Browser: Simple HTML Rendering

Version 9.4.0.5

November 14, 2026

The `browser` library provides the following procedures and classes for parsing and viewing HTML files. The `browser/htmltext` library provides a simplified interface for rendering to a subclass of the GRacket `text%` class. The `browser/external` library provides utilities for launching an external browser (such as Firefox).

1 Browser

```
(require browser)
```
package: drracket

The browser supports basic HTML commands, plus special Racket hyperlinks of the form `<A RACKET=sexpr>...</A>`. When the user clicks on such a link, the string `sexpr` is parsed as a Racket program and evaluated. Since `sexpr` is likely to contain Racket strings, and since escape characters are difficult for people to read, a `|` character in `sexpr` is converted to a `"` character before it is parsed. Thus,

```
<A RACKET="|This goes nowhere.|">Nowhere</A>
```

creates a "Nowhere" hyperlink, which executes the Racket program

```
"This goes nowhere."
```

The value of that program is a string. When a Racket hyperlink returns a string, it is parsed as a new HTML document. Thus, where the use clicks on "Nowhere," the result is a new page that says "This goes nowhere."

The browser also treats comment forms containing `RACKET=sexpr` specially. Whereas the `<A RACKET=sexpr>...</A>` form executes the expression when the user clicks, the `RACKET` expression in a comment is executed immediately during HTML rendering. If the result is a string, the comment is replaced in the input HTML stream with the content of the string. Thus,

```
<!-- RACKET="(format |<B>Here</B>: ~a| (current-directory))" -->
```

inserts the path of the current working directory into the containing document (and "Here" is boldfaced). If the result is a snip instead of a string, it replaces the comment in the document. Other types of return values are ignored.

If the html file is being accessed as a `file:` url, the `current-load-relative-directory` parameter is set to the directory during the evaluation of the Racket code (in both examples). The Racket code is executed through `eval`.

The `RACKET` forms are disabled unless the web page is a `file:` url that points into the `doc` collection.

```
(open-url url) → (is-a?/c hyper-frame%)
  url : (or/c url? string? input-port?)
```

Opens the given url in a vanilla browser frame and returns the frame. The frame is an instance of `hyper-frame%`.

```
(html-img-ok) → boolean?
(html-img-ok ok?) → void?
  ok? : any/c
```

A parameter that determines whether the browser attempts to download and render images.

```
(html-eval-ok) → boolean?
(html-eval-ok ok?) → void?
  ok? : any/c
```

A parameter that determines whether RACKET= tags are evaluated.

```
hyper-frame<%> : interface?
```

```
(send a-hyper-frame get-hyper-panel%) → (subclass?/c panel%)
```

Returns the class that is instantiated when the frame is created. Must be a panel with hyper-panel-mixin mixed in. Defaults to just returning hyper-panel%.

```
(send a-hyper-frame get-hyper-panel) → (is-a?/c panel%)
```

Returns the hyper panel in this frame.

```
hyper-frame-mixin : (class? . -> . class?)
  argument extends/implements: frame%
  result implements: hyper-frame<%>
```

```
 (new hyper-frame-mixin
    [url url]
    ...superclass-args...)
 → (is-a?/c hyper-frame-mixin)
  url : (or/c url? string? input-port?)
```

Shows the frame and visits *url*.

```
hyper-no-show-frame% : class?
  superclass: (hyper-frame-mixin (frame:status-line-mixin frame:basic%))
```

```
hyper-no-show-frame-mixin : (class? . -> . class?)
  argument extends/implements: frame%
```

The same as the hyper-frame-mixin, except that it doesn't show the frame and the initialization arguments are unchanged.

```
hyper-frame% : class?
  superclass: (hyper-no-show-frame-mixin (frame:status-line-mixin frame:basic%))
```

```
hyper-text<%> : interface?
```

```
(send a-hyper-text url-allows-evalling? url) → boolean?
  url : (or/c port? url?)
```

Determines if RACKET annotations are actually evaluated, for a given url.

```
hyper-text-mixin : (class? . -> . class?)
  argument extends/implements: text%
  result implements: hyper-text<%>
```

An instance of a hyper-text-mixin-extended class should be displayed only in an instance of a class created with hyper-canvas-mixin.

```
(new hyper-text-mixin
   [url url]
   [status-frame status-frame]
   [post-data post-data]
   ...superclass-args...)
→ (is-a?/c hyper-text-mixin)
 url : (or/c url? string? input-port?)
 status-frame : (or/c (is-a?/c top-level-window<%>) false/c)
 post-data : (or/c false/c bytes?)
```

The *url* is loaded into the text% object (using the reload method), a top-level window for status messages and dialogs, a progress procedure used as for get-url, and either #f or a post string to be sent to a web server (technically changing the GET to a POST).

Sets the autowrap-bitmap to #f.

```
(send a-hyper-text map-shift-style start
                                   end
                                   shift-style) → void?
```

```
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
  shift-style : style<%>
```

Maps the given style over the given range.

```
(send a-hyper-text make-link-style start
                                   end)  → void?
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
```

Changes the style for the given range to the link style.

```
(send a-hyper-text get-url)
 → (or/c url? string? input-port? false/c)
```

Returns the URL displayed by the editor, or `#f` if there is none.

```
(send a-hyper-text get-title) → string?
```

Gets the page's title.

```
(send a-hyper-text set-title str) → void?
  str : string?
```

Sets the page's title.

```
(send a-hyper-text hyper-delta) → style-delta%
```

Override this method to set the link style.

```
(send a-hyper-text add-tag name pos) → void?
  name : string?
  pos : exact-nonnegative-integer?
```

Installs a tag.

```
(send a-hyper-text find-tag name/number)
 → (or/c exact-nonnegative-integer? false/c)
  name/number : (or/c string? exact-nonnegative-integer?)
```

Finds the location of a tag in the buffer (where tags are installed in HTML with `<A NAME="name">`) and returns its position. If `name` is a number, the number is returned (assumed to be an offset rather than a tag). Otherwise, if the tag is not found, `#f` is returned.

```
(send a-hyper-text remove-tag name) → void?
  name : string?
```

Removes a tag.

```
(send a-hyper-text post-url  url
                             [post-data-bytes]) → void?
  url : (or/c string? url?)
  post-data-bytes : (or/c bytes? false/c) = #f
```

Follows the link, optionally with the given post data.

```
(send a-hyper-text add-link start end url) → void?
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
  url : (or/c url? string?)
```

Installs a hyperlink.

```
(send a-hyper-text add-racket-callback start
                                       end
                                       racket-expr) → void?
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
  racket-expr : string?
```

Installs a Racket evaluation hyperlink.

```
(send a-hyper-text add-thunk-callback start
                                      end
                                      thunk) → void?
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
  thunk : (-> any)
```

Installs a thunk-based hyperlink.

```
(send a-hyper-text eval-racket-string str) → any
  str : string?
```

Called to handle the `<A RACKET="expr">...</A>` tag and `<! RACKET="expr">` comments (see above). Evaluates the string; if the result is a string, it is opened as an HTML page.

```
(send a-hyper-text reload) → void?
```

Reloads the current page.

By default, the text uses the basic style named "Html Standard" in the editor (if it exists).

```
(send a-hyper-text remap-url url) → (or/c url? string?)
  url : (or/c url? string?)
```

When visiting a new page, this method is called to remap the url. The remapped url is used in place of the original url. If this method returns #f, the page doesn't go anywhere.

This method may be killed (if the user clicks the "stop" button).

```
(send a-hyper-text get-hyper-keymap) → (is-a?/c keymap%)
```

Returns a keymap suitable for frame-level handling of events to redirect page-up, etc. to the browser canvas.

```
hyper-canvas% : class?
  superclass: (hyper-canvas-mixin canvas:basic%)
```

```
hyper-text% : class?
  superclass: (hyper-text-mixin text:keymap%)
```

Extends the text:keymap% class to support standard key bindings in the browser window.

```
hyper-canvas-mixin : (class? . -> . class?)
  argument extends/implements: editor-canvas%
```

A hyper-can-mixin-extended canvas's parent should be an instance of a class derived with hyper-panel-mixin.

```
(new hyper-canvas-mixin ...superclass-args...)
 → (is-a?/c hyper-canvas-mixin)
```

```
(send a-hyper-canvas get-editor%) → (subclass?/c text%)
```

Returns the class used to implement the editor in the browser window. It should be derived from `hyper-text%` and should pass on the initialization arguments to `hyper-text%`.

The dynamic extent of the initialization of this editor is called on a thread that may be killed (via a custodian shutdown). In that case, the editor in the browser's editor-canvas may not be an instance of this class.

```
(send a-hyper-canvas current-page) → any/c
```

Returns a representation of the currently displayed page, which includes a particular editor and a visible range within the editor.

```
(send a-hyper-canvas goto-url  url
                               relative-to-url
                              [progress-proc
                               post-data])      → void?
  url : (or/c url? string?)
  relative-to-url : (or/c url? string? false/c)
  progress-proc : (boolean? . -> . any) = void
  post-data : (or/c bytes? false/c) = #f
```

Changes to the given url, loading it by calling the `make-editor` method. If *relative-to-url* is not `#f`, it must be a URL for resolving *url* as a relative URL. *url* may also be a port, in which case, *relative-to-url* must be `#f`.

The *progress-proc* procedure is called with a boolean at the point where the URL has been resolved and enough progress has been made to dismiss any message that the URL is being resolved. The procedure is called with `#t` if the URL will be loaded into a browser window, `#f` otherwise (e.g., the user will save the URL content to a file).

If `post-data-bytes` is a byte string instead of false, the URL GET is changed to a POST with the given data.

```
(send a-hyper-canvas set-page page notify?) → void?
  page : any/c
  notify? : any/c
```

Changes to the given page. If *notify?* is not `#f`, the canvas's parent is notified about the change by calling its `leaving-page` method.

```
(send a-hyper-canvas after-set-page) → void?
```

Called during `set-page`. Does nothing by default.

```
hyper-panel<%> : interface?
```

```
hyper-panel-mixin : (class? . -> . class?)
  argument extends/implements: area-container<%>
  result implements: hyper-panel<%>
```

```
(new hyper-panel-mixin
   [info-line? info-line?]
   ...superclass-args...)
 → (is-a?/c hyper-panel-mixin)
  info-line? : any/c
```

Creates controls and a hyper text canvas. The controls permit a user to move back and forth in the hypertext history.

The *info-line?* argument indicates whether the browser should contain a line to display special DOCNOTE tags in a page. Such tags are used primarily by the PLT documentation.

```
(send a-hyper-panel make-canvas container) → void?
  container : (is-a?/c area-container<%>)
```

Creates the panel's hypertext canvas, an instance of a class derived using hyper-canvas-mixin. This method is called during initialization.

```
(send a-hyper-panel get-canvas%)
 → (subclass?/c editor-canvas%)
```

Returns the class instantiated by make-canvas. It must be derived from hyper-canvas-mixin.

```
(send a-hyper-panel make-control-bar-
panel container) → any/c
  container : (is-a?/c area-container<%>)
```

Creates the panel's sub-container for the control bar containing the navigation buttons. If #f is returned, the panel will have no control bar. The default method instantiates horizontal-panel%.

```
(send a-hyper-panel rewind) → void?
```

Goes back one page, if possible.

```
(send a-hyper-panel forward) → void?
```

Goes forward one page, if possible.

```
(send a-hyper-panel get-canvas) → (is-a?/c editor-canvas%)
```

Gets the hypertext canvas.

```
(send a-hyper-panel on-navigate) → void?
```

Callback that is invoked any time the displayed hypertext page changes (either by clicking on a link in the canvas or by `rewind` or `forward` calls).

```
(send a-hyper-panel leaving-page page
                                 new-page) → any
  page : any/c
  new-page : any/c
```

This method is called by the hypertext canvas to notify the panel that the hypertext page changed. The *page* is `#f` if *new-page* is the first page for the canvas. See also `page->editor`.

```
(send a-hyper-panel filter-notes notes) → (listof string?)
  notes : (listof string?)
```

Given the notes from a page as a list of strings (where each string is a note), returns a single string to print above the page.

```
(send a-hyper-panel reload) → void?
```

Reloads the currently visible page by calling the `reload` method of the currently displayed hyper-text.

```
hyper-panel% : class?
  superclass: (hyper-panel-mixin vertical-panel%)
```

```
(editor->page editor) → any/c
  editor : (is-a?/c text%)
```

Creates a page record for the given editor, suitable for use with the `set-page` method of `hyper-canvas-mixin`.

```
(page->editor page) → (is-a?/c text%)
  page : any/c
```

Extracts the editor from a page record.

```
(bullet-size) → exact-nonnegative-integer?
(bullet-size n) → void?
  n : exact-nonnegative-integer?
```

Parameter controlling the point size of a bullet.

```
image-map-snip% : class?
  superclass: snip%
```

Instances of this class behave like image-snip% objects, except they have a `<map> ... </map>` associated with them and when clicking on them (in the map) they will cause their init arg text to follow the corresponding link.

```
(new image-map-snip% [html-text html-text])
 → (is-a?/c image-map-snip%)
  html-text : (is-a?/c html-text<%>)
```

```
(send an-image-map-snip set-key key) → void?
  key : string?
```

Sets the key for the image map (eg, "#key").

```
(send an-image-map-snip get-key) → string?
```

Returns the current key.

```
(send an-image-map-snip add-area shape
                                region
                                href)  → void?
  shape : string?
  region : (listof number?)
  href : string?
```

Registers the shape named by *shape* whose coordinates are specified by *region* to go to *href* when that region of the image is clicked on.

```
(struct exn:cancelled exn ()
    #:extra-constructor-name make-exn:cancelled)
```

This exception may be raised by the reload method.

```
(struct exn:file-saved-instead exn (pathname)
    #:extra-constructor-name make-exn:file-saved-instead)
  pathname : path-string?
```

This exception may be raised by the `reload` method.

2 Browser Unit

```
(require browser/browser-unit)
```
package: drracket

```
browser@ : unit?
```

Imports mred^, tcp^, and url^, and exports browser^.

3 Browser Signature

```
(require browser/browser-sig)
```
package: drracket

```
browser^ : signature
```

Includes all of the bindings of the `browser` library.

4 HTML As Text Editor

```
(require browser/htmltext)
```
package: drracket

```
html-text<%> : interface?
  implements: text%
```

```
(send a-html-text get-url) → (or/c url? string? false/c)
```

Returns a base URL used for building relative URLs, or #f if no base is available.

```
(send a-html-text set-title str) → void?
  str : string?
```

Registers the title *str* for the rendered page.

```
(send a-html-text add-link start end url) → void?
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
  url : (or/c url? string?)
```

Registers a hyperlink for the given region in rendered page.

```
(send a-html-text add-tag name pos) → void?
  name : string?
  pos : exact-nonnegative-integer?
```

Installs a tag.

```
(send a-html-text make-link-style start
                                  end)  → void?
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
```

Changes the style for the given range to the link style.

```
(send a-html-text add-racket-callback start
                                      end
                                      racket-expr) → void?
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
  racket-expr : string?
```

Installs a Racket evaluation hyperlink.

```
(send a-html-text add-thunk-callback start
                                     end
                                     thunk) → void?
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
  thunk : (-> any)
```

Installs a thunk-based hyperlink.

```
(send a-html-text post-url  url
                            [post-data-bytes]) → void?
  url : (or/c string? url?)
  post-data-bytes : (or/c bytes? false/c) = #f
```

Follows the link, optionally with the given post data.

```
html-text-mixin : (class? . -> . class?)
  argument extends/implements: text%
```

Extends the given `text%` class with implementations of the `html-text<%>` methods. Hyperlinks are attached to clickbacks that use `send-url` from `net/sendurl`.

```
(render-html-to-text in
                     dest
                     load-img?
                     eval-rkt?) → void?
  in : input-port?
  dest : (is-a? html-text<%>)
  load-img? : any/c
  eval-rkt? : any/c
```

Reads HTML from *in* and renders it to *dest*. If *load-img?* is `#f`, then images are rendered as Xed-out boxes. If *eval-rkt?* is `#f`, then RACKET hyperlink expressions and comments are not evaluated.

Uses the style named `"Html Standard"` in the editor's style-list (if it exists) for all of the inserted text's default style.

5 Launching an External Browser

```
(require browser/external)
```
package: drracket-core-lib

```
(send-url str [separate-window?]) → null
  str : null
  separate-window? : void = #t
```

Like send-url from net/sendurl, but on Unix, the user is prompted for a browser to use if none is recorded in the preferences file.

```
(browser-preference? v) → boolean?
  v : any/c
```

Returns #t if *v* is a valid browser preference.

```
(update-browser-preference url) → void?
  url : (or/c string? false/c)
```

On Unix, prompts the user for a browser preference and records the user choice as a framework preference (even if one is already recorded). If *url* is not #f, it is used in the dialog to explain which URL is to be opened; if it is #f, the 'internal will be one of the options for the user.

```
(install-help-browser-preference-panel) → void?
```

Installs a framework preference panel for "Browser" options.

```
(add-to-browser-prefs-panel proc) → void?
  proc : ((is-a?/c panel%) . -> . any)
```

The *proc* is called when the "Browser" panel is constructed for preferences. The supplied argument is the panel, so *proc* can add additional option controls. If the panel is already created, *proc* is called immediately.

6 DrRacket Browser Preference Panel

```
(require browser/tool)
```
package: drracket-core-lib

```
tool@ : unit?
```

A unit that implements a DrRacket tool to add the “Browser” preference panel.