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

  style-expr : (or/c number-style author+date-style author+date-square-bracket-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*, *generate-bibliography-id*, (optionally) *cite-author-id*, and (optionally) *cite-year-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, with a title as per the `#:sec-title` argument (defaults to `"Bibliography"`) and a tag as per the `#:tag` argument (defaults to `"doc-bibliography"`). If the `#:sec-title` argument is `#f` instead a string, only the content of the bibliography is created, as a table, and not the enclosing section as a part.

```
(->* () (#:tag string? #:sec-title (or/c #f string?))
     (or/c part? block?))
```

If provided, the function bound to *cite-author-id* generates an element containing the authors of a paper.

```
(->* (bib?) element?)
```

If provided, the function bound to *cite-year-id* generates an element containing the year the paper was published in, or possibly multiple years if multiple papers are provided.

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

The functions bound to *cite-author-id* and *cite-year-id* make it possible to create possessive textual citations.

```
@citeauthor[scribble-cite]'s (@citeyear[scribble-cite]) autobib
library is pretty nifty.
```

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 styles 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.

Changed in version 1.22 of package scribble-lib: Add optional ids for author-name and author-year

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

Styles for use with define-cite.

With number-style, bibliography entries use hanging indentation in HTML and LaTeX output, with citation numbers aligned in a separate label column. Text output instead separates each number from its entry with a non-breaking space.

The author+date-square-bracket-style definition is the same as author+date-style, except that references to citations are enclosed in [] instead of ().

In LaTeX output, Scribble tries to keep short bibliography entries together, reserving at least five lines before starting an entry. This approximates the previous behavior, which prevented page breaks within individual entries altogether. Longer entries may now span pages.

The \AutobibNeedlines counter controls the minimum number of lines, defaulting to 5. Set it to 0 to disable this constraint. The optional needspace package is required for the

constraint to take effect; this behavior is disabled if the package is unavailable, as if the counter were 0.

The `\AutobibEntrySetup` command, empty by default, allows additional LaTeX settings to be applied locally to each bibliography entry.

To require four lines before each entry and relax line breaking for long annotations, configure these settings using `\AtBeginDocument` from e.g. a `tex-addition` that you add to your document's style:

```
(tex-addition
  (bytes-append
    #"\\AtBeginDocument{%\n"
    #"  \\AutobibNeedlines=4\\relax\n"
    #"  \\renewcommand{\\AutobibEntrySetup}{%\n"
    #"    \\emergencystretch=2em\n"
    #"    \\tolerance=1000}}%\n"))
```

```
(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
            #:accessed accessed
            #:doi doi
            #: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 : (or/c #f string?) = #f
  accessed : any/c = #f
  doi : (or/c #f string?) = #f
  note : any/c = #f
```

Produces a value that represents a document to cite. Except for *is-book?*, *url*, and *doi*, 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.

The #:note argument may contain multiple paragraphs, separated by blank lines. The first paragraph follows the bibliographic information; subsequent paragraphs remain within the same bibliography entry.

When both #:doi and #:url are supplied, the DOI takes precedence. A period is inserted after a DOI when followed by a non-empty note. No period is appended directly to a URL.

#:accessed gives the date a #:url (in CSL terms, the date it was accessed), and is only used when #:url is displayed, i.e. when no #:doi is supplied. It is rendered right after the URL, separated from it by a space, as (accessed ...). A period is inserted after it when followed by a non-empty note; a naked URL (no #:accessed) never gets that period.

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.

Changed in version 1.49 of package scribble-lib: Added #:doi.
Changed in version 1.68: Added #:accessed, which replaces the accessed-date support formerly provided by the now-removed webpage-location function: the accessed date is now attached directly to the bib entry instead of being embedded in its #:location, so it renders next to the URL rather than before the date.

```
(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.

Each of the following *-location functions combines its arguments into a single element?. proceedings-location, book-location, booklet-location, misc-location, and manual-location may legitimately be called by scriblib/bibtex with no useful information at all (e.g. a BibTeX entry that supplies none of the corresponding optional fields): each returns #f when every one of its arguments is either omitted (#f) or supplied as empty content. The rest each have at least one genuinely required argument, so they always have something to report and never return #f.

For a handful of fields across these functions – an edition, an editor, a location, a page range, a series number – #f alone isn't enough to tell "omitted" from "supplied but empty," because the field gets wrapped in surrounding text (e.g. an editor's name becomes "NAME (Ed.)"; an edition becomes "EDITION edition"). Omitting the field (#f, the default) still quietly contributes nothing, but explicitly supplying empty or otherwise trivial content for one of these particular fields (e.g. "") raises a contract violation instead of silently producing an orphaned fragment like "(Ed.)" with no name attached. Each function's entry below says

which of its fields this applies to.

```
(proceedings-location [#:editor editor_]
                      location
                     [#:series series
                      #:volume volume
                      #:number number
                      #:pages pages
                      #:organization organization]
                      #:publisher publisher
                      #:address address)
 → (or/c element? #f)
  editor_ : any/c = #f
  location : any/c
  series : any/c = #f
  volume : any/c = #f
  number : any/c = #f
  pages : (or (list/c any/c any/c) #f) = #f
  organization : any/c = #f
  publisher : #f
  address : #f
```

Combines the supplied information to produce content suitable for describing a paper's location within a conference or workshop proceedings. Returns `#f` when every argument is omitted or empty, except that explicitly-supplied-but-trivial content raises for *location*, `#:editor`, `#:number`, and `#:pages` specifically, since each of those is wrapped in surrounding text (see above).

Changed in version 1.61 of package `scribble-lib`: Added fields for bibtex support: editor number organization publisher address.
Changed in version 1.68: Added `#f` as a possible result.

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

Combines the supplied information to produce content suitable for describing a paper's location within a journal.

```
(book-location [#:edition edition
                #:chapter chapter
                #:editor editor
                #:series series
                #:volume volume
                #:number number
                #:pages pages
                #:publisher publisher
                #:address address])    → (or/c element? #f)
  edition : any/c = #f
  chapter : any/c = #f
  editor : any/c = #f
  series : any/c = #f
  volume : any/c = #f
  number : any/c = #f
  pages : any/c = #f
  publisher : any/c = #f
  address : any/c = #f
```

Combines the supplied information to produce content suitable for describing a book's location. Returns `#f` when every argument is omitted or empty, except that explicitly-supplied-but-trivial content raises for `#:edition`, `#:editor`, `#:number`, and `#:pages` specifically (see above).

A numeric *chapter*, supplied as a number or a string of decimal digits, is prefixed with "chapter". Other chapter content is used unchanged.

Changed in version 1.61 of package `scribble-lib`: Added fields for bibtex support: editor chapter series volume number pages address. Made all arguments optional.
Changed in version 1.68: Added `#f` as a possible result.

```
(booklet-location [#:howpublished howpublished
                   #:address address])
 → (or/c element? #f)
  howpublished : any/c = #f
  address : any/c = #f
```

Combines the supplied information to produce content suitable for describing a booklet's location. Neither argument is wrapped in surrounding text, so returns `#f` when both are omitted or empty, with no exceptions.

Added in version 1.61 of package `scribble-lib`.
Changed in version 1.68: Added `#f` as a possible result.

```
(misc-location [#:howpublished howpublished])
 → (or/c element? #f)
  howpublished : any/c = #f
```

Combines the supplied information to produce content suitable for describing a bibtex misc entry's location. Its one argument isn't wrapped in surrounding text, so returns #f when it's omitted or empty, with no exception.

Added in version 1.61 of package scribble-lib.
Changed in version 1.68: Added #f as a possible result.

```
(manual-location [#:organization organization
                  #:edition edition])
 → (or/c element? #f)
  organization : any/c = #f
  edition : any/c = #f
```

Combines the supplied information to produce content suitable for describing a manual's location. Returns #f when both arguments are omitted or empty, except that explicitly-supplied-but- trivial content raises for #:edition specifically (see above).

Added in version 1.61 of package scribble-lib.
Changed in version 1.68: Added #f as a possible result.

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

Combines the supplied information to produce content suitable for describing a technical report's location.

Changed in version 1.61 of package scribble-lib: Added fields for bibtex support: type address.

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

Combines the supplied information to produce content suitable for describing a dissertation.

Changed in version 1.61 of package scribble-lib: Added fields for bibtex support: type address.

```
(book-chapter-location  title
                       [#:edition edition
                        #:chapter chapter
                        #:editor editor
                        #:series series
                        #:volume volume
                        #:number number
                        #:pages pages
                        #:publisher publisher
                        #:address address])    → element?
  title : any/c
  edition : any/c = #f
  chapter : any/c = #f
  editor : any/c = #f
  series : any/c = #f
  volume : any/c = #f
  number : any/c = #f
  pages : any/c = #f
  publisher : any/c = #f
  address : any/c = #f
```

Combines the supplied information to produce content suitable for describing a paper's location within a chapter or part of a book or collection of books.

The *chapter* argument is formatted as by book-location.

Changed in version 1.61 of package scribble-lib: Added fields for bibtex support: editor chapter number address.

```
(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 : (or/c element? string?)
```

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 : (or/c element? string?)
```

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`.

Raises a contract violation if *name* is empty or otherwise trivial content (e.g. `""`), rather than silently producing a name-less “(Ed.)” credit.

Changed in version 1.68 of package `scribble-lib`: Raises on empty or trivial content instead of silently producing a bogus, name-less credit.

```
(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`.

```
(url-rendering) → (-> string? any)
(url-rendering rendering-function) → void?
  rendering-function : (-> string? any)
```

Accepts a URL as a string and renders it for use in a bibliography entry.

Defaults to `(λ (url) (link url (make-element 'url (list url))))`.

Added in version 1.39 of package `scribble-lib`.

4 BibTeX Bibliographies

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

This library supports parsing BibTeX `.bib` files.

We support the 14 standard BibTeX entry types documented in Oren Patashnik's *BIBTEXing* (February 8, 1988), §3.1 "Entry Types" — the definitive reference for classic BibTeX, which updates Appendix B.2 of Leslie Lamport's *LaTeX: A Document Preparation System* (1986): `article`, `book`, `booklet`, `conference`, `inbook`, `incollection`, `inproceedings`, `manual`, `mastersthesis`, `misc`, `phdthesis`, `proceedings`, `techreport`, and `unpublished`.

Human-readable BibTeX fields are converted from a subset of LaTeX syntax into Scribble content. This includes grouping braces, common accent and special-letter commands, `\emph`, `\texttt`, `\textit`, `\textbf`, `\textsc` and `\url`.

Grouping braces are preserved internally where relevant to BibTeX name parsing. Unknown LaTeX commands and their attached arguments are retained rather than discarded. Inline and display mathematics are preserved in LaTeX syntax; other backends do not translate them into native mathematical expressions.

Blank lines in `note` fields separate paragraphs.

The `url` and `doi` fields are interpreted as scalar strings rather than general LaTeX content. Enclosing brace groups are removed and conventional LaTeX escapes for special URL characters, such as `\_`, `\%` and `\&`, are unescaped. Other LaTeX commands are not interpreted in these fields.

We support all the required and optional fields documented in *BIBTEXing*, with the following known limitations so far:

- We fail to process `month`.
- We only support `pages` fields that have decimal numbers separated by one or more dashes.
- We fail to process the `type` optional field of `incollection`.
- We do not at this time support the non-standard but often seen fields `isbn`, `issn`, nor any other non-standard field except those described below.

For each standard entry type, the fields that *BIBTEXing* §3.1 marks required are enforced at parse time: a missing one raises an error, rather than being silently treated as absent the way every other field is. This is:

- `article`: `author`, `title`, `journal`, `year`.
- `book`: `author` or `editor`, `title`, `publisher`, `year`.
- `booklet`: `title`.
- `conference`: same as `inproceedings`.
- `inbook`: `author` or `editor`, `title`, `chapter` and/or `pages`, `publisher`, `year`.
- `incollection`: `author`, `title`, `booktitle`, `publisher`, `year`.
- `inproceedings`: `author`, `title`, `booktitle`, `year`.
- `manual`: `title`.
- `mastersthesis`: `author`, `title`, `school`, `year`.
- `misc`: none.
- `phdthesis`: `author`, `title`, `school`, `year`.
- `proceedings`: `title`, `year`.
- `techreport`: `author`, `title`, `institution`, `year`.
- `unpublished`: `author`, `title`, `note` (no `year`).

An `author`-or-`editor` or `chapter`-and/or-`pages` requirement above is enforced as: an error unless at least one of the two fields is present.

Note that this is stricter than classic BibTeX itself: per *BIBTEXing*, "required" is a property of the standard bibliography styles, not a constraint on the `.bib` file format itself — a real BibTeX run only warns about a missing required field and still processes the entry, however poorly formatted the result. We chose to make it a hard error here instead, on the theory that a citation silently missing its journal or its year is worse than one that fails to build at all.

In addition to the 14 standard entries, we support the often seen `online` and `webpage` entry types, for which we support the fields `url`, `title`, `author`. Additionally `online` has field `urldate` for the day the site was visited, whereas `webpage` instead has field `lastchecked`. Neither is among *BIBTEXing*'s standard entry types, so there is no spec to draw a required-fields list from; but by our own choice, `title` and `url` are required for both (a webpage citation with neither isn't a citation), while `author` stays optional, since most web pages don't have a clean byline.

Also, for every entry type, we support the extra fields `note`, `url`, `doi` — all optional, except that `url` is required (see above) for `online` and `webpage`, the two entry types it's actually about. But mind that the `doi` field currently overrides the `url` in `scriblib/autobib`.

We do support the `@string` feature defined in the format of BibTeX.

Changed in version 1.61 of package `scribble-lib`: Support all standard entry types plus `online` and `webpage`, all fields but `month` (or `type` for `incollection`), and support `note`, `url`, `doi` on all entry types.

Changed in version 1.68 of package `scribble-lib`: Added structured LaTeX content parsing, improved author-name handling, URL and DOI unescaping, and support for multi-paragraph notes. Started enforcing, as hard parse-time errors, the fields *BIBTEXing* marks required for each standard entry type (see above), including its author-or-editor and chapter-and/or-pages disjunctions; also require `title` and `url` on `online` and `webpage` entries, by our own choice rather than any spec. Fixed a latent bug where an `editor`-only `book` or `inbook` entry (valid, since author is not required when editor is given) would silently render a bogus `"#f"` in place of the missing author's name.

```
(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.

```
(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.

To produce a numbered note for HTML (which can useful if a CSS specialization changes how "margin notes" are rendered), use `define-footnote` with `#:margin`, instead.

The element generated by `note` uses the style `"NoteBox"` wrapped around an element using the style `"NoteContent"`. CSS and Latex macros configure rendering for HTML and Latex/PDF to create a margin note or footnote, but those CSS classes or Latex macros can be redefined or overridden to produce a different result.

```
(define-footnote footnote-id footnote-part-id)
(define-footnote footnote-id #:margin)
```

Binds *footnote-id* to a form like `note` that registers a footnote. For Latex/PDF output, this is the same result as using `note`.

If *footnote-part-id* is provided, it is bound to a function that generates a section to display the registered footnotes in HTML. The section generated by *footnote-part-id* will not show a title or appear in a table of contents; it will look like a footnote area. If `#:margin` is supplied instead, then each footnote is rendered as an immediate margin note, and no separate footnote section is needed.

Using *footnote-id* within the argument to a *footnote-id* will not always work, but it can work in `#:margin` mode for HTML output.

Example:

```
#lang scribble/manual
@require[scriblib/footnote]

@define-footnote[my-note make-my-note]

@title{Months of the Year}

@section{January}
January has 31 days.

@section{February}
February has 28 days in common years.@my-note{In leap years,
```

```
February has 29 days.}

@make-my-note[]

@section{March}
March has 30 days.
```

The elements and parts generated by *footnote-id* and *footnote-part-id* use the following style names, which can be redefined as CSS classes or Latex macros to adjust the result:

- "Footnote": Style for the result of *footnote-id*, which is mapped to `\footnote` for Latex.
- "FootnoteRef": Wrapped around the reference to a footnote that is rendered by *footnote-part-id* or as a margin note. For Latex, this name is mapped to a macro that returns nothing, leaving the reference managment to `\footnote`.
- "FootnoteRefNumber": Wrapped around the footnote number as a reference (inside of "FootnoteRef"), so that the default superscript style for a footnote reference can be changed for HTML output (or potentially for Latex, if "Footnote" is not mapped to `\footnote`).
- "FootnoteTarget": Wrapped around the footnote that is rendered by *footnote-part-id* or as a margin note. For Latex, this name is mapped to a macro that returns nothing, leaving the reference managment to `\footnote`.
- "FootnoteTargetNumber": Wrapped around the footnote number rendered by *footnote-part-id* or as a margin note, so that the default superscript style within a footnote can be changed for HTML (or potentially for Latex, if "Footnote" is not mapped to `\footnote`).
- "FootnoteContent": For Latex, wrapped around the content of a footnote as rendered by *footnote-part-id*.
- "FootnoteMarginContent": Wrapped around the content of a footnote as rendered as a margin note. By default, CSS styling for this name floats the note into the right margin.
- "FootnoteBlock": Wrapped around the output of *footnote-part-id*.
- "FootnoteBlockContent": Also wrapped around the output of *footnote-part-id*, inside the "FootnoteBlock" wrapper.

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`, `cond-block`, and `cond-part` 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`, `cond-block`, and `cond-part` 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?`.

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

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

Added in version 1.67 of package `scribble-lib`.

7 Book-Style Indexing

```
(require scriblib/book-index)
```
package: scribble-lib

Provides a list of style properties to attach to a Scribble document that contains an index part, making the index more suitable for a traditional rendering on paper. The style properties cause index entries to be merged when they have the same content, with (potentially) multiple page numbers attached to the merged entry.

```
book-index-style-properties : list?
```

Combine these style properties with others for the style of a part (typically specified in `title`) for a document that contains an index. The style properties enable index merging and select an implementation based on the `cleveref` Latex package.

Example:

```
#lang scribble/base
@(require scriblib/book-index
          (only-in scribble/core make-style))

@title[#:style (make-style #f book-index-style-properties)]{Demo}

This paragraph is about @as-index{examples}.

This paragraph is about @as-index{examples}, too.

@index-section[]
```