Expeditor: Terminal Expression Editor

Version 9.4.0.4

November 5, 2026

```
(require expeditor)
```
package: `expeditor-lib`

The expeditor (the **exp**ression **editor**) supports multi-line editing with indentation and completion support within a terminal. It's based on Chez Scheme's expression editor, but adapts to Racket languages using the same hooks and APIs as DrRacket. Normally, the expeditor is run automatically by `xrepl`, which is turn loaded by the `racket` command-line executable.

For customization of keybindings, see §2 "Customizing Expeditor". To disable or enable color in expeditor, you can use xrepl's `,color` command.

1 Default Key Bindings

In the keybinding descriptions below, a "Meta-" combination usually means "Alt-" or "Option-", depending on your keyboard. It also can be typed as the Esc key (pressed then released) followed by the rest of the combination; a terminal will typically report a combination using the Alt or Option key as that Esc sequence. In a "Ctl-" combination, the letter case of the key doesn't matter (i.e., doesn't require holding the Shift key).

Evaluation, Indentation, and Completion

- Return or Ctl-J — Reads and evaluates the current entry, if it is complete, and otherwise inserts a newline and auto-indents. The notion of "complete" depends on a language, but typically includes requirements like no unbalanced parentheses. (Implementation: `ee-newline/accept`)
- Meta-Return — Inserts a newline and indents. (Implementation: `ee-newline`)
- Meta-Ctl-J — Reads and evaluates the current entry, even if it is not otherwise recognized as complete. (Implementation: `ee-accept`)
- Ctl-O — Creates a new line for input, similar to a non-accepting Return, but does not move the cursor to the new line or indent. (Implementation: `ee-open-line`)
- Tab — Either indents or completes, depending on the text before the cursor. If no text is present before the cursor on the same line, then the line is indented or cycled to the next possible indentation. If the cursor is after an identifier, it's completed or a list of possible completions is shown. Completion depends on the language, but it is typically drawn from the set of available top-level bindings. (Implementation: `ee-id-completion/indent`)
- Shift-Tab — Like Tab completion or indentation, but cycles through indentation options in reverse order. (Implementation: `ee-id-completion/indent/reverse`)
- Ctl-R — Steps through the next possible completion when there are multiple possible completions. (Implementation: `ee-next-id-completion`)
- Meta-Tab — Indents the current line or cycles though possible indentations. The cursor is moved to just after the indentation before the rest of the line content. (Implementation: `ee-indent`)
- Meta-q or Meta-Q or Meta-Ctl-Q — Reindents the full editor region. (Implementation: `ee-indent-all`)

Navigation

- Left or Ctl-B — Moves the cursor back one character. (Implementation: `ee-backward-char`)
- Right or Ctl-F — Moves the cursor forward one character. (Implementation: `ee-forward-char`)
- Up or Ctl-P — Moves the cursor up to the previous line—unless the cursor is at the start of the editor-region, in which case replaces the editor region with the previous history entry. (Implementation: `ee-previous-line`)
- Down or Ctl-N — Moves the cursor down to the next line—unless the cursor is at the end of the editor region, in which case replaces the editor region with the next history entry. (Implementation: `ee-next-line`)
- Home or Ctl-A — Moves the cursor to the start of the current line. (Implementation: `ee-beginning-of-line`)
- End or Ctl-E — Moves the cursor to the end of the current line. (Implementation: `ee-end-of-line`)
- PageUp or Ctl-X [— Moves the cursor up to the previous page. (Implementation: `ee-backward-page`)
- PageDown or Ctl-X] — Moves the cursor down to the next page. (Implementation: `ee-backward-page`)
- Meta-< — Moves the cursor to the start of the editor region. (Implementation: `ee-beginning-of-entry`)
- Meta-> — Moves the cursor to the end of the editor region. (Implementation: `ee-end-of-entry`)
- Ctl-Right or Meta-f or Meta-F — Moves the cursor forward one whitespace-delimited word. (Implementation: `ee-forward-word`)
- Ctl-Left or Meta-b or Meta-B — Moves the cursor backward one whitespace-delimited word. (Implementation: `ee-backward-word`)
- Meta-] — Moves the cursor to the opener or closer opposite the one under the cursor. (Implementation: `ee-goto-matching-delimiter`)
- Ctl-] — Flashes the cursor on the opener or closer opposite the one under the cursor. (Implementation: `ee-flash-matching-delimiter`)
- Meta-Ctl-Right or Meta-Ctl-F — Moves the cursor forward one expression, where the definition of "expression" is language-specific. (Implementation: `ee-forward-exp`)
- Meta-Ctl-Left or Meta-Ctl-B — Moves the cursor backward one language-specific expression. (Implementation: `ee-backward-exp`)

- Meta-Ctl-U — Moves the cursor upward/outward one language-specific expression. (Implementation: `ee-upward-exp`)
- Meta-Ctl-D — Moves the cursor downward/inward one language-specific expression. (Implementation: `ee-downward-exp`)
- Ctl-X Ctl-X — Moves the cursor to the location of the mark while setting the mark to the cursor's current position. (Implementation: `ee-exchange-point-and-mark`)

History

- Meta-Up or Meta-Ctl-P — Replaces the editor region with the previous history entry. (Implementation: `ee-history-bwd`)
- Meta-Down or Meta-Ctl-N — Replaces the editor region with the next history entry. (Implementation: `ee-history-fwd`)
- Meta-p — Replaces the editor region with the previous history entry that starts the same as the current editor content. (Implementation: `ee-history-bwd-prefix`)
- Meta-P — Replaces the editor region with the previous history entry that includes the same as the current editor content. (Implementation: `ee-history-bwd-contains`)
- Meta-n — Replaces the editor region with the next history entry that starts the same as the current editor content. (Implementation: `ee-history-fwd-prefix`)
- Meta-N — Replaces the editor region with the next history entry that includes the same as the current editor content. (Implementation: `ee-history-fwd-contains`)

Deletion, Insertion, and Transposition

- Backspace or Ctl-H — Deletes the previous character, if any. (Implementation: `ee-backward-delete-char`)
- Ctl-D — Deletes the next character, if any—unless the editor region is empty, in which case returns an end-of-file as the input. (Implementation: `ee-eof/delete-char`)
- Delete — Deletes the next character, if any. (Implementation: `ee-delete-char`)
- Ctl-U — Deletes the content of the current line, no matter where the cursor is within the line. (Implementation: `ee-delete-line`)
- Ctl-K or Meta-k — Deletes the content of the current line following the cursor, or merges the next line with the current one if the cursor is at the end of the line. (Implementation: `ee-delete-to-eol`)

- Ctl-G — Deletes the full content of the editor region. (Implementation: `ee-delete-entry`)
- Ctl-C — Deletes the full content of the editor region, and also moves to the end of the history—unless the editor region is empty, in which case sends a break signal to the current thread. (Implementation: `ee-reset-entry/break`)
- Meta-d — Deletes one whitespace-delimited word after the cursor. (Implementation: `ee-delete-word`)
- Meta-Delete or Meta-Ctl-K — Deletes one expression after the cursor, where the definition of "expression" is language-specific. (Implementation: `ee-delete-exp`)
- Meta-Backspace or Meta-Ctl-H — Deletes one expression before the cursor. (Implementation: `ee-backward-delete-exp`)
- Ctl-@ or Ctl-Ctl- — Set the mark to be the same position as the cursor. The *mark* is a kind of second cursor, but invisible, that is used by various editing operations. (Implementation: `ee-set-mark`)
- Ctl-W — Deletes content between the cursor and the mark. When no mark is set, deletes one expression before the cursor. (Implementation: `ee-delete-between-point-and-mark-or-backward`)
- Ctl-Y — Inserts content previously deleted, where multiple consecutive deletions accumulate to one set of content to insert. (Implementation: `ee-yank-kill-buffer`)
- Ctl-V — Inserts the content of the system clipboard. (Implementation: `ee-yank-selection`)
- Ctl-T — Transposes characters to left and right of the cursor—unless the cursor is at the end of a line, in which case transposes the previous two characters. (Implementation: `ee-transpose-char`)
- Meta-t — Transposes space-delimited words to the left and right of the cursor. (Implementation: `ee-transpose-word`)
- Meta-Ctl-T — Transposes language-specific expressions to the left and right of the cursor. (Implementation: `ee-transpose-exp`)

Process Control

- Ctl-L — Refreshes the editor region's display. (Implementation: `ee-redisplay`)
- Ctl-Z — Suspends the current process. (Implementation: `ee-suspend-process`)

2 Customizing Expeditor

```
(require (submod expeditor configure))
```

When `expeditor-configure` is called—such as when xrepl initializes the expeditor, which is the default behavior when running `racket` at the command line—it dynamically requires the module file reported by `(expeditor-init-file-path)`, if that file exists. The module file can import `(submod expeditor configure)` to configure key bindings and colors.

For example, the following module as `(expeditor-init-file-path)` changes the behavior of Ctl-J and changes the color of literals from green to magenta:

```
#lang racket/base
(require (submod expeditor configure))

(expeditor-bind-key! "^J" ee-newline)
(expeditor-set-syntax-color! 'literal 'magenta)
```

2.1 Key-Handling Functions

A key-handling function accepts three values: a representation of the terminal state, a representation of the current editor region, and a character. The result is a representation of the editor region (usually the one passed in) or `#f` to indicate that the current editor region should be accepted as input.

```
(expeditor-bind-key! key handler) → void?
  key : string?
  handler : (eestate? entry? char . -> . (or/c #f entry?))
```

Binds the action of *key* to *handler*, where *handler* is typically one of the `ee-` functions described below.

The *key* string encodes either a single key or a sequence of keys:

- The sequence `\e` (so, in a literal string as `"\\e"`) is treated as Escape, which at the start of a sequence is normally the way terminals report “Meta-” key combinations.
- A `^` prefix on a character implies a “Ctl-” combination, like `"^a"` for Ctl-A.
- The sequence `\\` is a backslash (so, in a literal string as `"\\\\"`).
- The sequence `\^` is the character character.
- Anything else stands for itself.

The result of a key binding is a potentially updated entry, where only predefined functions can update an entry, or `#f` to indicate that the current entry should be accepted as an `expeditor-read` result.

As examples, here are a few bindings from the default set:

```
(expeditor-bind-key! "^B" ee-backward-char) ; Ctl-B
(expeditor-bind-key! "\\ef" ee-forward-word) ; Esc-f
(expeditor-bind-key! "\\e[C" ee-forward-char) ; Right
(expeditor-bind-key! "\\e[1;5C" ee-forward-word) ; Ctl-Right
```

The Right and Ctl-Right bindings are derived from a typical sequence that a terminal generates for those key combinations. In your terminal, the `od` program may be helpful in figuring how key presses turn into program input.

```
(ee-insert-self/paren ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

This function is the default operation for unmapped keys.

Like `ee-insert-self`, but if *c* is a "parenthesis" character, flashes its match like `ee-flash-matching-delimiter`. Furthermore, if *c* is a closing "parenthesis", it may be corrected automatically based on its apparently intended match.

```
(ee-insert-self ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Inserts *c*, as long as it is not a control character.

```
((make-ee-insert-string s) ee entry c) → entry?
  s : string?
  ee : eestate?
  entry : entry?
  c : char?
```

Creates a key-handling function that inserts *s*.

```
(ee-newline/accept ee entry c) → (or/c entry? #f)
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Return. Note that the return value is `#f` in the case that the input should be accepted.

```
(ee-accept ee entry c) → (or/c #f entry?)
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Ctl-J. Note that the return value is `#f` in the case that the input should be accepted.

```
(ee-newline ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Return.

```
(ee-open-line ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-O.

```
(ee-indent ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Tab.

```
(ee-indent-all ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-q.

```
(ee-id-completion/indent ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Tab.

```
(ee-id-completion/indent/reverse ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Shift-Tab.

Added in version 1.2.

```
(ee-id-completion ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Like ee-id-completion, but always attempts completion instead of tabbing.

```
(ee-next-id-completion ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-R.

```
(ee-backward-char ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Left.

```
(ee-forward-char ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Right.

```
(ee-next-line ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Down.

```
(ee-previous-line ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Up.

```
(ee-forward-word ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-Right.

```
(ee-forward-exp ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Ctl-Right.

```
(ee-backward-word ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-Left.

```
(ee-backward-exp ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Ctl-Left.

```
(ee-upward-exp ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Ctl-U.

```
(ee-downward-exp ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Ctl-D.

```
(ee-backward-page ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for PageUp.

```
(ee-forward-page ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for PageDown.

```
(ee-beginning-of-line ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Home.

```
(ee-end-of-line ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for End.

```
(ee-beginning-of-entry ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-<.

```
(ee-end-of-entry ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta->.

```
(ee-goto-matching-delimiter ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-].

```
(ee-flash-matching-delimiter ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-].

```
(ee-transpose-char ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-T.

```
(ee-transpose-word ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-t.

```
(ee-transpose-exp ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Ctl-T.

```
(ee-set-mark ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-@.

```
(ee-exchange-point-and-mark ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-X Ctl-X.

```
(ee-delete-char ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Delete.

```
(ee-backward-delete-char ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Backspace.

```
(ee-delete-line ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-U.

```
(ee-delete-to-eol ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-K.

```
(ee-delete-between-point-and-mark-or-backward ee
                                              entry
                                              c)    → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-W.

```
(ee-delete-entry ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-G.

```
(ee-reset-entry/break ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-C.

```
(ee-reset-entry ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Like ee-reset-entry/break, but never sends a break signal.

```
(ee-delete-word ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-d.

```
(ee-delete-exp ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Delete.

```
(ee-backward-delete-exp ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Backspace.

```
(ee-yank-selection ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-V.

```
(ee-yank-kill-buffer ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-Y.

```
(ee-eof/delete-char ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-D.

```
(ee-eof ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Like `ee-eof/delete-char`, but always return an end-of-file.

```
(ee-redisplay ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-L.

```
(ee-history-bwd ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Up. See also current-ee-backward-history-point.

```
(ee-history-fwd ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-Down. See also current-ee-forward-history-point.

```
(ee-history-bwd-prefix ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-p. See also current-ee-backward-history-point.

```
(ee-history-bwd-contains ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-P. See also current-ee-backward-history-point.

```
(ee-history-fwd-prefix ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-n. See also current-ee-forward-history-point.

```
(ee-history-fwd-contains ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Meta-N. See also `current-ee-forward-history-point`.

```
(ee-command-repeat ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Accumulates *c* into a repeat count if it is a digit. Otherwise, performs the command associated with *c* the number of times set up for repeating.

```
(ee-suspend-process ee entry c) → entry?
  ee : eestate?
  entry : entry?
  c : char?
```

Implements the behavior described for Ctl-Z.

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

Returns `#t` if *v* is a representation of the terminal state, `#f` otherwise.

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

Returns `#t` if *v* is a representation of the current editor region, `#f` otherwise.

2.2 Colors

```
(expeditor-set-syntax-color! category
                             color)   → void?
  category : (or/c 'error
                   'paren
                   'literal
                   'identifier
                   'comment)
```

```
  color : (or/c 'default
                'black
                'white
                'red
                'green
                'blue
                'yellow
                'cyan
                'magenta
                'dark-gray
                'light-gray
                'light-red
                'light-green
                'light-blue
                'light-yellow
                'light-cyan
                'light-magenta)
```

Sets the color used for a syntactic category when coloring is enabled. The `'error` color is used by `expeditor-error-display` in addition to being used for invalid tokens.

2.3 History Navigation

```
(current-ee-backward-history-point)
 → (or/c 'start 'top 'bottom 'end)
(current-ee-backward-history-point start-at) → void?
  start-at : (or/c 'start 'top 'bottom 'end)
```

A parameter that determines where the cursor starts when the editor content is changed to an earlier entry in the history via `ee-history-bwd` and similar functions:

- `'start` — at the start of the entry
- `'top` — at the end of the first line of the entry
- `'bottom` or `'end` — at the end of the last line of the entry

The default is `'top`.

```
(current-ee-forward-history-point)
 → (or/c 'start 'top 'bottom 'end)
(current-ee-forward-history-point start-at) → void?
  start-at : (or/c 'start 'top 'bottom 'end)
```

Like `current-ee-backward-history-point`, but used when the editor content is changed to a later entry in the history via `ee-history-fwd` and similar functions.

The default is `'bottom`.

3 Expeditor API

```
(expeditor-open history) → (or/c eestate? #f)
  history : (listof string?)
```

Attempts to start the expeditor. On success, which requires that `(current-input-port)` and `(current-output-port)` are terminal ports and the terminal configuration is recognized, the result is a representation of the terminal state. The result is `#f` if the expeditor cannot be initialized.

The *history* argument provides the initial list of history entries, which is navigated by functions like `ee-history-bwd`. This history is updated as input is accepted during `expeditor-read`, and `expeditor-close` reports an updated history. The amount of preserved history is limited.

```
(expeditor-close ee) → (listof string?)
  ee : estate?
```

Closes the expeditor, relinquishing terminal configuration and resources, if any. The result is the expeditor's history as initialized by `expeditor-open` and updated by `expeditor-read` calls.

```
(expeditor-read ee [#:prompt prompt-str]) → any/c
  ee : estate?
  prompt-str : string? = ">"
```

Reads input from the terminal. The *ee* argument holds terminal state as well as history that is updated during `expeditor-read`. The *prompt-str* is used as a prompt; a space is added between *prompt-str* and input, unless *prompt-str* is `""`.

Changed in version 1.1 of package `expeditor-lib`: Added the `#:prompt` argument.

```
(call-with-expeditor  proc
                      [#:prompt prompt-str]) → any
  proc : ((->* () (#:prompt string?) any/c) -> any)
  prompt-str : string? = ">"
```

Combines `expeditor-open`, a call to *proc*, and `expeditor-close`, where the reading procedure passed to *proc* can be called any number of times to read input. The *prompt-str* argument is used in the same way as for `expeditor-read`, the reading procedure can also receive an optional string to update the *prompt-str*.

Expeditor history is initialized from `current-expeditor-history` on open, and the value of `current-expeditor-history` is updated with the new history on close.

Changed in version 1.1 of package `expeditor-lib`: Added the `#:prompt` argument.

```
(expeditor-configure) → void?
```

Sets expeditor parameters based on `current-interaction-info`, the user's preferences file, and `(expeditor-init-file-path)`.

The `current-expeditor-reader` parameter is first set to use `current-read-interaction`.

then, `expeditor-configure` checks for information via `current-interaction-info`, currently checking for the following keys:

- `'color-lexer` — Sets `current-expeditor-lexer`. If `'color-lexer` is not provided and `syntax-color/racket-lexer` is available, then the Racket lexer is installed.
- `'drracket:submit-predicate` — Sets `current-expeditor-ready-checker`.
- `'drracket:paren-matches` — Sets `current-expeditor-parentheses`.
- `'drracket:grouping-position` — Sets `current-expeditor-grouper`.
- `'drracket:indentation`, `'drracket:range-indentation`, and `'drracket:range-indentation/reverse-choices` — Sets `current-expeditor-indenter` based on a combination of both values.

The `'expeditor-color-enabled` preference (via `get-preference`) determines `current-expeditor-color-enabled`.

Finally, if the file named by `(expeditor-init-file-path)`, it is `dynamic-required`.

```
(expeditor-init-file-path) → path?
```

Returns a path that is used by `expeditor-configure`.

If `(find-system-path 'init-dir)` produces a different result than `(find-system-path 'home-dir)`, then the result is `(build-path (find-system-path 'init-dir) "expeditor.rkt")`. Otherwise, the result is `(build-path (find-system-path 'home-dir) ".expeditor.rkt")`.

```
(current-expeditor-reader) → (input-port? . -> . any/c)
(current-expeditor-reader proc) → void?
  proc : (input-port? . -> . any/c)
```

A parameter that determines the reader used to parse input when an entry is accepted. The default function uses `read`.

```
(current-expeditor-post-skipper) → (input-port? . -> . any/c)
(current-expeditor-post-skipper proc) → void?
  proc : (input-port? . -> . any/c)
```

A parameter that determines a function used to consume extra whitespace after a reader consumes from an accepted entry. The default function consumes whitespace.

```
(current-expeditor-lexer) → procedure?
(current-expeditor-lexer proc) → void?
  proc : procedure?
```

A parameter that determines the lexer used for syntax coloring and parenthesis matching. See the DrRacket manual for more information. The default function simply recognizes common parenthesis-like characters.

```
(current-expeditor-ready-checker) → procedure?
(current-expeditor-ready-checker proc) → void?
  proc : procedure?
```

A parameter that determines how expeditor entry is treated as ready to accept or not. See the DrRacket manual for more information. The default function attempts to `read` all input, returning `#f` only if `exn:fail:read:eof` is raised.

```
(current-expeditor-parentheses)
 → (listof (list/c symbol? symbol?))
(current-expeditor-parentheses pairs) → void?
  pairs : (listof (list/c symbol? symbol?))
```

A parameter that determines character sequences that are considered matching opener and closer pairs. See the DrRacket manual for more information. The default is `'((|(| |)|) (|[| |]|) (|{| |}|))`.

```
(current-expeditor-grouper) → procedure?
(current-expeditor-grouper proc) → void?
  proc : procedure?
```

A parameter that determines how expression-based navigation operators work. See the DrRacket manual for more information.

```
(current-expeditor-indenter) → procedure?
(current-expeditor-indenter proc) → void?
  proc : procedure?
```

A parameter that determines how automatic indentation works. It expects either three or four arguments: a representation of the editor, a position in the editor, a boolean indicating whether the indentation request is automatic due to starting a new line, and an optional

boolean indicating whether to cycle through indentation choices in reverse order. The protocol for results is the same as an indentation function for DrRacket. See the DrRacket manual for more information.

Changed in version 1.2 of package `expeditor-lib`: Added support for an optional fourth argument.

```
(current-expeditor-color-enabled) → boolean?
(current-expeditor-color-enabled on?) → void?
  on? : boolean?
```

A parameter that determines whether syntax and error coloring are enabled.

```
(current-expeditor-history) → (listof string?)
(current-expeditor-history strs) → void?
  strs : (listof string?)
```

Expeditor history as consumed and produced by `call-with-expeditor`.

```
(current-expeditor-history-whitespace-trim-enabled) → boolean?
(current-expeditor-history-whitespace-trim-enabled on?) → void?
  on? : any/c
```

A parameter that determines whether tailing whitespace is trimmed from input before recording it as history. The default is `#t`.

```
(expeditor-error-display s) → void?
  s : string?
```

Similar to `display` of *s*, but when color is enabled, the string is printed in the error color.