DrRacket Plugins

Version 9.4.0.3

Robert Bruce Findler

October 29, 2026

```
(require drracket/tool-lib)
 (require drracket/tool)
 (require drscheme/tool-lib)
 (require drscheme/tool)
```

package: `drracket-core-lib`

This manual describes DrRacket's plugins interface. It assumes familiarity with Racket, as described in the *The Racket Guide*, and the *The Racket Reference*, DrRacket, as described in *DrRacket: The Racket Programming Environment* and the GUI library, as described in *The Racket Graphical Interface Toolkit*. The Framework, as described in *Framework: Racket GUI Application Framework*, may also come in handy.

The `drscheme/tool-lib` and `drscheme/tool` libraries are for backward compatibility; they export all of the bindings of their `drracket` counterpart.

Contents

1 Tool support for `#lang`-based Languages **6**

1.1 Syntax Coloring . . . 7

1.2 Indentation . . . 8

1.3 Comments . . . 9

1.4 Keystrokes . . . 11

1.5 Filename Extensions . . . 12

1.6 REPL Submit Predicate . . . 12

1.7 Show Big “Definitions” and “Interactions” Labels . . . 13

1.8 Toolbar Buttons . . . 13

1.8.1 Opting Out of Standard Toolbar Buttons . . . 13

1.8.2 Opting In to Language-Specific Toolbar Buttons . . . 13

1.8.3 Adding New Toolbar Buttons . . . 14

1.9 Definition Popup-Menu Navigation . . . 14

1.10 Documentation Language Family . . . 16

1.11 Defaulting to Debugging or other Errortrace-based Annotations . . . 16

1.12 Definitions Text Surrogate . . . 17

2 Implementing DrRacket Plugins **18**

3 Adding Languages to DrRacket **23**

3.1 Adding Module-based Languages to DrRacket . . . 23

3.2 Adding Arbitrary Languages to DrRacket . . . 24

3.3 Language Extensions . . . 25

4 Creating New Kinds of DrRacket Frames **27**

5 Extending the Existing DrRacket Classes **28**

6 Expanding the User's Program Text and Breaking **29**

7 Editor Modes **30**

7.1 Color Schemes 30

7.2 General-purpose Modes 30

8 Plugin Capabilities **31**

9 Check Syntax **32**

9.1 Check Syntax Button 32

9.2 Syntax Properties that Check Syntax Looks For 32

10 Cooperating with Background Check Syntax **39**

11 Teaching Languages **40**

12 Signatures **41**

13 `drracket:get/extend` **42**

14 `drracket:unit` **46**

15 `drracket:language` **65**

16 `drracket:language-configuration` **94**

17 `drracket:debug` **97**

18 `drracket:rep` **104**

19 `drracket:frame` **113**

20 drracket:help-desk **118**

21 drracket:eval **119**

22 drracket:modes **124**

23 drracket:module-language-tools **126**

24 drracket:module-language **130**

25 drracket:tracing **132**

26 drracket:init **133**

27 Backwards Compatibility **134**

Index **162**

Index **162**

Thanks

Thanks to PLT and the early adopters of the tools interface for their feedback and help.

A special thanks to Eli Barzilay, John Clements, Matthias Felleisen, Cormac Flanagan, Matthew Flatt, Max Hailperin, Philippe Meunier, and Christian Queinnec for their help being early clients for DrRacket plugins.

1 Tool support for `#lang`-based Languages

A variety of tools can use extra information specified by a language. These tools include DrRacket, expeditor, and more.

The simplest and best way to extend tools to support a new language is to implement the language via `#lang` (see §17.3 "Defining new `#lang` Languages" for more details). Tools will then use `read-language` to find code and values that it uses to customize itself to the language.

If the call to `read-language` raises an error, DrRacket logs the error at the *`debug`* level to a logger with the name `'drracket-language` (see §15.5 "Logging" for more about how to follow specific loggers).

With the exception of the `'definitions-text-surrogate`, if there is an error during a use of one of these extensions, DrRacket notices the error and removes all of the extensions for the language. It also shows the error at the bottom of the DrRacket frame (prefixed by `#lang`). Note that this applies only to errors that occur during the dynamic extent of a use of one of these extensions. If an extension were to, for example, create a new thread that (eventually) caused an error, DrRacket would not notice that error and would not remove the extensions.

When experimenting with changes to these extensions, use the Racket|Reload #lang Extensions menu item to cause DrRacket to remove the extensions and reload the implementations from the files on disk.

DrRacket calls the language's `read-language`'s *`get-info`* procedure with the following *`key`* arguments. Other tools may use only a subset.

- `color-lexer`
- `drracket:indentation`
- `drracket:range-indentation`
- `drracket:paren-matches`
- `drracket:quote-matches`
- `drracket:comment-delimiters`
- `drracket:grouping-position`
- `drracket:default-filters`
- `drracket:default-extension`
- `drracket:keystrokes`

- drracket:show-big-defs/ints-labels
- drracket:opt-out-toolbar-buttons
- drracket:opt-in-toolbar-buttons
- drracket:submit-predicate
- drracket:toolbar-buttons
- drracket:define-popup
- documentation-language-family
- definitions-text-surrogate
- drracket:default-instrumentation

1.1 Syntax Coloring

When a language's *get-info* procedure responds to 'color-lexer, it is expected to return a procedure suitable to pass as the *get-token* argument to start-colorer.

The recognized token styles (specified implicitly via start-colorer's *token-sym->style* argument) are:

- 'symbol
- 'keyword
- 'comment
- 'string
- 'constant
- 'parenthesis
- 'error
- 'other

The precise colors for these identifiers are controlled by the preferences dialog in DrRacket, and by other customization mechanisms in other tools.

1.2 Indentation

When a language's *get-info* procedure responds to `'drracket:indentation`, it is expected to return a function with this contract:

```
(-> (is-a?/c color-textoid<%>)
    exact-nonnegative-integer?
    (or/c #f exact-nonnegative-integer?))
```

Although DrRacket might supply an object that implements `racket:text<%>`, if your language can limit itself to the smaller number of methods in the `color-textoid<%>` interface then it will work with more tools.

The function is used to indent lines. It is called with the position containing the line to be indented. It is expected to return the number of spaces that should appear at the beginning of the line or `#f`. If `#f` is returned, the tool uses the standard s-expression indentation rules.

Added in version 1.3 of package `drracket-core-lib`.

When a language's *get-info* procedure responds to `'drracket:range-indentation`, it is expected to return a function with this contract:

```
(-> (is-a?/c color-textoid<%>)
    exact-nonnegative-integer?
    exact-nonnegative-integer?
    (or/c #f (listof (list/c exact-nonnegative-
integer? string?))))
```

Although DrRacket might supply an object that implements `racket:text<%>`, if your language can limit itself to the smaller number of methods in the `color-textoid<%>` interface then it will work with more tools.

The function is used to indent a range that potentially spans multiple lines. It is called with the start and ending position of the range. The function is expected to return either `#f` or a list with an item for each line in the range. Returning `#f` falls back to iterating indentation over every line in the range (using `'drracket:indentation`, if available). Returning a list indicates an update for each corresponding line, where a line update takes the form `(list` *delete-amount insert-string*`)`: first delete *delete-amount* items from the start of the line, and then insert *insert-string* at the start of the line. If the returned list has fewer items then the range of lines to indent, the list is effectively padded with `(list 0 "")` no-op items. If the list has more items than the range of lines to indent, the extra items are ignored. Note that returning an empty list causes no lines to be updated, as opposed to returning `#f` to trigger a different indentation mechanism.

When both `'drracket:indentation` and `'drracket:range-indentation` are available, the function for `'drracket:range-indentation` is called first—except in the case of an implicit indentation from creating a newline, in which case only `'drracket:indentation` is used.

Added in version 1.10 of package `drracket-core-lib`.

When a language's *get-info* procedure responds to `'drracket:range-indentation/reverse-choices`, it is expected to return a procedure like one for `'drracket:range-indentation`, but if there are multiple indentation choices to cycle

through, then cycling should go through the choices in reverse order. When this function returns `#f`, then a non-reversed indentation is tried.

Added in version 1.16 of package `drracket-core-lib`.

When a language's *get-info* procedure responds to `'drracket:paren-matches`, it is expected to return a list of opening and closing parentheses, matching this contract:

```
(listof (list/c symbol? symbol?))
```

Each element of the outer list corresponds to a pair of parentheses, opening first and closing second.

These are used with the `framework` library's `color:text<%>` object; they are supplied as the *pairs* argument to the `start-colorer` method. The default value is

```
'((|(| |)|)
  (|[| |]|)
  (|{| |}|))
```

They are also used to introduce keybindings that match the parentheses, via `racket:map-pairs-keybinding-functions`.

Added in version 1.12 of package `drracket-core-lib`.

When a language's *get-info* procedure responds to `'drracket:quote-matches`, it is expected to return a list of characters that are self-matching, e.g. `"` in racket, matching this contract:

```
(listof char?)
```

These characters are used to introduce keybindings via `racket:map-pairs-keybinding-functions`, where the *open* and *close* arguments are both the character.

The default value is `(list #\" #\|)`.

Added in version 1.13 of package `drracket-core-lib`.

1.3 Comments

When a language's *get-info* procedure responds to `'drracket:comment-delimiters`, it is expected to return a value with this contract:

```
(listof
```

```
(or/c (list/c 'line
              (and/c string? (not/c #rx"[\r\n]"))
              (and/c string? (not/c #rx"[\r\n]")))
      (list/c 'region
              (and/c string? (not/c #rx"[\r\n]"))
              (and/c string? (not/c #rx"[\r\n]"))
              (and/c string? (not/c #rx"[\r\n]"))
              (and/c string? (not/c #rx"[\r\n]")))))
```

The value is a list of comment styles. Each comment style is expressed as one of:

- `(list 'line start padding)`, where *start* plus *padding* starts a comment that is teriminated by the end of a line.

  Lisp example: `'(line ";;" " ")`.

  C++ example: `'(line "//" " ")`.

- `(list 'region start continue end padding)`, where:

  - *start* then *padding* opens a comment
  - *continue* then `padding` is added to the beginning of each line except the first one when a comment spans multiple lines
  - *padding* then *end* closes a comment

  Racket example: `'(region "#|" "  " "|#" " ")`.

  C++ example: `'(region "/*" " *" "*/" " ")`.

When not specified by a lang, the default value is suitable for Racket s-expression langs:

```
'((line ";;" " ")
  (region "#|" "  " "|#" " "))
```

An intended use for these values is by (un)comment commands, which vary among tools. Some tools (un)comment entire lines, whereas others may handle portions of a line. Generally this is orthogonal to using a lang's line vs. region style: A tool can wrap entire lines using region comments. A tool can insert line breaks to make it possible to use line comments on a portion of a line. The point of `'drracket:comment-delimiters` is to enable a lang to tell a tool about its comment delimiters — not to say exactly how the (un)comment commands could or should work, exactly.

When the list has multiple styles: Some tools may present the styles for the user to pick one. Other tools may default to using the first style in the list (allowing the user to configure another preference by other means). Therefore when a language supports multiple comment styles, it should *list the most popular or preferred style first*.

Added in version 1.15 of package `drracket-core-lib`.

1.4 Keystrokes

When a language's *get-info* procedure responds to 'drracket:keystrokes, it is expected to return a list of keybindings and callbacks matching this contract:

```
(listof (list/c string?
                (-> (is-a?/c text%)
                    (is-a?/c event%)
                    any)))
```

Each element of the list is a different keybinding, where the string indicates the keystroke (see the documentation for map-function for the precise contents of the string and how it maps to particular keystrokes) and the procedure is called when the user types that keystroke in the definitions window.

The procedure's first argument will be the definitions text, the second will be the event object supplied from the GUI system and the result of the procedure is ignored.

When a language's *get-info* procedure responds to 'drracket:grouping-position, it is expected to return a function that determines where positions relevant to the nesting structure of the program appear. This function is used for a number of motion and selection operations in the editor, as well as determining where to flash to for closing parentheses.

Specifically the result must be a function matching this contract:

```
(-> (is-a?/c color-textoid<%>)
    natural?
    natural?
    (or/c 'up 'down 'backward 'forward)
    (or/c #f #t natural?))
```

Although DrRacket might supply an object that implements racket:text<%>, if your language can limit itself to the smaller number of methods in the color-textoid<%> interface then it will work with more tools.

Consider first the first and third argument. The first argument indicates a position in the editor to start from and the third argument is a direction to look. The result return the position for the corresponding direction, where the nesting structure of the program is viewed as a tree. That is, if the third argument is 'up, the function should return the position that goes up one layer in the tree from the given position to the parent. Similarly 'down should return the position going on layer deeper into that tree, going down to the first child. The 'backward and 'forward arguments correspond to position where we stay at the same level in the tree, moving between siblings. The result should be #f when there is no corresponding position to move to, e.g., when the current position has no children, no parents, or no siblings in the corresponding direction.

The second argument is a limit. Positions smaller than the limit should be ignored, so if the corresponding position appears to be before the limit, return #f.

Finally, return #t to get the default behavior, namely motion in Racket-style s-expressions.

Added in version 1.11 of package drracket-core-lib.

1.5 Filename Extensions

When a language's *get-info* procedure responds to 'drracket:default-filters, it is expected to return (listof (list/c string? string?)).

These results are added as a prefix to finder:default-filters, extending the default that DrRacket normally uses, namely:

```
`(["Racket Sources" "*.rkt;*.scrbl;*.rktl;*.rktd;*.ss;*.scm"]
  ["Any" "*.*"])
```

Added in version 1.2 of package drracket-core-lib.

When a language's *get-info* procedure responds to 'drracket:default-extension, it is expected to return (and/c string? (not/c #rx"[.]")); the result is used as the default extension when saving files by setting finder:default-extension.

Added in version 1.2 of package drracket-core-lib.

1.6 REPL Submit Predicate

When using the language declared in the source, DrRacket queries that language via read-language to determine if an expression in the interactions window is ready to be submitted to the evaluator (when the user types return). The info procedure is passed 'drracket:submit-predicate and should return a function matching this contract:

```
(-> input-port?
    boolean?
    boolean?)
```

This function's first argument is a port that contains the interactions window's data, starting just after the prompt and continuing to the end of the editor. The second argument is a boolean indicating if the insertion point is followed only by whitespace. The results should be a boolean indicating if the expression should be evaluated.

For backwards compatibility reasons, DrRacket also queries the result of module->language-info for 'drracket:submit-predicate. It does this during the evaluation of the definitions (so the Racket|Reload #lang extensions menu item does not trigger a re-load). If the submit predicate is specified both ways, then the predicate supplied via read-language takes precedence.

Changed in version 1.5 of package drracket-core-lib: Look for drracket:submit-predicate via read-language.

1.7 Show Big "Definitions" and "Interactions" Labels

If the read-language predicate returns #t for 'drracket:show-big-defs/ints-labels, then DrRacket shows the words "Definitions" and "Interactions" in a large font in the corresponding windows. This is intended as a help for students who are reading instructions about where to type their programs but might not have internalized this particular bit of DrRacket terminology.

1.8 Toolbar Buttons

1.8.1 Opting Out of Standard Toolbar Buttons

Some of the built-in buttons in the DrRacket button bar at the top of the window can be disabled on a per-language basis. DrRacket will invoke the *get-info* proc returned by read-language with 'drracket:opt-out-toolbar-buttons (and 'drscheme:opt-out-toolbar-buttons for backwards compatibility).

If the result is a list of symbols, the listed symbols are opted out. If the result is #f, all buttons are opted out. The default is the empty list, meaning that all opt-out buttons appear.

The Check Syntax button uses the symbol 'drracket:syncheck; the debugger uses the symbol 'debug-tool and the macro stepper uses 'macro-stepper.

Plugins may add more opt-out buttons via drracket:module-language-tools:add-opt-out-toolbar-button.

1.8.2 Opting In to Language-Specific Toolbar Buttons

Like drracket:opt-out-toolbar-buttons, but for languages to opt in to buttons that are not enabled by default.

Plugins may add more opt-out buttons via drracket:module-language-tools:add-opt-in-toolbar-button.

Added in version 1.6 of package drracket-core-lib.

1.8.3 Adding New Toolbar Buttons

DrRacket queries the result of `read-language` to determine if there are any new toolbar buttons to be used when editing files in this language.

Specifically, DrRacket will pass `'drracket:toolbar-buttons` to the function and expect back a value matching this contract:

```
(or/c (listof (list/c string?
                      (is-a?/c bitmap%)
                      (-> (is-a?/c drracket:unit:frame<%>) any)
                      (or/c real? #f)))
      #f)
```

which is then used to create new toolbar buttons, one for each element in the result list. The string is the label on the button; the bitmap is the icon (it should be 16x16 pixels); the function is called when the button is clicked; and the number is passed as the `#:number` argument to `register-toolbar-button`.

If the result is `#f`, then no toolbar buttons are created.

To implement functionality similar to the Run button, call the `execute-callback` method. You may also want to use the `drracket:rep:after-expression` parameter.

If `'drracket:toolbar-buttons` is not recognized, DrRacket will also pass `'drscheme:toolbar-buttons`; this is for backwards compatibility and new code should not use it. Similarly, if the fourth element from the list (the argument to `#:number`) is not present, then it is treated as `#f`.

1.9 Definition Popup-Menu Navigation

A popup menu in the DrRacket button bar jumps to definitions based on a heuristic search of the program text. DrRacket will invoke the *`get-info`* proc returned by `read-language` with `'drracket:define-popup` to obtain a configuration for the menu.

The value must satisfy the contract

```
(non-empty-listof (or/c (list/c string? string? string?)
                        (list/c string? string? string?
                                (or/c #f
                                      (-> (is-a/c text%)
                                          string?
                                          exact-integer?
                                          (->* ((is-a/c text%)
                                                string?
```

```
                                                  exact-integer?)
                                                 (#:case-
sensitive? any/c
                                                  #:delimited? any/c)
                                                 (or/c exact-
integer? #f))
                                             (or/c exact-
integer? #f)))
                                    (or/c #f
                                          (-> (is-a/c text%)
                                              exact-integer?
                                              (-> (is-a/c text%)
                                                  exact-integer?
                                                  string?)
                                              string?)))))
```

where the first string in each nested list is literal text to search for (outside of comments and literal strings), the second string is a label to describe the category of matches, and the third string is a short form of the label. The labels from the first nested list are used for the definition-popup button itself, while all labels are used for the user to select which categories are enabled.

When a nested list contains fourth and fifth elements, they can supply replacements (when not `#f`) for the default functions that find a prefix and extract the subsequent name:

- The prefix-finding function receives a text-editor object for the content to search, the prefix string to find, a position to start the search, and a default prefix-finding function. The result is a position in the text editor for the start of a found prefix, or `#f` if the prefix is not found.

  The provided default finding function accepts two optional keyword arguments: a true value for `#:case-sensitive?` requires case-insensitive matching, and a true value for `#:delimited?` indicates that the matched text's edges must coincide with forward and backward expression navigation.

- The name-extracting function receives a text-editor object for the content to extract, a position after a found prefix string, and a default name-extracting function. The result must be a string for the extracted defined name.

Plugins can provide a default popup-menu configuration via `drracket:language:register-capability` using `'drscheme:define-popup`.

Added in version 1.14 of package `drracket-core-lib`.

1.10 Documentation Language Family

When a language's *get-info* procedure (accessed via read-language) responds to the 'documentation-language-family key, the result is a string that names a language family. The language family should be defined as language-family in some collection's "info.rkt" file, and that definition configures a documentation entry point, a navigation configuration, and search precedence.

DrRacket uses the contents of this hash in three ways:

- When typing f1 (or right clicking) to search for documentation in DrRacket, DrRacket calls perform-search or send-language-family-page to summon browser-based documentation, and it passes along the language family.
- DrRacket uses the name in the first menu item in the Help menu item, replacing the word "Racket" with the string in the hash.
- When selecting that menu item, DrRacket visits the documentation page the language family by passing it along to send-language-family-page.

1.11 Defaulting to Debugging or other Errortrace-based Annotations

DrRacket uses the errortrace library to instrument programs before evaluating them, resulting in a tradeoff between feedback to the programmer and execution (and compile-time) cost. But default, each #lang-based language is run with errortrace's debugging annotations, meaning that error context information is relatively fine-grained. For some languages, however, this default isn't appropriate. By responding to the key drracket:default-instrumentation, a language can choose a different default. These are the options:

- 'none: no errortrace-based annotations
- 'debug: instruments the program to provide fine-grained stacktrace information
- 'debug/profile: instruments the program to provide fine-grained stacktrace information and profiling information
- 'test-coverage: instruments the program to provide fine-grained stacktrace information and test coverage information

Added in version 1.17 of package drracket-core-lib.

1.12 Definitions Text Surrogate

Using a `#lang`-specific definitions text surrogate is a very powerful way to flexibly control DrRacket's behavior when a new language is installed. It is also easy to cause DrRacket to completely misbehave with this form of extension. It is here only when one of the other forms of extension listed above are not sufficient for the kind of extension your language requires. And even in that case, it is preferable to add something to this list that is more easily controlled in the case of errors, using the definitions text surrogate only until that more easily controlled extension has been added to DrRacket.

DrRacket calls `read-language`'s `get-info` procedure with `'definitions-text-surrogate` and expects it to return a value matching the contract `(or/c #f module-path?)`, which is then passed to `dynamic-require` together with `'surrogate%`. The result is expected to be a class implementing the interface `racket:text-mode<%>` (presumably derived from `racket:text-mode%`. That mode is installed into the definitions text, where it can change its behavior by changing how is responds to any of the methods in the mode.

One consequence of this power is that errors that happen during the dynamic extent of calls into the mode are not trapped (much as errors that occur on newly created threads are not trapped, as described in the introduction to this section).

2 Implementing DrRacket Plugins

Plugins are designed for major extensions in DrRacket's functionality. To extend the appearance or the functionality the DrRacket window (say, to annotate programs in certain ways or to add buttons to the DrRacket frame) use a tool. The Macro Stepper, the Syntax Checker, the Stepper, and the teaching languages are all implemented as tools.

When DrRacket starts up, it looks for tools by reading fields in the `info.rkt` file of each collection and the newest version of each PLaneT package installed on the system. (Technically, DrRacket looks in a cache of the `"info.rkt"` files contents created by `raco setup`. Be sure to re-run `raco setup` if you change the contents of the `info.rkt` files). DrRacket checks for these fields:

- `drracket-tools`: `(listof (listof string [subcollection-name]))`
- `drracket-tool-names`: `(listof (or/c #f string))`
- `drracket-tool-icons`:

```
(listof (or/c #f
              string[relative-pathname]
              (cons string[filename]
                    (listof string[collection-name]))))
```

- `drracket-tool-urls`: `(listof (or/c #f string [url]))`

The `drracket-tools` field names a list of tools in this collection. Each tool is specified as a collection path, relative to the collection where the `info.rkt` file resides. As an example, if there is only one tool named `tool.rkt`, this suffices:

```
(define drracket-tools (list (list "tool.rkt")))
```

If the `drracket-tool-icons` or `drracket-tool-names` fields are present, they must be the same length as `drracket-tools`. The `drracket-tool-icons` field specifies the path to an icon for each tool and the name of each tool. If it is `#f`, no tool is shown. If it is a relative pathname, it must refer to a bitmap and if it is a list of strings, it is treated the same as the arguments to `lib`, inside `require`.

This bitmap and the name show up in the about box, the bug report form, and the splash screen as the tool is loaded at DrRacket's startup.

Each of the `drracket-tools` files must contain a module that `provides` `tool@`, which must be bound to a `unit`. The unit must import the `drracket:tool^` signature, which is provided by the `drracket/tool` library. The `drracket:tool^` signature contains all

of the names listed in this manual. The unit must export the drracket:tool-exports^ signature.

If the tool raises an error as it is loaded, invoked, or as the phase1 or phase2 thunks are called, DrRacket catches the error and displays a message box. Then, DrRacket continues to start up, without the tool.

For example, if the info.rkt file in a collection contains:

```
#lang info
(define drracket-tool-names (list "Tool Name"))
(define drracket-tools (list (list "tool.rkt")))
```

then the same collection would be expected to contain a tool.rkt file. It might contain something like this:

```
#lang racket/gui
(require drracket/tool)

(provide tool@)

(define tool@
  (unit
    (import drracket:tool^)
    (export drracket:tool-exports^)
    (define (phase1) (message-box "tool example" "phase1"))
    (define (phase2) (message-box "tool example" "phase2"))
    (message-box "tool example" "unit invoked")))
```

This tool just opens a few windows to indicate that it has been loaded and that the phase1 and phase2 functions have been called.

Finally, here is a more involved example. This module defines a plugin that adds a button to the DrRacket frame that, when clicked, reverses the contents of the definitions window. It also adds an easter egg. Whenever the definitions text is modified, it checks to see if the definitions text contains the text "egg". If so, it adds "easter " just before.

```
#lang racket/base
(require drracket/tool
         racket/class
         racket/gui/base
         racket/unit
         mrlib/switchable-button)
(provide tool@)

(define secret-key "egg")
```

```
(define to-insert "easter ")

(define tool@
  (unit
    (import drracket:tool^)
    (export drracket:tool-exports^)

    (define easter-egg-mixin
      (mixin ((class->interface text%)) ()

        (inherit begin-edit-sequence
                 end-edit-sequence
                 insert
                 get-text)

        (define/augment (on-insert start len)
          (begin-edit-sequence))
        (define/augment (after-insert start len)
          (check-range (max 0 (- start (string-length secret-
key)))
                       (+ start len))
          (end-edit-sequence))

        (define/augment (on-delete start len)
          (begin-edit-sequence))
        (define/augment (after-delete start len)
          (check-range (max 0 (- start (string-length secret-
key)))
                       start)
          (end-edit-sequence))

        (define/private (check-range start stop)
          (let/ec k
            (for ((x (in-range start stop)))
              (define after-x
                (get-text x (+ x (string-length secret-key))))
              (when (string=? after-x secret-key)
                (define before-x
                  (get-text (max 0 (- x (string-length to-
insert))) x))
                (unless (string=? before-x to-insert)
                  (insert to-insert x x)
                  (k (void)))))))

        (super-new)))
```

```
(define reverse-button-mixin
  (mixin (drracket:unit:frame<%>) ()
    (super-new)
    (inherit get-button-panel
             get-definitions-text)
    (inherit register-toolbar-button)

    (let ((btn
           (new switchable-button%
                (label "Reverse Definitions")
                (callback (λ (button)
                            (reverse-content
                             (get-definitions-text))))
                (parent (get-button-panel))
                (bitmap reverse-content-bitmap))))
      (register-toolbar-button btn #:number 11)
      (send (get-button-panel) change-children
            (λ (l)
              (cons btn (remq btn l)))))))

(define reverse-content-bitmap
  (let* ((bmp (make-bitmap 16 16))
         (bdc (make-object bitmap-dc% bmp)))
    (send bdc erase)
    (send bdc set-smoothing 'smoothed)
    (send bdc set-pen "black" 1 'transparent)
    (send bdc set-brush "blue" 'solid)
    (send bdc draw-ellipse 2 2 8 8)
    (send bdc set-brush "red" 'solid)
    (send bdc draw-ellipse 6 6 8 8)
    (send bdc set-bitmap #f)
    bmp))

(define (reverse-content text)
  (for ((x (in-range 1 (send text last-position))))
    (send text split-snip x))
  (define snips
    (let loop ((snip (send text find-first-snip)))
      (if snip
          (cons snip (loop (send snip next)))
          '())))
  (define released-snips
    (for/list ((snip (in-list snips))
               #:when (send snip release-from-owner))
      snip))
  (for ((x (in-list released-snips)))
```

```
    (send text insert x 0 0)))

(define (phase1) (void))
(define (phase2) (void))

(drracket:get/extend:extend-definitions-text easter-egg-mixin)
(drracket:get/extend:extend-unit-frame reverse-button-mixin)))
```

3 Adding Languages to DrRacket

3.1 Adding Module-based Languages to DrRacket

For backwards compatibility, DrRacket also supports and `info.rkt` file-based method for specifying such languages. Include these definitions:

- `drscheme-language-modules`: This must be bound to a list of collection path specifications or strings, one for each language in the collection. Each collection path specification is the quoted form of what might appear as an argument to `require`, using the `lib` argument (but without the `lib`). The strings represent relative paths starting at the directory containing the `info.rkt` file. They are interpreted like string arguments to `require`.

- `drscheme-language-positions`: This must be bound to a list of language positions. Each language position corresponds to the position of the language in language dialog. Each language position is a list of strings whose length must be at least two.

  If the first string is the same as `(string-constant teaching-languages)`, then it is put into the "Teaching Languages" section of the dialog. Otherwise, it goes into the "Other Languages" section of the dialog.

- `get-drscheme-language-positions`: This must be bound to a list that contains a module path followed by a symbol. The module path and symbol are combined with `dynamic-require` to obtain a list that is appended to the one from `drscheme-language-positions`, which allows access to `string-constants` to specify language positions.

- `drscheme-language-numbers`: This is optional. If present, it must be a list of a list of numbers. Each list corresponds to a single language from this collection. Each number indicates a sorting order in the language dialog for the corresponding string in `drscheme-language-positions`. If absent, it defaults to a list of zeros that has the same length as `drscheme-language-positions`. This will rarely be correct.

- `drscheme-language-one-line-summaries`: This is optional. If present, it must be a list of strings. Each string is displayed at the bottom of the language dialog when the corresponding language is selected.

- `drscheme-language-urls`: This is optional. If present, it must be a list whose elements are either strings or `#f`. Clicking the corresponding language's name in the interactions window opens a web browser to the url.

- `drscheme-language-readers`: This is optional. If present, it must be bound to a quoted list of module specifications (that is, a quoted version of the argument to `require`). Each specification must be a module that exports a function named `read-syntax`. Each of these `read-syntax` functions must match Racket's `read-syntax` primitive's contract, but may read different concrete syntax.

If the module specification is a plain string, it represents a relative path starting at the directory containing the `info.rkt` file. It is interpreted like the string arguments to `require`.

The lists must have the same length.

As an example, the *Essentials of Programming Languages* language specification's `info.rkt` used to look like this:

```
#lang info
(require string-constants)
(define name "EoPL Support")
(define drscheme-language-modules
  (list "eopl-lang.rkt"))
(define drscheme-language-positions
  (list (list (string-constant teaching-languages)
              "Essentials of Programming Languages")))
```

This `info.rkt` file indicates that there is a single language in this collection. The module that implements the language is the `eopl-lang.rkt` file in the same directory as the `info.rkt` file. Additionally, the language dialog will contain `Essentials of Programming Languages` as a potential language. The use of the string constant `teaching-languages` ensures that EoPL's language is placed properly in foreign language versions of DrRacket.

For collections that define multiple (related) languages, if the language-positions contain multiple strings, the languages whose leading strings match are grouped together. That is, if two languages have strings:

```
'("My Text" "First Language")
```

and

```
'("My Text" "Second Language")
```

the two languages will be grouped together in the language dialog.

3.2 Adding Arbitrary Languages to DrRacket

With some additional work, any language that can be compiled to Racket is supported by the tools interface, not just those that use standard configurations and `module`.

Each language is a class that implement the `drracket:language:language<%>` interface. DrRacket also provides two simpler interfaces: `drracket:language:module-based-language<%>` and `drracket:language:simple-module-based-language<%>`, and

mixins drracket:language:simple-module-based-language->module-based-language-mixin and drracket:language:module-based-language->language-mixin that build implementations of drracket:language:language<%>s from these simpler interfaces.

Once you have an implementation of the drracket:language:language<%> interface, call drracket:language-configuration:add-language to add the language to DrRacket.

Each language comes with its own type, called settings. This can be any type the language designer chooses, but to aid documentation, we call it settings here. The settings type is expected to contain parameters of the language, such as case sensitivity, etc. The implementor of the language provides a GUI so the user can configure the settings and all of the language's operations accept a setting. DrRacket maintains the current settings for each language.

3.3 Language Extensions

Some tools may require additional functionality from the drracket:language:language<%> interface. The drracket:language:extend-language-interface function and the drracket:language:get-default-mixin mixin make this possible.

For example, the MrFlow tool expands a program, analyzes it and then displays sets of values for each program point. These sets of values should be rendered in the syntax of the language that MrFlow analyzes. Since MrFlow doesn't know which languages are available, it can call drracket:language:extend-language-interface to extend the drracket:language:language<%> interface with a method for rendering sets of values and provide a default implementation of that method. Tools that know about MrFlow can then override the value rendering method to provide a language-specific implementation of value rendering. Additionally, since the drracket:language:get-default-mixin adds the default implementation for the value-set rendering method, all languages at least have some form of value-set rendering.

In some cases, it is important for one tool to avoid depending on another in the manner above. For example, if a tool that provides a new language provides an implementation for the MrFlow-specific method, that tool may fail to load if MrFlow is not present (Indeed, with the tool manager, this can happen to any tool that depends on another in this manner.)

To avoid this problem, consider writing your tool to first check to see if the base method is available before extending it. For example, if the MrFlow tool provides the render-value<%> interface, then a tool that overrides that method can first test to see if the superclass implements that method before overriding it:

```
(define (my-language-mixin %)
```

```
(if (implementation? % mrflow:render-value<%>)
    (class %
      (define/override ...)
      (super-new))
    %))
```

To help test your tool, use the `PLTONLYTOOL` environment variable to load it in isolation.

4 Creating New Kinds of DrRacket Frames

Each frame in DrRacket has certain menus and functionality, most of which is achieved by using the framework. Additionally, there is one mixin that DrRacket provides to augment that. It is `drracket:frame:basics-mixin`. Be sure to mix it into any new frame class that you add to DrRacket.

5 Extending the Existing DrRacket Classes

Each of the names:

- drracket:get/extend:extend-interactions-text
- drracket:get/extend:extend-definitions-text
- drracket:get/extend:extend-interactions-canvas
- drracket:get/extend:extend-definitions-canvas
- drracket:get/extend:extend-unit-frame
- drracket:get/extend:extend-tab

is bound to an extender function. In order to change the behavior of DrRacket, you can derive new classes from the standard classes for the frame, texts, canvases. Each extender accepts a function as input. The function it accepts must take a class as its argument and return a classes derived from that class as its result. For example:

```
(drracket:get/extend:extend-interactions-text
  (lambda (super%)
    (class super%
      (define/public (method1 x) ...)
      (super-new))))
```

extends the interactions text class with a method named `method1`.

6 Expanding the User's Program Text and Breaking

Macro-expanding a program may involve arbitrary computation and requires the setup of the correct language. To aid this, DrRacket's tool interface provides `drracket:eval:expand-program` to help. Use this method to extract the fully expanded program text in a particular language.

Because expanding the user's program may require DrRacket to evaluate arbitrary code that the user wrote, tools that expand the user's program should also allow the user to break the expansion. To help with this, the tools interfaces provides these methods: `enable-evaluation` and `disable-evaluation`. Since your tool will be expanding the program text, you should be both overriding `enable-evaluation` and `disable-evaluation` to disable your tool and calling them to ensure that only one expansion is happening at a time.

Finally, DrRacket provides the `set-breakables` method. This method controls what behavior the Break button has.

7 Editor Modes

7.1 Color Schemes

DrRacket uses the framework's color schemes to colorize source text and other aspects of itself. See `color-prefs:register-info-based-color-schemes` for details on how to add new color schemes via `"info.rkt"` files.

7.2 General-purpose Modes

Plugins can register modes via `drracket:modes:add-mode`. Each mode is visible in the Modes submenu of the Edit menu. Initially, DrRacket only supports two modes: Racket mode and text mode. (The Racket mode consults the language in the `#lang` line; see §1.12 "Definitions Text Surrogate" for more details.)

DrRacket automatically selects a mode for each open file based on the file's extension (and the language chosen as described above). If the file ends with `.txt`, DrRacket uses text mode. Otherwise, DrRacket uses Racket mode.

8 Plugin Capabilities

DrRacket's capability interface provides a mechanism for tools to allow languages to hide their GUI interface, if the tool does not apply to the language. Tools register capabilities keyed with symbols via. `drracket:language:register-capability`. Once registered, a tool can query a language, via the `capability-value` method. The result from this method controls whether or not the tool shows this part of the GUI for DrRacket.

See `drracket:language:register-capability` for a list of the capabilities registered by default.

9 Check Syntax

Check Syntax is a part of the DrRacket collection, but is implemented via the plugin API. See also drracket/check-syntax.

9.1 Check Syntax Button

```
(require drracket/syncheck-drracket-button)
                        package: drracket-core-lib
```

```
syncheck-drracket-button : (list/c
                            string?
                            (is-a?/c bitmap%)
                            (-> (is-a?/c
                                 top-level-window<%>)
                                any))
```

This is meant to be used with the 'drracket:toolbar-buttons argument to the info proc returned from read-language.

```
syncheck:button-callback
```

This is defined with define-local-member-name and is bound to a method of no arguments of the DrRacket frame that runs Check Syntax.

```
syncheck-bitmap : (is-a?/c bitmap%)
```

The bitmap in the Check Syntax button on the DrRacket frame.

9.2 Syntax Properties that Check Syntax Looks For

Check Syntax collects the values of the syntax-propertys named 'disappeared-use, 'disappeared-binding, 'original-for-check-syntax, 'identifiers-as-disappeared-uses?, 'identifier-as-keyword, 'sub-range-binders, 'sub-range-binding, and 'mouse-over-tooltips and uses them to add control which arrows are added to the program text. These properties are intended for use when a macro discards or manufactures identifiers that, from the programmers perspective, should be binding each other, or when there are identifiers that are intended to be used more in the spirit of keywords, and thus should be ignored.

- Check Syntax draws arrows only between identifiers that are free-identifier=?. They must be syntax-original? or have the syntax-property 'original-for-check-syntax set to #t. See also current-recorded-disappeared-uses.

- The properties `'disappeared-use`, `'disappeared-binding`, and allow macro authors to inform Check Syntax that an identifier (or what appears to be an identifier to the programmer) does not appear in the output of the macro, but should still have binding arrows.

  For example, here is program with a macro that discards its arguments, but adds properties to the result syntax object so that the two occurrences of `a` are treated as a binding/bound pair by Check Syntax.

```
(define-syntax (m stx)
  (syntax-case stx ()
    [(_ id1 id2)
     (and (identifier? #'id1) (identifier? #'id2))
     (syntax-property
      (syntax-property
       #'1
       'disappeared-use (list (syntax-local-
introduce #'id1)))
      'disappeared-binding (list (syntax-local-
introduce #'id2)))]))

(m a a)
```

  Another approach for situations where identifiers are discarded by a macro is to introduce a `let` expression that doesn't contribute to the result of the computation, but does signal to Check Syntax that there are some arrows to draw. For example, Check Syntax is unable to draw the arrows between the introductions and uses of `a`, `b`, and `c` for this code:

```
#lang racket
(require (for-syntax syntax/parse))

(define-syntax (depths stx)
  (syntax-parse stx
    [(_ id expr)
     (define table (make-hash))
     (let loop ([stx #'expr] [depth 0])
       (cond
         [(syntax->list stx)
          =>
          (λ (lst)
            (for ([ele (in-list lst)])
              (loop ele (+ depth 1))))]
         [(identifier? stx)
          (hash-set! table (syntax-e stx) depth)]))
     #`(define-syntax id #,table)]))
```

```
(define-syntax (depth-of stx)
  (syntax-parse stx
    [(_ id1 id2)
     (datum->syntax
      #'here
      (hash-ref (syntax-local-value #'id1)
                (syntax-e #'id2)))]))

(depths my-sexp ((a) b (((((((c)))))))))

(depth-of my-sexp a)
(depth-of my-sexp b)
(depth-of my-sexp c)
```

Extending these macro to cooperate with Check syntax requires more information to be collected on the definition side and then used at the use side:

```
#lang racket
(require (for-syntax syntax/parse))

(define-syntax (depths stx)
  (syntax-parse stx
    [(_ id expr)
     (define table (make-hash))
     (let loop ([stx #'expr] [depth 0])
       (cond
         [(syntax->list stx)
          =>
          (λ (lst)
            (for ([ele (in-list lst)])
              (loop ele (+ depth 1))))]
         [(identifier? stx)
          (hash-set! table (syntax-e stx)
                     (cons (vector (syntax-source stx)
                                   (syntax-line stx)
                                   (syntax-column stx)
                                   (syntax-position stx)
                                   (syntax-span stx))
                           depth))]))
     #`(define-syntax id #,table)]))

(define-syntax (depth-of stx)
  (syntax-parse stx
    [(_ id1 id2)
     (define pr
       (hash-ref (syntax-local-value #'id1)
```

```
                (syntax-e #'id2)))
      (define fake-binder
        (datum->syntax #'id2 (syntax-e #'id2) (car pr) #'id2))
      #`(begin
          (void (let ([#,fake-binder 1]) id2))
          #,(cdr pr))]))

(depths my-sexp ((a) b (((((((c)))))))))

(depth-of my-sexp a)
(depth-of my-sexp b)
(depth-of my-sexp c)
```

- For each syntax object that appears in the fully expanded program, DrRacket traverses it looking for identifiers and connecting them to likely binding occurrences. When it finds such identifiers it draws an arrow with a large question mark near the head of the arrow. But, if the syntax object has the property `'identifiers-as-disappeared-uses?`, then the arrows are the normal arrow color.

- The value of the `'sub-range-binders` property is expected to be a tree of `cons` pairs (in any configuration) whose leaves are either ignored or are vectors with either of these shapes:

```
(or/c (vector/c identifier? natural? natural?
                identifier? natural? natural?)

      (vector/c identifier?
                natural? natural?
                (real-in 0 1) (real-in 0 1)

                identifier?
                natural? natural?
                (real-in 0 1) (real-in 0 1)))
```

  Each vector is interpreted as a single arrow. The first identifier in the vector is the start of the arrow and the second identifier in the vector is the destination of the arrow. The two natural numbers that follow each identifier adjust the precise starting and ending ranges for the arrows, however. They are interpreted as offsets into the position of each corresponding identifier, making the arrows start and end on just a portion of the identifier, instead of the entire identifier.

  If the vector has 8 elements, then the two real numbers are treated as the precise location where the arrow starts and ends, inside the rectangle that corresponds to the start and end of the identifier. The first real number is for the x direction and the second one is for the y direction. For example, if some identifier has a position and span of 100 and 10, and the offset are 1 and 5, then the rectangle that bounds the corresponding end of the arrow would be from position 101 to 105. This entire range gets highlighted

when the mouse moves over it. The arrow itself, however, will start from some specific point inside that editor range, normally in the center and corresponds to the situation where the two real numbers are both `0.5`. If, however the two reals are both `1/3`, then the arrow will start one third of the way from the top to the bottom and one third of the way from the left to the right.

The property is looked for in expression positions and on binding identifiers.

Here's an example:

```
#lang racket/base
(require (for-syntax racket/base))
(define-syntax (define/hyphen stx)
  (syntax-case stx ()
    [(_ id1 id2 rhs-expr)
     (let ()
       (define first-part (symbol->string (syntax-e #'id1)))
       (define second-part (symbol->string (syntax-e #'id2)))
       (define first-len (string-length first-part))
       (define second-len (string-length second-part))
       (define hyphenated-id
         (datum->syntax
          #'id1
          (string->symbol (string-append first-part "-
" second-part))))
       (syntax-property
        #`(define #,hyphenated-id rhs-expr)
        'sub-range-binders
        (list
         (vector (syntax-local-introduce hyphenated-id)
                 0 first-len 0.5 0.5
                 (syntax-local-introduce #'id1)
                 0 first-len 0.5 0.5)
         (vector (syntax-local-introduce hyphenated-id)
                 (+ first-len 1) second-len 0.5 0
                 (syntax-local-introduce #'id2)
                 0 second-len 0.5 1)))))]))

(define/hyphen big generator
  11)

(+ big-generator big-generator)
```

After putting this code in the DrRacket window, mouse over the words "big" and "generator" to see arrows pointing to the individual pieces of the identifier *big-generator*. The four `0.5`s in the first vector put the arrows on *big* in the center of the identifiers; the `0.5 0` and the `0.5 1` in the second vector put the arrows at the top and bottom center for *generator*.

- The `'sub-range-binding` property is expected to match the contract

```
(or/c (vector/c natural? natural?)
      (vector/c natural? natural?
                (real-in 0 1) (real-in 0 1)))
```

  and, if it is attached to an identifier then Check Syntax will draw only one of the arrows that `'sub-range-binding` would otherwise indicate should be drawn. Specifically, if the identifier matches the subrange binder (e.g, it is `big-generator` in the previous example) and the `'sub-range-binding` property were attached with the vector containing the number 4 (for the start) and 9 (for the span) then Check Syntax will draw only an arrow to the `generator` portion of the identifier. Additionally, the arrow will go to the first 9 characters of the reference, according to its source location.

- The value of the `'mouse-over-tooltips` property is expected to be to be a tree of `cons` pairs (in any configuration) whose leaves are either ignored or are vectors of the shape

```
(vector/c syntax?
          exact-nonnegative-integer?
          exact-nonnegative-integer?
          (or/c string? (-> string?)))
```

  Each vector's content indicates where to show a tooltip. The first three components are a syntax object whose `syntax-source` field indicates which file the tooltip goes in, the start and end position in the editor where mouseovers will show the tooltip, and the content of the tooltip. Note that editor positions count from zero, while syntax object positions count from one, so use `sub1` to convert between them. If the tooltip content is a procedure, this procedure is called by Check Syntax to compute the string used for the tooltip, as Check Syntax traverses the syntax objects looking for properties.

  For example, here's a macro that shows the span of itself in a tooltip on mouseover:

```
#lang racket
(define-syntax (char-span stx)
  (syntax-case stx ()
    [(_ a)
     (syntax-property
      #'a
      'mouse-over-tooltips
      (vector
       stx
       (sub1 (syntax-position stx))
       (sub1 (+ (syntax-position stx)
                (syntax-span stx)))
       (format "this expression\nspans ~a chars"
               (syntax-span stx))))]))

(char-span (+ 1 2))
```

- If the syntax property `'identifier-as-keyword` is any value except `#f` and appears on an identifier, then Check Syntax ignores the identifier, not drawing any arrows to it.

Changed in version 1.3 of package `drracket-core-lib`: Looks for `'sub-range-binders` on binding identifiers (not just in expression positions).
Changed in version 1.5: Looks for `'identifier-as-keyword` on identifiers.

10 Cooperating with Background Check Syntax

DrRacket's continuous, background check syntax runs each time an edit to the definitions text happens. In some cases, that expansion process fails, but there is still a well-formed syntax object that check syntax can use to display information to the user. In order to communicate that syntax object to check syntax, send a log message with the name `'online-check-syntax`, e.g.

```
(define-logger online-check-syntax)
(log-message online-check-syntax-logger
             'info
             "ignored message"
             list-of-syntax-objects)
```

The fourth argument to `log-message` should be a list of syntax objects; these are processed as if they were the result of expansion.

The syntax objects whose `syntax-source` field does not match the source of the file that is currently being expanded are ignored. That is, sometimes a macro may log a syntax object to be used by DrRacket in this fashion, but the macro may not be from the file that DrRacket's expanding, but one from one that is required by it; hence this check is in place to skip them.

Note: the identifiers in these objects should be `syntax-original?` or else they will be ignored by check syntax.

11 Teaching Languages

The teaching language are implemented via the tools interface and thus not part of DrRacket proper, but one helper library is documented here.

```
(require lang/htdp-langs-save-file-prefix)
```
package: htdp-lib

```
htdp-save-file-prefix : (listof string?)
```

These strings are used as the prefix in a file saved while using the teaching languages. Each string is on a separate line in the saved file.

```
(htdp-file-prefix? ip) → boolean?
  ip : input-port?
```

Determines if the contents of *ip* is one of the possible prefixes that DrRacket saves at the beginning of a teaching language file.

In the case that this function returns `#t`, it consumes the entire prefix from *ip* (and discards it). In the case that this function returns `#f`, it does not consume anything from *ip*.

12 Signatures

drracket:tool^ : signature

This signature includes all of the names in this manual that begin with drracket: (except these two signatures).

drracket:tool-exports^ : signature

The drracket:tool-exports^ signature contains two names: phase1 and phase2. After all of the tools are loaded, all of the phase1 functions are called and then all of the phase2 functions are called. Certain primitives can only be called during the dynamic extent of those calls.

This mechanism is designed to support DrRacket's drracket:language:language<%> extension capabilities. That is, this mechanism enables two tools to cooperate via new capabilities of languages. The first phase is used for adding functionality that each language must support and the second is used for creating instances of languages. As an example, a tool may require certain specialized language-specific information. It uses phase1 to extend the drracket:language:language<%> interface and supply a default implementation of the interface extension. Then, other languages that are aware of the extension can supply non-default implementations of the additional functionality.

(phase1) → void?

These functions can be called only in the dynamic extent of a call to phase1 (see above for details).

- drracket:language:extend-language-interface
- drracket:unit:add-to-program-editor-mixin

(phase2) → void?

These functions can be called only in the dynamic extent of a call to phase2 (see above for details).

- drracket:language-configuration:add-language
- drracket:language:get-default-mixin
- drracket:language:get-language-extensions

13 drracket:get/extend

```
(drracket:get/extend:extend-unit-frame
 mixin
 [before
  #:name-for-changes name-for-changes])
→ void?
 mixin : (make-mixin-contract drracket:unit:frame%)
 before : boolean? = #t
 name-for-changes : (or/c #f symbol?) = #f
```

Extends the class that is used for the frame that implements the main DrRacket window.

The *before* argument controls if the mixin is applied before or after already installed mixins.

If *name-for-changes* is a symbol and drracket:get/extend:allow-re-extension! has been called (without a subsequent call to drracket:get/extend:disallow-re-extension!) then calling this function replaces any earlier mixins that have been added that have the same name. Otherwise, calling this with the same name twice is an error and calling it once drracket:get/extend:get-frame has been called is an error.

```
(drracket:get/extend:get-unit-frame)
 → (subclass?/c drracket:unit:frame%)
```

Returns a class whose objects are used for the DrRacket frames.

Once this function is called, drracket:get/extend:extend-unit-frame raises an error, disallowing any more extensions.

See also drracket:get/extend:allow-re-extension!.

```
(drracket:get/extend:extend-tab
 mixin
 [before
  #:name-for-changes name-for-changes])
→ void?
 mixin : (make-mixin-contract drracket:unit:tab<%>)
 before : boolean? = #t
 name-for-changes : (or/c #f symbol?) = #f
```

Like drracket:get/extend:extend-unit-frame, except it extends the class that implements the tabs in DrRacket. One is created for each tab in a frame (each frame always has at least one tab, even if the tab bar is not shown).

```
(drracket:get/extend:get-tab)
 → (implementation?/c drracket:unit:tab<%>)
```

Like drracket:get/extend:get-unit-frame, except it returns the class used for tabs.

```
(drracket:get/extend:extend-definitions-text
 mixin
 [before
  #:name-for-changes name-for-changes])
→ void?
 mixin : (make-mixin-contract drracket:unit:definitions-text<%>
                              editor:standard-style-list<%>
                              editor:info<%>
                              racket:text<%>
                              text:all-string-snips<%>
                              text:file<%>
                              text:info<%>
                              text:wide-snip<%>)
 before : boolean? = #t
 name-for-changes : (or/c #f symbol?) = #f
```

Like drracket:get/extend:extend-unit-frame, except this text is used in the top window of DrRacket frames.

```
(drracket:get/extend:get-definitions-text)
 → (and/c (implementation?/c drracket:unit:definitions-text<%>)
          (implementation?/c editor:standard-style-list<%>)
          (implementation?/c editor:info<%>)
          (implementation?/c racket:text<%>)
          (implementation?/c text:all-string-snips<%>)
          (implementation?/c text:file<%>)
          (implementation?/c text:info<%>)
          (implementation?/c text:wide-snip<%>))
```

Like drracket:get/extend:get-unit-frame, except for the text that is used in the top window of DrRacket frames.

```
(drracket:get/extend:extend-interactions-text
 mixin
 [before
  #:name-for-changes name-for-changes])
→ void?
```

```
  mixin : (make-mixin-contract drracket:rep:text<%>)
  before : boolean? = #t
  name-for-changes : (or/c #f symbol?) = #f
```

Like drracket:get/extend:extend-unit-frame, except it extends the class that implements the the editor in the interactions window.

```
(drracket:get/extend:get-interactions-text)
 → (implementation?/c drracket:rep:text<%>)
```

Like drracket:get/extend:get-unit-frame except it returns the class that implements the editor in the interactions window.

```
 (drracket:get/extend:extend-definitions-canvas
  mixin
 [before
  #:name-for-changes name-for-changes])
 → void?
  mixin : (make-mixin-contract drracket:unit:definitions-canvas%)
  before : boolean? = #t
  name-for-changes : (or/c #f symbol?) = #f
```

Like drracket:get/extend:extend-unit-frame, except it extends the class that implements the definitions window's editor-canvas%.

```
(drracket:get/extend:get-definitions-canvas)
 → (subclass?/c drracket:unit:definitions-canvas%)
```

Like drracket:get/extend:get-unit-frame except it returns the class that implements the definitions window's editor-canvas%.

```
 (drracket:get/extend:extend-interactions-canvas
  mixin
 [before
  #:name-for-changes name-for-changes])
 → void?
  mixin : (make-mixin-contract drracket:unit:interactions-canvas%)
  before : boolean? = #t
  name-for-changes : (or/c #f symbol?) = #f
```

Like drracket:get/extend:extend-unit-frame, except it extends the class that implements the interactions window's editor-canvas%.

```
(drracket:get/extend:get-interactions-canvas)
 → (subclass?/c drracket:unit:interactions-canvas%)
```

Like drracket:get/extend:get-unit-frame except it returns the class that implements the definitions window's editor-canvas%.

```
(drracket:get/extend:disallow-re-extension!) → void?
```

Once this is called, re-extension of the mixins described in this section is not allowed. This is the default state of mixin extension, but it can be changed by drracket:get/extend:allow-re-extension!.

```
(drracket:get/extend:allow-re-extension!) → void?
```

Once this is called, re-extension of the mixins described in this section are now allowed (see drracket:get/extend:extend-unit-frame for details of how to effect a re-extension).

This mode is intended to support a faster development cycle, not for production code. Specifically, the issue is that replacing mixins in this manner does not affect any objects that have already been create and thus there can, in general, be a mixture of old and new objects in a single DrRacket. If some kind of systematic change to the classes is wanted, consider instead using the racket/surrogate library.

Once an extension happens, newly created objects will use the new mixins. Mostly, however, creating a new frame will create a new set of all of the objects that are extended in this section, so that can be used to experiment more quickly with changes.

14 drracket:unit

```
drracket:unit:tab<%> : interface?
  implements: drracket:rep:context<%>
```

```
(send a-drracket:unit:tab break-callback) → void?
```

Specification: This method is called when the break button is clicked and this tab is the active tab.

Default implementation: By default, breaks any evaluation that may be happening at this point.

```
(send a-drracket:unit:tab can-close?) → boolean?
```

Refine this method with `augment`.

Specification: This method is called to determine if it is okay to close this tab.

Default implementation: Calls the definitions text's and interactions text's `can-close?` method.

```
(send a-drracket:unit:tab disable-evaluation) → void?
```

Overrides `disable-evaluation` in `drracket:rep:context<%>`.

Disables the Run button, and the Run menu item and `lock`s the interactions window, and the definitions window.

```
(send a-drracket:unit:tab enable-evaluation) → void?
```

Overrides `enable-evaluation` in `drracket:rep:context<%>`.

Enables the Run button, and the Run menu item and unlocks (via the `lock` method) the interactions window and the definitions window.

```
(send a-drracket:unit:tab get-breakables)
 → (or/c thread? false/c)
   (or/c custodian? false/c)
```

Overrides `get-breakables` in `drracket:rep:context<%>`.

```
(send a-drracket:unit:tab get-defs)
 → (is-a?/c drracket:unit:definitions-text<%>)
```

This text is initially the top half of the DrRacket window and contains the users program.

This text defaults to a `text%` object, but if you change `drracket:get/extend:extend-definitions-text` procedure, it will use the extended class to create the text.

```
(send a-drracket:unit:tab get-directory)
 → (or/c string? false/c)
```

Overrides `get-directory` in `drracket:rep:context<%>`.

This is the directory that the file is saved in, or the directory DrRacket started up in, if the file has not been saved.

```
(send a-drracket:unit:tab get-enabled) → boolean?
```

Indicates if evaluation is currently enabled in this tab. Evaluation is typically disabled when some evaluation is already running (in another thread).

```
(send a-drracket:unit:tab get-frame)
 → (is-a?/c drracket:unit:frame%)
```

Returns the frame that this tab is inside.

```
(send a-drracket:unit:tab get-ints)
 → (is-a?/c drracket:rep:text%)
```

This text is initially the bottom half of the DrRacket window and contains the users interactions with the REPL.

This text defaults to a `drracket:rep:text%` object, but if you use the `drracket:get/extend:extend-interactions-text` procedure, it will use the extended class to create the text.

```
(send a-drracket:unit:tab is-current-tab?) → boolean?
```

Indicates if this tab is the currently active tab.

```
(send a-drracket:unit:tab is-running?) → boolean?
```

Indicates if the running message in the bottom right of DrRacket's frame should be "running" or "not running" when this frame is active.

```
(send a-drracket:unit:tab on-close) → void?
```

Refine this method with `augment`.

Specification: This method is called when the tab is closed.

Default implementation: Calls the definitions text's `on-close` and interactions text's `on-close` methods.

```
(send a-drracket:unit:tab reset-offer-kill) → void?
```

Overrides reset-offer-kill in drracket:rep:context<%>.

```
(send a-drracket:unit:tab set-breakables thread
                                          custodian) → void?
  thread : (or/c thread? false/c)
  custodian : (or/c custodian? false/c)
```

Overrides set-breakables in drracket:rep:context<%>.

```
(send a-drracket:unit:tab add-bkg-running-color id
                                                color
                                                label) → void?
  id : symbol?
  color : (or/c string? (is-a?/c color%))
  label : string?
```

This method is final, so it cannot be overridden.

Sets the color of the circle in the bottom-right corner of the DrRacket window to *color* with the tooltip window that appears over it containing *label*. If multiple colors are registered they are all shown.

See also remove-bkg-running-color.

```
(send a-drracket:unit:tab remove-bkg-running-
color id) → void?
  id : symbol?
```

This method is final, so it cannot be overridden.

Removes the color and label added with *id*.

See also add-bkg-running-color.

```
(send a-drracket:unit:tab touched) → void?
```

This method is final, so it cannot be overridden.

Called by the system to indicate that the tab has just been switched to from another tab in the same frame (when the frame has the focus) or the frame itself has come to the front (via on-activate) and the tab is the current tab in that frame.

This method updates the private state that get-last-touched returns.

```
(send a-drracket:unit:tab get-last-touched) → flonum?
```

This method is final, so it cannot be overridden.

Returns the time that this tab was last focused, as counted by current-inexact-milliseconds.

```
drracket:unit:tab% : class?
  superclass: object%
  extends: drracket:unit:tab<%>
```

The base class that implements the tab's functionality.

```
(make-object drracket:unit:tab%)
 → (is-a?/c drracket:unit:tab%)
```

```
(send a-drracket:unit:tab clear-annotations) → void?
```

Overrides `clear-annotations` in `drracket:rep:context<%>`.

Clears any error highlighting; calls `clear-annotations`.

```
drracket:unit:program-editor-mixin : (class? . -> . class?)
  argument extends/implements: text%
                               editor:basic<%>
```

This mixes in the ability to reset the highlighting for error message when the user modifies the buffer. Use it for editors that have program text where errors can occur.

```
(send a-drracket:unit:program-editor after-delete start
                                                   len)
 → void?
  start : number
  len : number
```

Augments `after-delete` in `text%`.

Calls the inner method.

Resets an error highlighting.

```
(send a-drracket:unit:program-editor after-insert start
                                                   len)
 → void?
  start : number
  len : number
```

Augments `after-insert` in `text%`.

Calls the inner method.

Resets an error highlighting.

```
drracket:unit:interactions-canvas% : class?
  superclass: canvas:wide-snip%
```

```
(new drracket:unit:interactions-canvas% ...superclass-
args...)
 → (is-a?/c drracket:unit:interactions-canvas%)
```

Passes all arguments to `super-init`.

```
drracket:unit:frame% : class?
  superclass: (drracket:frame:basics-mixin (drracket:frame:mixin frame:searchable%))
  extends: drracket:unit:frame<%>
```

This frame inserts the Racket and Language menus into the menu bar as it is initialized.

```
(new drracket:unit:frame% ...superclass-args...)
 → (is-a?/c drracket:unit:frame%)
```

Passes all arguments to `super-init`.

```
(send a-drracket:unit:frame add-show-menu-items show-menu)
 → void?
  show-menu : (is-a?/c menu%)
```

Overrides `add-show-menu-items` in `drracket:frame:<%>`.

Adds the "Show Definitions", "Show Interactions" and "Show Contour" menu items.

```
(send a-drracket:unit:frame break-callback) → void?
```

Specification: This method is called when the user clicks on the break button or chooses the break menu item.

Default implementation: Breaks the user's evaluation started by the Run button (or possibly a queued callback in the user's eventspace).

```
(send a-drracket:unit:frame change-to-file file) → void?
  file : string?
```

Loads this file into this already created frame. In normal DrRacket use, this method is only called if this is the first frame opened and no editing has occurred. It should be safe to call this at anytime, however.

```
(send a-drracket:unit:frame find-matching-tab p)
 → (or/c (is-a?/c drracket:unit:tab%) #f)
  p : path-string?
```

Returns the tab that is currently editing `p`, if there is one in this frame. Returns `#f` otherwise.

```
(send a-drracket:unit:frame change-to-tab tab) → void?
  tab : (is-a?/c drracket:unit:tab%)
```

Makes `tab` visible in this frame.

```
(send a-drracket:unit:frame edit-menu:between-select-all-
and-find edit-menu)
 → void?
  edit-menu : (is-a?/c menu%)
```

Overrides `edit-menu:between-select-all-and-find` in `frame:standard-menus<%>`.

Adds the `"Split"` and `"Collapse"` menu items.

```
(send a-drracket:unit:frame execute-callback) → void?
```

Specification: This method is called when the user clicks on the Run button or chooses the Run menu item.

Default implementation: It calls `ensure-rep-shown` and then it calls `evaluate-from-port` passing in the result of `get-interactions-text` and its entire range, unless the first two characters are `#!` in which case, it skips the first line.

```
(send a-drracket:unit:frame file-menu:between-open-and-
revert file-menu)
 → void?
  file-menu : (is-a?/c menu%)
```

Overrides `file-menu:between-open-and-revert` in `drracket:frame:basics-mixin`.

Calls the super method and adds a `separator-menu-item%` to the menu.

```
(send a-drracket:unit:frame file-menu:between-print-and-
close file-menu)
 → void?
  file-menu : (is-a?/c menu%)
```

Overrides `file-menu:between-print-and-close` in `drracket:frame:basics-mixin`.

Adds a menu item for printing the interactions.

```
(send a-drracket:unit:frame file-menu:between-save-as-and-
print file-menu)
 → void?
  file-menu : (is-a?/c menu%)
```

Overrides `file-menu:between-save-as-and-print` in `frame:standard-menus<%>`.

Adds a submenu that contains various save options:

- save definitions as text
- save interactions
- save interactions as
- save interactions as text

and adds a separator item.

```
(send a-drracket:unit:frame file-menu:print-string) → void?
```

Overrides `file-menu:print-string` in `frame:standard-menus<%>`.

returns `"Definitions"`

```
(send a-drracket:unit:frame file-menu:save-as-
string) → void?
```

Overrides `file-menu:save-as-string` in `frame:standard-menus<%>`.

Returns `"Definitions"`.

```
(send a-drracket:unit:frame file-menu:save-string) → void?
```

Overrides `file-menu:save-string` in `frame:standard-menus<%>`.

Returns `"Definitions"`.

```
(send a-drracket:unit:frame get-break-button)
 → (is-a?/c button%)
```

Returns the break button. Mostly used for test suites.

```
(send a-drracket:unit:frame get-button-panel)
 → (is-a?/c horizontal-panel%)
```

This panel goes along the top of the DrRacket window and has buttons for important actions the user frequently executes.

A tool can add a button to this panel to make some new functionality easily accessible to the user.

See also mrlib's `switchable-button%`.

```
(send a-drracket:unit:frame get-canvas)
 → (is-a?/c editor-canvas%)
```

Overrides `get-canvas` in `frame:editor<%>`.

Returns the result of `get-definitions-canvas`.

```
(send a-drracket:unit:frame get-canvas%) → (is-
a?/c canvas%)
```

Overrides `get-canvas%` in `frame:editor<%>`.

Returns the result of `drracket:get/extend:get-definitions-canvas`.

```
(send a-drracket:unit:frame get-definitions/interactions-
panel-parent)
 → (is-a?/c vertical-panel%)
(send a-drracket:unit:frame get-definitions/interactions-panel-parent)
 → void?
```

Specification: This method is provided so that tools can add `area-container<%>`s to the DrRacket frame. Override this method so that it returns a child of the super-classes's result and insert new children in between.

Default implementation: First case:

Returns the result of `get-area-container`

Second case:

```
(send a-drracket:unit:frame get-editor) → (is-
a?/c editor<%>)
```

Overrides `get-editor` in `frame:editor<%>`.

Returns the result of `get-definitions-text`.

```
(send a-drracket:unit:frame get-editor%) → (is-
a?/c editor<%>)
```

Overrides `get-editor%` in `frame:editor<%>`.

Returns the result of `drracket:get/extend:get-definitions-text`.

```
(send a-drracket:unit:frame get-execute-button)
 → (is-a?/c button%)
```

Returns the Run button. Mostly used for test suites.

```
(send a-drracket:unit:frame get-text-to-search)
 → (is-a?/c text:searching%)
```

Overrides get-text-to-search in frame:searchable-text-mixin.

returns the text that is active in the last canvas passed to make-searchable

```
(send a-drracket:unit:frame make-searchable canvas) → void?
  canvas : (is-a?/c drracket:unit:interactions-canvas%)
```

stores the canvas, until get-text-to-search is called.

```
(send a-drracket:unit:frame on-close) → void?
```

Augments on-close in frame:standard-menus<%>.

Sends the result of get-interactions-text the shutdown and on-close methods.

Always calls the inner method.

```
(send a-drracket:unit:frame on-size) → void?
```

Overrides on-size in window<%>.

Updates the preferences for the window width and height so next time a DrRacket window is opened, it will be this width and height.

```
(send a-drracket:unit:frame still-untouched?) → boolean?
```

Specification: determines if the definitions window has not been modified. Used in conjunction with change-to-file.

Default implementation: Returns #t if the buffer is empty, it has not been saved and it is unmodified.

```
(send a-drracket:unit:frame update-save-button modified?)
 → void?
  modified? : any/c
```

This method hides or shows the save button, based on the *modified?* argument.

If the save button has not been created yet, it remembers the *modified?* argument as an initial visibility for the save button.

This method is called by the set-modified method.

```
(send a-drracket:unit:frame update-save-
message name) → void?
  name : string?
```

Updates the save message on the DrRacket frame. This method is called by the set-filename method.

```
(send a-drracket:unit:frame update-shown) → void?
```

Overrides update-shown in drracket:frame:<%>.

Updates the interactions, definitions, and contour menu items based on the contents of the windows.

```
drracket:unit:frame<%> : interface?
  implements: drracket:frame:<%>
```

```
(send a-drracket:unit:frame get-language-menu)
 → (is-a?/c menu%)
```

Returns the language-specific menu. This menu is called the Racket menu in the Racket language but is, in general, controlled by the 'drscheme:language-menu-title capability (see drracket:language:register-capability for details on capabilities).

```
(send a-drracket:unit:frame ensure-defs-shown) → void?
```

Ensures that the definitions window is visible.

```
(send a-drracket:unit:frame ensure-rep-hidden) → void?
```

Makes sure the rep is hidden (by making the definitions window visible).

```
(send a-drracket:unit:frame ensure-rep-shown rep) → void?
  rep : (is-a?/c drracket:rep:text<%>)
```

This method is called to force the rep window to be visible when, for example, an error message is put into the rep. Also ensures that the appropriate tab is visible, if necessary.

```
(send a-drracket:unit:frame get-current-tab)
 → (is-a?/c drracket:unit:tab<%>)
```

Returns the currently active tab.

```
(send a-drracket:unit:frame get-tab-filename i) → string?
  i : (<=/c 0 (get-tab-count))
```

Returns a string naming the file in the ith tab or, if the file is not saved, something like “Untitled”.

```
(send a-drracket:unit:frame get-tab-count)
 → exact-positive-integer?
```

Returns the number of open tabs in the frame.

```
(send a-drracket:unit:frame open-in-new-tab
 filename
 [#:start-pos start-pos
  #:end-pos end-pos])
→ void?
 filename : (or/c path-string? #f)
 start-pos : exact-nonnegative-integer? = 0
 end-pos : (or/c exact-nonnegative-integer? 'same) = 'same
```

Opens a new tab in this frame. If *filename* is a path-string?, load that file in the definitions window of the new tab. If *start-pos* and *end-pos* are provided, call the tab's definition's window's set-position with them.

```
(send a-drracket:unit:frame create-new-tab
 [filename
  #:start-pos start-pos
  #:end-pos end-pos])
→ void?
 filename : (or/c path-string? #f) = #f
 start-pos : exact-nonnegative-integer? = 0
 end-pos : (or/c exact-nonnegative-integer? 'same) = 'same
```

Creates a new tab. If *filename* is not #f, behaves like open-in-new-tab.

```
(send a-drracket:unit:frame after-create-new-tab tab
                                                 filename
                                                 start-pos
                                                 end-pos)
→ void?
 tab : (is-a?/c drracket:unit:tab<%>)
 filename : (or/c path-string? #f)
 start-pos : exact-nonnegative-integer?
 end-pos : (or/c exact-nonnegative-integer? 'same)
```

Refine this method with augment.

Specification: Called after a tab is created, possibly with a file loaded.

Default implementation: Does nothing.

```
(send a-drracket:unit:frame reopen-closed-tab) → void?
```

Opens the most recently closed tabs.

```
(send a-drracket:unit:frame next-tab) → void?
```

Switches to the next tab.

```
(send a-drracket:unit:frame prev-tab) → void?
```

Switches to the previous tab.

```
(send a-drracket:unit:frame move-current-tab-right) → void?
```

Swaps the current tab with its right-hand neighbor.

```
(send a-drracket:unit:frame move-current-tab-left) → void?
```

Swaps the current tab with its left-hand neighbor.

```
(send a-drracket:unit:frame reorder-tabs tab-
order) → void?
  tab-order : (listof exact-nonnegative-integer?)
```

Reorders the tabs according to *tab-order*.

Each element in *tab-order* identifies a tab by its position in the list `get-tabs`, and the position of this element identifies the new position of the tab.

For example, considering that there are only 3 tabs open, `(send a-drracket-frame reorder-tabs '(2 1 0))` swaps the first and last tabs, leaving the middle one unchanged.

```
(send a-drracket:unit:frame close-current-tab) → void?
```

This method is final, so it cannot be overridden.

Closes the current tab, making some other tab visible. If there is only one tab open, this method does nothing.

```
(send a-drracket:unit:frame close-ith-tab i) → void?
  i : natural?
```

This method is final, so it cannot be overridden.

Closes the tab located at position *i* in the list returned by `get-tabs`. If there is only one tab open, this method does nothing.

Added in version 1.9 of package `drracket-core-lib`.

```
(send a-drracket:unit:frame close-given-tab tab) → void?
  tab : (is-a?/c drracket:unit:tab<%>)
```

This method is final, so it cannot be overridden.

Closes *tab*. If *tab* is the only open tab, this method does nothing.

Added in version 1.9 of package `drracket-core-lib`.

```
(send a-drracket:unit:frame get-definitions-canvas)
 → (is-a?/c drracket:unit:definitions-canvas%)
```

This canvas is the canvas containing the `get-definitions-text`. It is initially the top half of the DrRacket window.

This canvas defaults to a `drracket:unit:definitions-canvas%` object, but if you change the `drracket:get/extend:extend-definitions-canvas` procedure, it will use the class in the parameter to create the canvas.

```
(send a-drracket:unit:frame get-definitions-text)
 → (is-a?/c drracket:unit:definitions-text%)
```

Calls result of `get-current-tab`'s `get-defs` method.

```
(send a-drracket:unit:frame get-insert-menu) → (is-
a?/c menu%)
```

Specification: Returns the Insert menu.

```
(send a-drracket:unit:frame get-interactions-canvas)
 → (instanceof (derivedfrom drracket:unit:interactions-canvas%))
```

This canvas is the canvas containing the `get-interactions-text`. It is initially the bottom half of the DrRacket window.

This canvas defaults to a `drracket:unit:interactions-canvas%` object, but if you use the `drracket:get/extend:extend-interactions-canvas` procedure, it will use the extended class to create the canvas.

```
(send a-drracket:unit:frame get-interactions-text)
 → (is-a?/c drracket:rep:text%)
```

Calls result of `get-current-tab`'s `get-ints` method.

```
(send a-drracket:unit:frame get-tabs)
 → (listof (is-a?/c drracket:unit:tab<%>))
```

Returns the list of tabs in this frame.

```
(send a-drracket:unit:frame on-tab-change from-tab
                                          to-tab)  → void?
  from-tab : (is-a?/c drracket:unit:tab<%>)
  to-tab : (is-a?/c drracket:unit:tab<%>)
```

Refine this method with `augment`.

Specification: Called after a new tab becomes the selected tab in the frame.

Default implementation: The *from-tab* argument is the previously selected tab, and the *to-tab* argument is the newly selected tab.

```
(send a-drracket:unit:frame register-capability-menu-item
 key
 menu)
→ void?
 key : symbol
 menu : (is-a? menu%)
```

Registers the menu item that was most recently added as being controlled by the capability *key*. This means that the (boolean) value of the capability determines if the menu item is present in the menu (the capability is checked when the menus are clicked on).

This assumes that the menu items in this menu are not moved around, except by the this capability. If they are, things can go funny (i.e., no good checks are in place).

Note that the capability must be registered separately, via `drracket:language:register-capability`.

```
(send a-drracket:unit:frame register-toolbar-button
  tb
 [#:number num])
→ void?
 tb : (is-a?/c switchable-button%)
 num : (or/c #f real?) = #f
```

Registers the toolbar button *tb*.

The *num* argument controls the ordering of *tb* with respect to other toolbar buttons. If it is `#f`, then a number one smaller than the currently smallest number is used.

The buttons are sorted by their numbers, from left to right in horizontal mode and from top to bottom in vertical mode. If buttons are in sub-panels they cannot, in general, be sorted entirely by number without changing the panel structure, but when a sub-panel appears as a sibling of some toolbar buttons, the sorting routine looks for the smallest number appearing in a button in the sub-panel, and uses that number when sorting the panel that appears with the buttons.

A number of buttons already come with numbers: the Stop button's number is `101`, the Run button's number is `100`, the Scribble PDF button's number is `99`, the Scribble HTML button's number is `98`, the Macro Stepper button's number is `70`, the Debug button's number is `60`, the Stepper button's number is `59`, and the Check Syntax button's number is `50`.

All three are children of the panel returned by get-button-panel.

Registration is required so that the toolbar buttons properly switch orientation when the toolbar's position is moved and the ordering via the number argument is preserved. See also sort-toolbar-buttons-panel.

```
(send a-drracket:unit:frame register-toolbar-buttons
  tbs
 [#:numbers nums])
 → void?
  tbs : (listof (is-a?/c switchable-button%))
  nums : (listof (or/c real? #f)) = (make-list (length tbs) #f)
```

Simultaneously registers the toolbar buttons *tbs*.

See also register-toolbar-button.

```
(send a-drracket:unit:frame unregister-toolbar-button tb)
 → void?
  tb : (is-a?/c switchable-button%)
```

Unregisters the toolbar button *tb*. Use this method to ensure that the button is not referenced by this frame and thus can be gc'd.

```
(send a-drracket:unit:frame sort-toolbar-buttons-panel)
 → void?
```

Sorts the children of get-button-panel, according to the number argument passed to register-toolbar-button.

```
drracket:unit:definitions-text<%> : interface?
```

This interface is implemented by the definitions text.

```
(send a-drracket:unit:definitions-text after-set-next-
settings language-settings)
 → void?
  language-settings : language-settings
```

Refine this method with augment.

Specification: Called when the next settings changes. See also get-next-settings.

Default implementation:

```
(send a-drracket:unit:definitions-text begin-metadata-
changes)
 → void?
```

Augment this method to be notified when DrRacket is changing the buffer to insert metadata. The metadata is only inserted during saving, so tools that track changes to DrRacket will need to ignore changes that occur after this method is called, and before `end-metadata-changes` is called.

A call to `begin-metadata-changes` will always be followed with a call to `end-metadata-changes` (ie, the calls cannot be nested).

```
(send a-drracket:unit:definitions-text end-metadata-changes)
 → void?
```

Called when the changes to insert metadata are done, and the editor is back to its state at the time of the call to `begin-metadata-changes`.

A call to `begin-metadata-changes` will always be followed with a call to `end-metadata-changes` (ie, the calls cannot be nested).

```
(send a-drracket:unit:definitions-text get-next-settings)
 → language-settings
```

This method returns the language-settings that will be used when the user next clicks Run in this DrRacket window.

```
(send a-drracket:unit:definitions-text get-tab)
 → (is-a?/c drracket:unit:tab%)
```

Returns the editor's enclosing tab.

```
(send a-drracket:unit:definitions-text set-needs-execution-
message msg)
 → void?
  msg : string?
```

Specification: This method, when called, puts this DrRacket window in a state such that interactions submitted to the REPL will trigger a yellow warning message. The state is reset when the program is next Run.

Default implementation: Records *msg* and uses it the next time the user submits an interaction (unless the Runs first).

```
 (send a-drracket:unit:definitions-text set-next-settings
  language-settings
  [update-prefs?])
 → void?
  language-settings : language-settings
  update-prefs? : any/c = #t
```

Changes the language settings for this window. If *update-prefs?* is a true value, the preference is changed, which affects newly created windows.

See also after-set-next-settings and get-next-settings.

```
(send a-drracket:unit:definitions-text set-filename
  filename
 [temporary?])
 → void?
  filename : (or/c path-string? #f)
  temporary? : any/c = #f
```

Overrides <method not found>.

The class that is the result of (get-drracket:unit:definitions-text%) overrides this method and calls update-save-message.

```
(send a-drracket:unit:definitions-text set-
modified modified?)
 → void?
  modified? : any/c
```

Overrides <method not found>.

The class that is the result of (get-drracket:unit:definitions-text%) overrides this method and calls update-save-button.

```
drracket:unit:definitions-canvas% : class?
  superclass: editor-canvas%
```

Initializes the visibility of the save button.

```
(drracket:unit:get-definitions-text%)
 → (and/c (instanceof/c racket:text<%>)
          (instanceof/c text:info%)
          (implementation?/c drracket:unit:definitions-text<%>))
```

Returns the class used to implement the definitions text in the DrRacket frame. Its result mixes in drracket:unit:program-editor-mixin.

```
(drracket:unit:get-program-editor-mixin)
 → ((subclass?/c text%) . -> . (subclass?/c text%))
```

Returns a mixin that must be mixed in to any text% object that might contain program text (and thus can be in the source field of some syntax object).

See also drracket:unit:add-to-program-editor-mixin.

```
(drracket:unit:add-to-program-editor-mixin mixin) → void?
  mixin : ((subclass?/c text%) . -> . (subclass?/c text%))
```

This function can only be called in phase 1 (see §2 “Implementing DrRacket Plugins” for details).

Adds *mixin* to the result of drracket:unit:get-program-editor-mixin.

```
(drracket:unit:open-drscheme-window [filename
                                     #:show? show?])
 → (is-a?/c drracket:unit:frame%)
  filename : (or/c string? #f) = #f
  show? : boolean? = #t
```

Opens a DrRacket frame that displays *filename*, or, if *filename* is #f, an empty file.

If *show?* is #t, then the show is not invoked before the function returns; otherwise it is.

```
(drracket:unit:add-search-help-desk-menu-item  text
                                               menu
                                               position
                                              [add-sep]) → void?
  text : (is-a?/c text%)
  menu : (is-a?/c menu-item-container<%>)
  position : exact-nonnegative-integer?
  add-sep : (-> any) = void
```

Adds a menu item to *menu* that searches in Help Desk for the word around *position* in *text*.

If there is only whitespace around *position*, then no menu-item%s are added, and *add-sep* is not called. If there is something to be added, then *add-sep* is called before the menu item is created.

```
(struct drracket:unit:teachpack-callbacks (get-names
                                           add
                                           remove
                                           remove-all)
```

```
  #:extra-constructor-name
  make-drracket:unit:teachpack-callbacks)
get-names : (-> any/c (listof string?))
add : (-> any/c path-string? any/c)
remove : (-> path-string? any/c any/c)
remove-all : (-> any/c any/c)
```

Holds callbacks for teachpack operations. DrRacket invokes these functions in response to GUI operations being triggered.

Each of the `any/c`s that appear in the field contracts are actually the settings of a language.

The `get-names` field returns the names of the teachpacks in the given settings; `add` returns a new settings that includes the `path-string?` argument as a new teachpack; `remove` removes the given teachpack and `remove-all` removes them all.

```
drracket:unit:struct:teachpack-callbacks : struct-type?
```

This is an alias for `struct:drracket:unit:teachpack-callbacks`.

```
drracket:unit:make-teachpack-callbacks : procedure?
```

This is an alias for `make-drracket:unit:teachpack-callbacks`.

```
(drracket:unit:find-symbol text pos) → string?
  text : (is-a?/c text%)
  pos : exact-nonnegative-integer?
```

returns a string that corresponds to the a symbol surrounding *pos* (in *text*).

This is intended to be used with the "f1" keybinding for searching in the documentation, so the result is not always a symbol, but instead a best effort to find something that is likely to be useful to search for around a point in the *text*.

15 drracket:language

drracket:language:simple-module-based-language<%> : interface?

This interface represents the bare essentials when defining a module-based language. Use the drracket:language:simple-module-based-language->module-based-language-mixin mixin to construct an implementation of drracket:language:module-based-language<%> from an implementation of this interface.

The class drracket:language:simple-module-based-language% provides an implementation of this interface.

(send *a-drracket:language:simple-module-based-language* get-language-numbers)
 → (cons/c number? (listof number?))

Returns a list of numbers, whose length must be the same as the result of get-language-position. Each number indicates the sorted order of the language positions in the language dialog.

(send *a-drracket:language:simple-module-based-language* get-language-position)
 → (cons/c string? (listof string?))

This method is the same as get-language-position.

(send *a-drracket:language:simple-module-based-language* get-module)
 → s-expression

This method specifies the module that defines the language.

This method replaces front-end/complete-program and front-end/interaction.

The result is expected to be the `module` (its initial require) except as value, ie `quoted`.

(send *a-drracket:language:simple-module-based-language* get-one-line-summary)
 → (or/c #f string?)

The result of this method is shown in a tooltip in the language dialog when the user mouses over this language. If the result is #f, no tooltip is shown.

```
(send a-drracket:language:simple-module-based-language get-
reader)
 → (->* () (any/c input-port?) (or/c syntax? eof-object?))
```

This method must return a procedure that is used to read syntax from a port in the same manner as `read-syntax`. It is used as the reader for this language.

```
drracket:language:simple-module-based-language% : class?
  superclass: object%
  extends: drracket:language:simple-module-based-language<%>
```

```
 (make-object drracket:language:simple-module-based-language%
  module
  language-position
 [language-numbers
  one-line-summary
  documentation-reference]
  reader
  language-id)
 → (is-a?/c drracket:language:simple-module-based-language%)
 module : s-expression
 language-position : (cons/c string? (listof string?))
 language-numbers : (cons/c number? (listof number?))
                  = (map (lambda (x) 0) language-position)
 one-line-summary : string? = ""
 documentation-reference : (or/c #f something-else) = #f
 reader : (->* () (any/c input-port?) (or/c syntax? eof-object?))
 language-id : string?
```

The init args are used as the results of the `get-module` and `get-language-position` methods.

```
(send a-drracket:language:simple-module-based-language get-
language-numbers)
 → (cons/c number? (listof number?))
```

Overrides `get-language-numbers` in `drracket:language:simple-module-based-language<%>`.

returns the corresponding init arg.

```
(send a-drracket:language:simple-module-based-language get-
language-position)
 → s-expression
```

Overrides get-language-position in drracket:language:simple-module-based-language<%>.

returns the corresponding init arg.

```
(send a-drracket:language:simple-module-based-language get-module)
 → (cons/c string? (listof string?))
```

Overrides get-module in drracket:language:simple-module-based-language<%>.

returns the corresponding init arg.

```
(send a-drracket:language:simple-module-based-language get-one-line-summary)
 → string?
```

Overrides get-one-line-summary in drracket:language:simple-module-based-language<%>.

returns the corresponding initialization argument.

```
(send a-drracket:language:simple-module-based-language get-reader)
 → (->* () (any/c input-port?) (or/c syntax? eof-object?))
```

Overrides get-reader in drracket:language:simple-module-based-language<%>.

returns the corresponding init arg.

```
drracket:language:simple-module-based-language->module-based-language-mixin : (class? . -> . class?)
  argument extends/implements: drracket:language:simple-module-based-language<%>
  result implements: drracket:language:module-based-language<%>
```

This mixin uses a struct definition for its settings:

```
(define-struct drracket:language:simple-settings
  (case-sensitive  ; boolean?
   printing-style  ; (or/c 'constructor 'quasiquote 'write 'print)
   fraction-style  ; (or/c 'mixed-fraction 'mixed-fraction-e
                   ;  'repeating-decimal 'repeating-decimal-e)
   show-sharing    ; boolean?
   insert-newlines ; boolean?
   annotations))   ; (or/c 'none 'debug 'debug/profile
                   ;  'test-coverage)
```

The settings in this structure reflect the settings show in the language configuration dialog for languages constructed with this mixin. The first controls the input for the language. The rest specify printing controls for the language. The style `'print` is the default style, as normally used in the Racket REPL. The sharing field determines if cycles and sharing in values are displayed when the value is rendered. The insert newlines field determines if values in the repl are formatted with `write` style-line printouts, or with `pretty-print` multi-line printouts.

```
(send a-drracket:language:simple-module-based-language-
>module-based-language config-panel)
 → (case-> (-> settings) (settings -> void?))
```

Constructs a configuration panel that lets the user configure all of the settings for this language.

See also `drracket:language:simple-module-based-language->module-based-language-mixin` for details of the simple-settings structure, this mixin's `settings` type.

```
(send a-drracket:language:simple-module-based-language-
>module-based-language default-settings)
 → settings
```

The defaults for the settings are

- `case-sensitive` is `#f`
- `printing-style` is `'write`
- `show-sharing` is `#f`
- `insert-newlines` is `#t`

See also `drracket:language:simple-module-based-language->module-based-language-mixin` for details of the simple-settings structure, this mixins `settings` type.

```
(send a-drracket:language:simple-module-based-language-
>module-based-language default-settings?)
 → boolean?
```

```
(send a-drracket:language:simple-module-based-language-
>module-based-language get-init-code settings)
 → sexpression
  settings : settings
```

Creates an s-expression of a module that sets the `current-inspector`, `read-case-sensitive`, and `error-value->string` parameters. Additionally, it may load `errortrace`, if debugging is enabled.

```
(send a-drracket:language:simple-module-based-language-
>module-based-language get-transformer-module)
 → s-expression
```

Returns `'mzscheme`.

```
(send a-drracket:language:simple-module-based-language-
>module-based-language marshall-settings)
 → writable
```

Constructs a vector from the structure.

See also `drracket:language:simple-module-based-language->module-based-language-mixin` for details of the simple-settings structure, this mixins `settings` type.

```
(send a-drracket:language:simple-module-based-language-
>module-based-language on-execute)
 → void?
```

Sets the case sensitivity of the language.

Sets the structure inspector to a new inspector, saving the original inspector for use during printing.

Sets the `global-port-print-handler` to print based on the settings structure, but without any newlines.

If debugging is enabled, it sets the `current-eval` handler to one that annotates each evaluated program with debugging annotations. Additionally, it sets the `error-display-handler` to show the debugging annotations when an error is raised.

See also `drracket:language:simple-module-based-language->module-based-language-mixin` for details of the simple-settings structure, this mixin's `settings` type.

```
(send a-drracket:language:simple-module-based-language-
>module-based-language render-value)
 → void?
```

Translates the value to a string, based on the settings.

Restores a super struct inspector to render structs properly. (See also on-execute)

See also `drracket:language:simple-module-based-language->module-based-language-mixin` for details of the simple-settings structure, this mixin's `settings` type.

```
(send a-drracket:language:simple-module-based-language-
>module-based-language render-value/format)
 → void?
```

Translates the value to a string, based on the settings.

Restores a super struct inspector to render structs properly. (See also on-execute.)

See also drracket:language:simple-module-based-language-
>module-based-language-mixin for details of the simple-settings structure, this mixin's settings type.

```
(send a-drracket:language:simple-module-based-language-
>module-based-language unmarshall-settings)
 → (or/c #f settings)
```

Builds a settings structure from the vector, or #f if the vector doesn't match the types of the structure.

See also drracket:language:simple-module-based-language-
>module-based-language-mixin for details of the simple-settings structure, this mixin's settings type.

```
(send a-drracket:language:simple-module-based-language-
>module-based-language use-mred-launcher)
 → boolean?
```

Returns #t.

```
drracket:language:module-based-language<%> : interface?
```

This interface is for languages that can be implemented with Racket modules.

Use the drracket:language:module-based-language->language-mixin mixin to construct an implementation of drracket:language:language<%> from an implementation of this interface.

```
(send a-drracket:language:module-based-language config-
panel parent)
 → (case-> (-> settings) (settings -> void?))
  parent : (is-a?/c panel%)
```

This method is the same as config-panel.

```
(send a-drracket:language:module-based-language default-
settings)
 → settings
```

This method is the same as `default-settings`.

```
(send a-drracket:language:module-based-language default-
settings? settings)
 → boolean?
  settings : settings
```

This method is the same as `default-settings?`.

```
(send a-drracket:language:module-based-language get-init-
code settings)
 → sexp
  settings : settings
```

Returns a module in sexpression form that is used for creating executables. The module must provide a thunk, called `init-code`.

When either a stand-alone executable or a launcher is created, the module is required, and `init-code` is invoked. This procedure is expected to set up the environment, based on the settings.

```
(send a-drracket:language:module-based-language get-
language-numbers)
 → (cons/c number? (listof number?))
```

This method is the same as `get-language-numbers`.

```
(send a-drracket:language:module-based-language get-
language-position)
 → (cons/c string? (listof string?))
```

This method is the same as `get-language-position`.

```
(send a-drracket:language:module-based-language get-module)
 → s-expression
```

This method specifies the module that defines the language. It is used to initialize the user's namespace.

The result is expected to be the `module` (its initial require) except as value, ie `quoted`.

See also `get-transformer-module`.

```
(send a-drracket:language:module-based-language get-one-
line-summary)
 → string?
```

The result of this method is shown in the language dialog when the user selects this language.

```
(send a-drracket:language:module-based-language get-reader)
 → (->* () (any/c input-port?) (or/c syntax? eof-object?))
```

This method must return a procedure that is used to read syntax from a port in the same manner as `read-syntax`. It is used as the reader for this language.

```
(send a-drracket:language:module-based-language get-
transformer-module)
 → (or/c quoted-module-path #f)
```

This method specifies the module that defines the transformation language. It is used to initialize the transformer portion of the user's namespace.

The result is expected to be the `module` (its initial require) except as value, i.e., `quoted` or `#f`.

If the result is `#f`, no module is required into the transformer part of the namespace.

See also `get-module`.

```
(send a-drracket:language:module-based-language marshall-
settings settings)
 → writable
  settings : settings
```

This method is the same as `marshall-settings`.

```
 (send a-drracket:language:module-based-language on-execute
  settings
  run-on-user-thread)
 → void?
  settings : settings
  run-on-user-thread : ((-> void?) -> void?)
```

This method is the same as `on-execute`.

```
 (send a-drracket:language:module-based-language render-value
  value
  settings
  port)
```

```
 → void?
  value : TST
  settings : settings
  port : port?
```

This method is the same as render-value.

```
(send a-drracket:language:module-based-language render-value/format
  value
  settings
  port
  width)
 → void?
  value : TST
  settings : settings
  port : port?
  width : (or/c number? 'infinity)
```

This method is the same as render-value/format.

```
(send a-drracket:language:module-based-language unmarshall-
settings input)
 → (or/c settings #f)
  input : writable
```

This method is the same as unmarshall-settings.

```
(send a-drracket:language:module-based-language use-mred-
launcher)
 → boolean?
```

This method is called when an executable is created to determine if the executable should use the GRacket or the Racket binary.

```
(send a-drracket:language:module-based-language use-
namespace-require/copy?)
 → boolean?
```

Specification: The result of this method controls how the module is attached to the user's namespace. If the method returns #t, the Racket primitive namespace-require/copy is used and if it returns #f, namespace-require is used. *Default implementation:* Returns #f by default.

```
drracket:language:module-based-language->language-mixin : (class? . ->
. class?)
  argument extends/implements: drracket:language:module-based-language<%>
  result implements: drracket:language:language<%>
```

```
(send a-drracket:language:module-based-language-
>language front-end/complete-program)
 → (-> (or/c sexp/c syntax? eof-object?))
```

Reads a syntax object, from `input`. Does not use `settings`.

For languages that use these mixins, there is no difference between this method and `front-end/interaction`.

```
(send a-drracket:language:module-based-language-
>language front-end/interaction)
 → (-> (or/c sexp/c syntax? eof-object?))
```

Reads a syntax object, from `input`. Does not use `settings`.

For languages that use these mixins, there is no difference between this method and `front-end/complete-program`.

```
(send a-drracket:language:module-based-language-
>language get-language-name)
 → string?
```

Returns the last element of the list returned by `get-language-position`.

```
(send a-drracket:language:module-based-language-
>language on-execute)
 → void?
```

Overrides `on-execute` in `drracket:language:module-based-language<%>`.

Calls the super method.

Uses `namespace-require` to install the result of `get-module` and Uses `namespace-require` combined with `for-syntax` to install the result of `get-transformer-module` into the user's namespace.

```
drracket:language:language<%> : interface?
```

Implementations of this interface are languages that DrRacket supports.

See §3 "Adding Languages to DrRacket" for an overview of adding languages to DrRacket.

```
(send a-drracket:language:language capability-
value key) → any
  key : symbol?
```

Specification: Returns the language-specific value for some capability. See also drracket:language:register-capability. *Default implementation:* By default, returns the value from: drracket:language:get-capability-default.

```
(send a-drracket:language:language config-panel parent)
 → (case-> (-> settings) (settings -> void?))
  parent : (is-a?/c panel%)
```

This method used by the language configuration dialog to construct the "details" panel for this language. It accepts a parent panel and returns a get/set function that either updates the GUI to the argument or returns the settings for the current GUI.

```
(send a-drracket:language:language create-executable
 settings
 parent
 program-filename)
 → void?
 settings : settings
 parent : (or/c (is-a?/c dialog%) (is-a?/c frame%))
 program-filename : string?
```

This method creates an executable in the given language. The *program-filename* is the name of the program to store in the executable and executable-filename is the name of a file where the executable goes.

See also drracket:language:create-module-based-stand-alone-executable and drracket:language:create-module-based-launcher.

```
(send a-drracket:language:language default-settings)
 → settings
```

Specifies the default settings for this language.

```
(send a-drracket:language:language default-
settings? settings)
 → boolean?
  settings : settings
```

Return #t if the input settings matches the default settings obtained via default-settings.

```
(send a-drracket:language:language first-opened settings)
 → void?
  settings : settings
```

This method is called after the language is initialized, but no program has yet been run. It is called from the user's eventspace's main thread.

See also initialize-console.

Calling this method should not escape. DrRacket calls this method in a parameterize where the error-escape-handler is set to an escaping continuation that continues initializing the interactions window. Thus, raising an exception will report the error in the user's interactions window as if this were a bug in the user's program. Escaping in any other way, however, can cause DrRacket to fail to start up.

Also, IO system will deadlock if the first-opened method does IO on the user's IO ports, so the calling context of first-opened sets the current-output-port and current-error-port to ports that just collect all of the IO that happened and then replay it later in the initialization of the user's program.

Contrary to the method contract spec, DrRacket will also invoke this method if it has zero arguments, passing nothing; the zero argument version is for backwards compatibility and is not recommended.

```
(send a-drracket:language:language front-end/complete-program
 port
 settings)
→ (-> (or/c sexp/c syntax? eof-object?))
 port : port?
 settings : settings
```

front-end/complete-program method reads and parses a program in the language. The *port* argument contains all of the data to be read (until eof) and the name of the *port* (obtained via object-name) is a value representing the source of the program (typically an editor, but may also be a string naming a file or some other value). The *settings* argument is the current settings for the language.

The front-end/complete-program method is expected to return a thunk that is called repeatedly to get all of the expressions in the program. When all expressions have been read, the thunk is expected to return eof.

This method is only called for programs in the definitions window. Notably, it is not called for programs that are loaded or evaled. See current-load and current-eval for those.

This method is expected to raise an appropriate exception if the program is malformed, eg an exn:syntax or exn:read.

This is called on the user's thread, as is the thunk it returns.

Implementations of this method should not return fully expanded expressions, since there are two forms of expansion, using either expand or expand-top-level-with-compile-time-evals and the use of the expanded code dictates which applies.

See also front-end/interaction and front-end/finished-complete-program.

```
(send a-drracket:language:language front-end/finished-
complete-program settings)
 → any
  settings : settings
```

This method is called when Run is clicked, but only after front-end/complete-program has been called. Specifically, front-end/complete-program is first called to get a thunk that reads from the program. That thunk is called some number of times, eventually returning eof, or raising an exception. Then, this method is called.

This method is called on the user's main eventspace thread, and without a prompt or other control delimiter. It must return without raising an error, or else the DrRacket window will be wedged.

```
 (send a-drracket:language:language front-end/interaction
  port
  settings)
 → (-> (or/c sexp/c syntax? eof-object?))
  port : input-port?
  settings : settings
```

This method is just like front-end/complete-program except that it is called with program fragments, for example the expressions entered in the interactions window. It is also used in other contexts by tools to expand single expressions.

See also front-end/finished-complete-program.

```
(send a-drracket:language:language get-comment-character)
 → string? char?
```

Returns text to be used for the "Insert Large Letters" menu item in DrRacket. The first result is a prefix to be placed at the beginning of each line and the second result is a character to be used for each pixel in the letters.

```
(send a-drracket:language:language get-language-name)
 → string?
```

Returns the name of the language, as shown in the REPL when executing programs in the language and in the bottom left of the DrRacket window.

```
(send a-drracket:language:language get-language-numbers)
 → (cons/c number? (listof number?))
```

This method is used in a manner analogous to get-language-position.

Each element in the list indicates how the names at that point in dialog will be sorted. Names with lower numbers appear first. If two languages are added to DrRacket with the same strings (as given by the get-language-position method) the corresponding numbers returned by this method must be the same. Additionally, no two languages can have the same set of numbers.

(Note: this method should always return the same result, for the same language.)

```
(send a-drracket:language:language get-language-position)
 → (cons/c string? (listof string?))
```

This method returns a list of strings that is used to organize this language with the other languages. Each entry in that list is a category or subcategory of the language and the last entry in the list is the name of the language itself. In the language dialog, each element in the list except for the last will be a nested turn down triangle on the left of the dialog. The final entry will be the name that the user can choose to select this language. Names that are the same will be combined into the same turndown entry.

For example, if one language's position is:

```
(list "General Category" "Specific Category" "My Lan-
guage")
```

and another's is:

```
(list "General Category" "Specific Category" "My Other
Language")
```

The language dialog will collapse the first two elements in the list, resulting in only a pair of nested turn-down triangles, not parallel pairs of nested turn-down triangles.

```
(send a-drracket:language:language get-language-url)
 → (or/c string? #f)
```

Specification: Returns a url for the language. *Default implementation:* If the result isn't #f, the name of the language is clickable in the interactions window and clicking takes you to this url.

```
(send a-drracket:language:language get-metadata modname
                                                settings)
 → string?
  modname : symbol?
  settings : any/c
```

This method is only called when get-reader-module returns an sexp.

It is expected to return a string that contains N lines, where N is the result of calling get-metadata-lines. The string is prefixed to the buffer before the file is saved by DrRacket, and removed from the buffer after it is opened in DrRacket.

The string is expect to be a prefix to the file that sets up a reader for files in this language, using #reader.

The *modname* argument's printed form is the same as the file's name, but without the path, and without an extension. The *settings* argument is the current language's settings value.

See also metadata->settings, get-metadata-lines, and get-reader-module.

```
(send a-drracket:language:language get-metadata-lines)
 → number?
```

This method is only called when get-reader-module returns an sexp.

The result of the method is a count of the number of lines in the strings that get-metadata returns. The get-metadata function does not necessarily return the same string each time it is called (see metadata->settings) but it is expected to always return a string with a fixed number of lines, as indicated by the result of this method.

```
(send a-drracket:language:language get-one-line-summary)
 → string?
```

Specification: The result of this method is shown in the language dialog when the user selects this language. *Default implementation:*

```
(send a-drracket:language:language get-reader-module)
 → (or/c sexp-representing-a-require-spec #f)
```

The result of this method is used when saving or loading files.

If the result is a sexp, saved files get a prefix inserted at the beginning (the prefix is determined by calling get-metadata). When the file is then loaded, DrRacket recognizes this prefix and sets the language back to match the saved file.

See also metadata->settings, get-metadata-lines, and get-metadata.

```
(send a-drracket:language:language get-style-delta)
 → (or/c #f
         (is-a?/c style-delta%)
         (listof (list/c (is-a?/c style-delta%)
                         number?
                         number?)))
```

The style delta that this method returns is used in the language dialog and the DrRacket REPL when the language's name is printed.

When it is `#f`, no styling is used.

If the result is a list, each element is expected to be a list of three items, a style-delta, and two numbers. The style delta will be applied to the corresponding portion of the name.

```
(send a-drracket:language:language extra-repl-information
 settings
 port)
 → void?
 settings : settings
 port : output-port?
```

This method is called on the DrRacket eventspace main thread to insert extra information into the REPL to reflect the state of the program.

It is used, for example, to print out the "Teachpack" lines in the HtDP languages.

```
(send a-drracket:language:language marshall-
settings settings)
 → writable
 settings : settings
```

Translates an instance of the settings type into a Racket object that can be written out to disk.

```
(send a-drracket:language:language metadata-
>settings metadata)
 → settings
 metadata : string?
```

This method is only called when `get-reader-module` returns an sexp.

When a file is opened in DrRacket, if this language's `get-reader-module` returns an sexp, the prefix of the file (the first N lines, where N is the number returned by `get-metadata-lines`) is scanned for `"#reader"` followed by the result of `get-reader-module`. If that pattern is found, the language is set to this language. Also, the entire prefix is passed, as a string, to this method which returns a `settings` value, used as the settings for this language.

```
(send a-drracket:language:language on-execute
 settings
 run-on-user-thread)
 → any
 settings : settings
 run-on-user-thread : ((-> any) -> any)
```

The on-execute method is called on DrRacket's eventspace's main thread before any evaluation happens when the Run button is clicked. It is also called when a new DrRacket tab (or window) is created to initialize the empty interactions window.

Use this method to initialize Racket's §11.3.2 "Parameters" for the user. When this function is called, the user's thread has already been created, as has its custodian. These parameters have been changed from the defaults in Racket:

- current-custodian is set to a new custodian.
- current-namespace has been set to a newly created empty namespace. This namespace has the following modules copied (with namespace-attach-module) from DrRacket's original namespace:
  - 'mzscheme
  - 'mred
- read-curly-brace-as-paren is #t,
- read-square-bracket-as-paren is #t,
- The port-write-handler and port-display-handler have been set to procedures that call pretty-print and pretty-display instead of write and display. When pretty-print and pretty-display are called by these parameters, the pretty-print-columns parameter is set to 'infinity, so the output looks just like write and display. This is done so that special scheme values can be displayed as snips.
- The current-print-covert-hook is to a procedure so that snip%s are just returned directly to be inserted into the interactions text% object.
- The output and input ports are set to point to the interactions window with these parameters: current-input-port, current-output-port, and current-error-port.
- The event-dispatch-handler is set so that DrRacket can perform some initial setup and close down around the user's code.
- The current-directory and current-load-relative-directory are set to the directory where the definitions file is saved, or if it isn't saved, to the initial directory where DrRacket started up.
- The snip-class-list, returned by get-the-snip-class-list is initialized with all of the snipclasses in DrRacket's eventspace's snip-class-list.
- The error-print-source-location parameter is set to #f and the error-display-handler is set to a handler that creates an error message from the exception record, with font and color information and inserts that error message into the definitions window.

The *run-on-user-thread* arguments accepts thunks and runs them on the user's eventspace's main thread. The output ports are not yet functioning, so print outs should be directed to the original DrRacket output port, if necessary.

This thunk is wrapped in a `with-handlers` that catches all exceptions matching `exn:fail?` and then prints out the exception message to the original output port of the DrRacket process.

```
(send a-drracket:language:language order-manuals manuals)
 → (listof bytes?) boolean?
  manuals : (listof bytes?)
```

Returns a sublist of its input, that specifies the manuals (and their order) to search in. The boolean result indicates if `doc.txt` files should be searched.

```
(send a-drracket:language:language render-value value
                                                settings
                                                port)
 → void?
  value : TST
  settings : settings
  port : port?
```

This method is just like `render-value/format` except that it is expected to put the entire value on a single line with no newline after the value.

```
 (send a-drracket:language:language render-value/format
  value
  settings
  port
  width)
 → void?
  value : TST
  settings : settings
  port : port?
  width : (or/c number? 'infinity)
```

This method is used to print values into a port, for display to a user. The final argument is a maximum width to use (in characters) when formatting the value.

This method is expected to format the value by inserting newlines in appropriate places and is expected to render a newline after the value.

See also `render-value`.

```
(send a-drracket:language:language unmarshall-
settings input)
 → (or/c settings #f)
  input : writable
```

Translates a Racket value into a settings, returning `#f` if that is not possible.

```
drracket:language:object/c : contract?

 (object-contract
  (config-panel (-> (is-a?/c area-container<%>)
                    (case-> (-> any/c void?)
                            (-> any/c))))
  (create-executable (-> any/c
                         (or/c (is-a?/c dialog%) (is-a?/c frame%))
                         path?
                         void?))
  (default-settings (-> any/c))
  (default-settings? (-> any/c boolean?))
  (front-end/complete-program (-> input-port?
                                  any/c
                                  (-> any/c)))
  (front-end/interaction (-> input-port?
                             any/c
                             (-> any/c)))
  (get-language-name (-> string?))
  (get-language-numbers (-> (cons/c number? (listof number?))))
  (get-language-position (-> (cons/c string? (listof string?))))
  (get-language-url (-> (or/c false/c string?)))
  (get-one-line-summary (-> (or/c #f string?)))
  (get-comment-character (-> (values string? char?)))
  (get-style-delta
   (-> (or/c false/c
             (is-a?/c style-delta%)
             (listof
              (list/c (is-a?/c style-delta%)
                      number?
                      number?)))))
  (marshall-settings (-> any/c printable/c))
  (on-execute (-> any/c (-> (-> any) any) any))
  (render-value (-> any/c
                    any/c
                    output-port?
                    void?))
  (render-value/format (-> any/c
                           any/c
                           output-port?
                           (or/c number? (symbols 'infinity))
                           any))
  (unmarshall-settings (-> printable/c any))

  (capability-value
```

```
(->i ((s (and/c symbol?
                drracket:language:capability-registered?)))
     (res (s) (drracket:language:get-capability-contract s)))))
```

```
(drracket:language:register-capability s
                                       the-contract
                                       default)     → void?
  s : symbol?
  the-contract : contract?
  default : the-contract
```

Registers a new capability with a default value for each language and a contract on the values the capability might have.

By default, these capabilities are registered as DrRacket starts up:

- `'drracket:check-syntax-button : boolean? = #t` — controls the visiblity of the check syntax button
- `'drracket:language-menu-title : string? = (string-constant scheme-menu-name)` — controls the name of the menu just to the right of the language menu (named “Racket” by default)
- `'drscheme:define-popup :`

```
(or/c #f
      (list/c string? string? string?)
      (non-empty-listof (list/c string? string? string?))
      (non-empty-listof (list/c string? string? string?
                                (or/c #f
                                      (-> (is-a/c text%)
                                          string?
                                          exact-integer?
                                          (->* ((is-
a/c text%)
                                                string?
                                                exact-
integer?)
                                               (#:case-
sensitive? any/c
                                                #:delimited? any/c)
                                               (or/c exact-
integer? #f))
```

```
                                                (or/c exact-
integer? #f)))
                                        (or/c #f
                                              (-> (is-a/c text%)
                                                  exact-integer?
                                                  (-> (is-a/c text%)
                                                      exact-integer?
                                                      string?)
                                                  string?))))
        (cons/c string? string?))
```

= (list "(define" "(define ...)" "δ") — specifies the prefix that the define popup should look for and what label it should have, or #f if it should not appear at all. Text is found only when it is not in a comment or string literal.

If the list of three strings alternative is used, the first string is the prefix that is looked for when finding definitions. The second and third strings are used as the label of the control, in horizontal and vertical mode, respectively.

If it is a list of lists, then multiple prefixes are used for the definition pop-up. The name of the popup menu is based only on the first element of the list. When a nested list contains fourth and fifth elements, they can supply replacements (when not #f) for the default functions that find a prefix and extract the subsequent name. See §1.9 "Definition Popup-Menu Navigation" for information about the protocols for the finding and extraction procedures.

The pair of strings alternative is deprecated. If it is used, the pair (cons a-str b-str) is the same as (list a-str b-str "δ").

- 'drscheme:help-context-term : (or/c false/c string?) = #f — specifies a context query for documentation searches that are initiated in this language, can be #f (no change to the user's setting) or a string to be used as a context query (note: the context is later maintained as a cookie, "" is different from #f in that it clears the stored context)
- 'drscheme:special:insert-fraction : boolean? = #t — determines if the insert fraction menu item in the special menu is visible
- 'drscheme:special:insert-lambda : boolean? = #t — determines if the insert lambda menu item in the special menu is visible
- 'drscheme:special:insert-large-letters : boolean? = #t — determines if the insert large letters menu item in the special menu is visible
- 'drscheme:special:insert-image : boolean? = #t — determines if the insert image menu item in the special menu is visible
- 'drscheme:special:insert-comment-box : boolean? = #t — determines if the insert comment box menu item in the special menu is visible

- `'drscheme:special:insert-gui-tool : boolean? = #t` — determines if the insert gui menu item in the special menu is visible
- `'drscheme:special:slideshow-menu-item : boolean? = #t` — determines if the insert pict box menu item in the special menu is visible
- `'drscheme:special:insert-text-box : boolean? = #t` — determines if the insert text box menu item in the special menu is visible
- `'drscheme:special:xml-menus : boolean? = #t` — determines if the insert scheme box, insert scheme splice box, and the insert xml box menu item in the special menu are visible
- `'drscheme:autocomplete-words : (listof string?) = '()` — determines the list of words that are used when completing words in this language
- `'drscheme:tabify-menu-callback :`

  ```
  (or/c (-> (is-a?/c text%)
            number?
            number?
            void?)
        #f)
  ```

  `= (λ (t a b) (send t tabify-selection a b))` — is used as the callback when the "Reindent" or "Reindent All" menu is selected. The first argument is the editor, and the second and third are a range in the editor.

```
(drracket:language:capability-registered? s) → boolean?
  s : symbol?
```

Indicates if `drracket:language:register-capability` has been called with `s`.

```
(drracket:language:get-capability-default s)
 → (drracket:language:get-capability-contract s)
  s : (and/c symbol? drracket:language:capability-registered?)
```

Returns the default for a particular capability.

```
(drracket:language:get-capability-contract s) → contract?
  s : (and/c symbol? drracket:language:capability-registered?)
```

Returns the contract for a given capability, which was specified when `drracket:language:register-capability` was called.

```
(drracket:language:add-snip-value  test-value
                                   convert-value
                                  [setup-thunk]) → void?
  test-value : (-> any/c boolean?)
  convert-value : (-> any/c (is-a?/c snip%))
  setup-thunk : (-> any/c) = void
```

Registers a handler to convert values into snips as they are printed in the REPL.

The `test-snip` argument is called to determine if this handler can convert the value and the *convert-value* argument is called to build a snip. The (optional) *setup-thunk* is called just after the user's namespace and other setings are built, but before any of the user's code is evaluated.

All three functions are called on the user's thread and with the user's settings.

```
(drracket:language:extend-language-interface
 interface
 default-implementation)
→ void?
 interface : interface?
 default-implementation : (make-mixin-contract drracket:language:language<%>)
```

This function can only be called in phase 1 (see §2 "Implementing DrRacket Plugins" for details).

Each language added passed to `drracket:language-configuration:add-language` must implement *interface*.

The *default-implementation* is a mixin that provides a default implementation of *interface*. Languages that are unaware of the specifics of `extension` use *default-implementation* via `drracket:language:get-default-mixin`.

```
(drracket:language:get-default-mixin)
 → (make-mixin-contract drracket:language:language<%>)
```

This function can only be called in phase 2 (see §2 "Implementing DrRacket Plugins" for details).

The result of this function is the composite of all of the `default-implementation` arguments passed to `drracket:language:extend-language-interface`.

```
(drracket:language:get-language-extensions)
 → (listof interface?)
```

This function can only be called in phase 2 (see §2 “Implementing DrRacket Plugins” for details).

Returns a list of the interfaces passed to drracket:language:extend-language-interface.

```
(drracket:language:put-executable parent
                                  program-filename
                                  mode
                                  mred?
                                  title)
 → (or/c false/c path?)
  parent : (is-a?/c top-level-window<%>)
  program-filename : path?
  mode : (or/c boolean? 'launcher 'standalone 'distribution)
  mred? : boolean?
  title : string?
```

Calls the GRacket primitive put-file with arguments appropriate for creating an executable from the file *program-filename*.

The arguments *mred?* and *mode* indicates what type of executable this should be (and the dialog may be slightly different on some platforms, depending on these arguments). For historical reasons, #f is allowed for *mode* as an alias for 'launcher, and #t is allowed for *mode* as an alias for 'stand-alone.

The *title* argument is used as the title to the primitive put-file or get-directory primitive.

```
(drracket:language:create-executable-gui parent
                                         program-name
                                         show-type
                                         show-base)
 → (or/c #f
         (list/c (or/c 'no-show 'launcher 'stand-alone 'distribution)
                 (or/c 'no-show 'mred 'mzscheme)
                 string?
                 (listof (cons/c symbol? any/c))))
  parent : (or/c #f (is-a?/c top-level-window<%>))
  program-name : (or/c #f string?)
  show-type : (or/c #t 'launcher 'standalone 'distribution)
  show-base : (or/c #t 'mzscheme 'mred)
```

Opens a dialog to prompt the user about their choice of executable. If *show-type* is #t, the user is prompted about a choice of executable: stand-alone, launcher, or distribution;

otherwise, the symbol determines the type. If *show-base* is #t, the user is prompted about a choice of base binary: mzscheme or mred; otherwise the symbol determines the base.

The *program-name* argument is used to construct the default executable name in a platform-specific manner.

The *parent* argument is used for the parent of the dialog.

The result of this function is #f if the user cancel's the dialog and a list of three items indicating what options they chose. If either *show-type* or *show-base* was not #t, the corresponding result will be 'no-show, otherwise it will indicate the user's choice.

```
(drracket:language:create-module-based-stand-alone-executable
 program-filename
 executable-filename
 module-language-spec
 transformer-module-language-spec
 init-code
 gui?
 use-copy?)
→ void?
 program-filename : (or/c path? string?)
 executable-filename : (or/c path? string?)
 module-language-spec : any/c
 transformer-module-language-spec : any/c
 init-code : any/c
 gui? : boolean?
 use-copy? : boolean?
```

This procedure creates a stand-alone executable in the file *executable-filename* that runs the program *program-filename*.

The arguments *module-language-spec* and *transformer-module-language-spec* specify the settings of the initial namespace, both the transformer portion and the regular portion. Both may be #f to indicate there are no initial bindings.

The *init-code* argument is an s-expression representing the code for a module. This module is expected to provide the identifier *init-code*, bound to a procedure of no arguments. That module is required and the *init-code* procedure is executed to initialize language-specific settings before the code in *program-filename* runs.

The *gui?* argument indicates if a GRacket or Racket stand-alone executable is created.

The *use-copy?* argument indicates if the initial namespace should be populated with namespace-require/copy or namespace-require.

```
(drracket:language:create-module-based-distribution
 program-filename
 distribution-filename
 module-language-spec
 transformer-module-language-spec
 init-code
 gui?
 use-copy?)
→ void?
 program-filename : (or/c path? string?)
 distribution-filename : (or/c path? string?)
 module-language-spec : any/c
 transformer-module-language-spec : any/c
 init-code : any/c
 gui? : boolean?
 use-copy? : boolean?
```

Like drracket:language:create-module-based-stand-alone-executable, but packages the stand-alone executable into a distribution.

```
(drracket:language:create-distribution-for-executable
 distribution-filename
 gui?
 make-executable)
→ void?
 distribution-filename : (or/c path? string?)
 gui? : boolean?
 make-executable : (-> path? void?)
```

Creates a distribution where the given *make-executable* procedure creates the stand-alone executable to be distributed. The *make-executable* procedure is given the name of the executable to create. The *gui?* argument is needed in case the executable's name (which drracket:language:create-distribution-for-executable must generate) depends on the type of executable. During the distribution-making process, a progress dialog is shown to the user, and the user can click an Abort button that sends a break to the current thread.

```
(drracket:language:create-module-based-launcher
 program-filename
 executable-filename
 module-language-spec
 transformer-module-language-spec
 init-code
 gui?
 use-copy?)
→ void?
 program-filename : (or/c path? string?)
 executable-filename : (or/c path? string?)
 module-language-spec : any/c
 transformer-module-language-spec : any/c
 init-code : any/c
 gui? : boolean?
 use-copy? : boolean?
```

This procedure is identical to drracket:language:create-module-based-stand-alone-executable, except that it creates a launcher instead of a stand-alone executable.

```
(drracket:language:simple-module-based-language-convert-value
 value
 settings)
→ any
 value : any/c
 settings : drracket:language:simple-settings?
```

The result can be either one or two values. The first result is the converted value. The second result is #t if the converted value should be printed with write (or pretty-write), #f if the converted result should be printed with print (or pretty-print); the default second result is #t.

The result of this function depends on the simple-settings-printing-style field of *settings*. If it is 'print, the result is (values *value* #f). If it is 'write or 'trad-write, the result is just *value*. Otherwise, the result is produced by adjusting the constructor-style-printing and show-sharing parameters based on *settings*, setting current-print-convert-hook to ignore snips, and then applying print-convert to *value*.

```
(drracket:language:setup-printing-parameters thunk
                                             settings
                                             width)   → any
 thunk : (-> any)
 settings : drracket:language:simple-settings?
 width : (or/c number? 'infinity)
```

Equivalent to (drracket:language:make-setup-printing-parameters).

```
(drracket:language:make-setup-printing-parameters)
 → (-> (-> any) drracket:language:simple-settings? (or/c number? 'infinity) any)
```

Returns a procedure that accepts three arguments: a thunk, settings, and a pretty-print width. The result procedure, when invoked sets all of the pretty-print and print-convert parameters either to the defaults to values based on settings and then invokes thunk, returning what it returns.

When drracket:language:make-setup-printing-parameters is invoked, it dynamic-requires pict/convert and closes over the results, using them to convert values when the resulting procedure is invoked.

```
(struct drracket:language:text/pos (text start end)
    #:extra-constructor-name make-drracket:language:text/pos)
  text : (is-a?/c text%)
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
```

A record that tracks a text% object and a range inside it.

```
drracket:language:make-text/pos : procedure?
```

An alias for make-drracket:language:text/pos.

```
drracket:language:struct:text/pos : struct-type?
```

An alias for struct:drracket:language:text/pos.

```
(struct drracket:language:simple-settings (case-sensitive
                                           printing-style
                                           fraction-style
                                           show-sharing
                                           insert-newlines
                                           annotations)
    #:extra-constructor-name
    make-drracket:language:simple-settings)
  case-sensitive : boolean?
  printing-style : (or/c 'constructor
                         'quasiquote
                         'write
                         'trad-write
                         'print)
```

```
  fraction-style : (or/c 'mixed-fraction
                         'mixed-fraction-e
                         'repeating-decimal
                         'repeating-decimal-e)
  show-sharing : boolean?
  insert-newlines : boolean?
  annotations : (or/c 'none 'debug 'debug/profile 'test-coverage)
```

A struct that tracks commonly used settings for a language.

```
drracket:language:make-simple-settings : procedure?
```

An alias for `make-drracket:language:simple-settings`.

```
drracket:language:struct:simple-settings : struct-type?
```

An alias for `struct:drracket:language:simple-settings`.

```
(drracket:language:simple-settings->vector simple-settings)
 → vector?
  simple-settings : drracket:language:simple-settings?
```

Constructs a vector whose elements are the fields of *simple-settings*.

16 drracket:language-configuration

```
(drracket:language-configuration:get-languages)
 → (listof (is-a?/c drracket:language:language<%>))
```

This can only be called after all of the tools initialization phases have completed.

Returns the list of all of the languages installed in DrRacket.

```
(drracket:language-configuration:add-language
  language
 [#:allow-executable-creation? allow-executable-creation?])
→ void?
 language : (and/c (is-a?/c drracket:language:language<%>)
                   drracket:language:object/c)
 allow-executable-creation? : boolean? = #f
```

This function can only be called in phase 2 (see §2 “Implementing DrRacket Plugins” for details).

Adds *language* to the languages offered by DrRacket.

If *allow-executable-creation?* is `#f`, then choosing the Create Executable... menu item results in a dialog box saying that executable creation is disabled. If it is `#t`, then the `create-executable` is called when that menu item is selected (after checking to make sure the file is saved).

```
(drracket:language-configuration:get-settings-preferences-symbol)
 → symbol?
```

Returns the symbol that is used to store the user’s language settings. Use as an argument to either `preferences:get` or `preferences:set`.

```
(struct drracket:language-configuration:language-settings
        (language
         settings)
    #:extra-constructor-name
    make-drracket:language-configuration:language-settings)
  language : (or/c (is-a?/c drracket:language:language<%>)
                   drracket:language:object/c)
  settings : any/c
```

This struct pairs together a language and some specific settings for the language.

The settings is a language-specific record that holds a value describing a parameterization of the language.

```
drracket:language-configuration:struct:language-settings
 : struct-type?
```

An alias for `struct:drracket:language-configuration:language-settings`.

```
drracket:language-configuration:make-language-settings
 : procedure?
```

An alias for `make-drracket:language-configuration:language-settings`.

```
(drracket:language-configuration:language-dialog
  show-welcome?
  language-settings-to-show
 [parent])
→ (or/c false/c drracket:language-configuration:language-settings?)
 show-welcome? : boolean?
 language-settings-to-show : drracket:language-configuration:language-settings?
 parent : (or/c false/c (is-a?/c top-level-window<%>)) = #t
```

Opens the language configuration dialog. See also `drracket:language-configuration:fill-language-dialog`.

The *show-welcome?* argument determines if if a "Welcome to DrRacket" message and some natural language buttons are shown.

The *language-settings-to-show* argument must be some default language settings that the dialog is initialized to. If unsure of a default, the currently set language in the user's preferences can be obtained via:

```
(preferences:get
 (drracket:language-configuration:get-settings-preferences-
symbol))
```

The *parent* argument is used as the parent to the dialog.

The result if `#f` when the user cancells the dialog, and the selected language if they hit ok.

```
(drracket:language-configuration:fill-language-dialog
 panel
 button-panel
 language-setting
 [re-center
 ok-handler])
→ (-> (is-a?/c drracket:language:language<%>))
   (-> any/c)
   (-> any/c (is-a?/c mouse-event%) any)
 panel : (is-a?/c vertical-panel%)
 button-panel : (is-a?/c area-container<%>)
 language-setting : drracket:language-configuration:language-settings?
 re-center : (or/c false/c (is-a?/c top-level-window<%>)) = #f
 ok-handler : (-> symbol? void?) = void
```

This procedure accepts two parent panels and fills them with the contents of the language dialog. It is used to include language configuration controls in some larger context in another dialog.

The *panel* argument is the main panel where the language controls will be placed. The function adds buttons to the *button-panel* to revert a language to its default settings and to show the details of a language.

The *language-setting* is the default language to show in the dialog.

The *re-center* argument is used when the Show Details button is clicked. If that argument is a `top-level-window<%>`, the Show Details callback will recenter the window each time it is clicked. Otherwise, the argument is not used.

ok-handler is a function that is in charge of interfacing the OK button. It should accept a symbol message: `'enable` and `'disable` to toggle the button, and `'execute` to run the desired operation. (The language selection dialog also uses an internal `'enable-sync` message.)

The first two results of the function return a language object and a settings for that language, as chosen by the user using the dialog. The final function should be called when keystrokes are typed in the enclosing frame. It is used to implement the shortcuts that choose the two radio buttons in the language dialog.

17 drracket:debug

```
drracket:debug:profile-unit-frame-mixin : (class? . -> . class?)
  argument extends/implements: drracket:frame:<%>
                               drracket:unit:frame<%>
```

Adds support for profiling information.

```
(send a-drracket:debug:profile-unit-frame show-profile-gui)
 → void?
```

Shows the GUI information shown about the profile.

```
(send a-drracket:debug:profile-unit-frame hide-profile-gui)
 → void?
```

Hides the GUI information shown about the profile.

```
drracket:debug:profile-tab-mixin : (class? . -> . class?)
  argument extends/implements: drracket:unit:tab<%>
  result implements: drracket:debug:profile-interactions-tab<%>
```

Tracks profiling information.

```
drracket:debug:profile-definitions-text-mixin : (class? . -> . class?)
  argument extends/implements: drracket:unit:definitions-text<%>
                               text%
```

Tracks profiling information.

```
drracket:debug:test-coverage-definitions-text-mixin : (class? . -> .
class?)
  argument extends/implements: text%
                               drracket:unit:definitions-text<%>
```

Tracks test case coverage information.

```
drracket:debug:test-coverage-interactions-text-mixin : (class? . -> .
class?)
```
argument extends/implements: drracket:rep:text<%>
text:basic<%>
result implements: drracket:debug:test-coverage-interactions-text<%>

Tracks test case coverage information.

```
drracket:debug:test-coverage-tab-mixin : (class? . -> . class?)
```
argument extends/implements: drracket:rep:context<%>
drracket:unit:tab<%>
result implements: drracket:debug:test-coverage-tab<%>

Tracks test case coverage information.

```
drracket:debug:test-coverage-frame-mixin : (class? . -> . class?)
```
argument extends/implements: drracket:unit:frame<%>
result implements: drracket:debug:test-coverage-frame<%>

Tracks test case coverage information.

```
(drracket:debug:error-display-handler/stacktrace
 msg
 exn
 [stack
  #:definitions-text defs
  #:interactions-text ints])
→ any/c
 msg : string?
 exn : any/c
 stack : (or/c false/c (listof srcloc?)) = #f
 defs : (or/c #f (is-a?/c drracket:unit:definitions-text<%>))
      = #f
 ints : (or/c #f (is-a?/c drracket:rep:text<%>)) = #f
```

Displays the error message represented by the string, adding embellishments like those that appears in the DrRacket REPL, specifically a clickable icon for the stack trace (if the srcloc location is not empty), and a clickable icon for the source of the error (read & syntax errors show their source locations and otherwise the first place in the stack trace is shown).

If *stack* is false, then the stack traces embedded in the *exn* argument (if any) are used. Specifically, this function looks for a stacktrace via errortrace-key in the continuation marks of *exn* and continuation-mark-set->context.

If *stack* is not false, that stack is added to the stacks already in the exception.

This should be called in the same eventspace and on the same thread as the error.

```
(drracket:debug:make-debug-error-display-handler oedh)
 → (-> string? (or/c any/c exn?) any)
  oedh : (-> string? (or/c any/c exn?) any)
```

This function implements an error-display-handler in terms of another error-display-handler.

See also Racket's error-display-handler parameter.

If the current-error-port is the definitions window in DrRacket, this error handler inserts some debugging annotations, calls *oedh*, and then highlights the source location of the run-time error.

It looks for both stack trace information in the continuation marks both via the errortrace/errortrace-key module and via continuation-mark-set->context.

```
(drracket:debug:hide-backtrace-window) → void?
```

Hides the backtrace window.

```
(drracket:debug:add-prefs-panel) → void?
```

Adds the profiling preferences panel.

```
(drracket:debug:make-debug-compile-handler oc)
 → (-> any/c boolean? compiled-expression?)
  oc : (-> any/c boolean? compiled-expression?)
```

Returns a function suitable for use with current-compile.

The result function first adds debugging information to its argument and then passes it to *oc*.

```
(drracket:debug:make-debug-eval-handler oe) → (-> any/c any)
  oe : (-> any/c any)
```

Returns a function suitable for use with `current-eval`.

The result function first adds debugging information to its argument and then passes it to *oe*.

```
(drracket:debug:test-coverage-enabled) → boolean?
(drracket:debug:test-coverage-enabled enabled?) → void?
  enabled? : boolean?
```

Determines if the test-coverage annotation is added by the result of `drracket:debug:make-debug-eval-handler`.

```
drracket:debug:test-coverage-on-style-name : string?
```

The name of the `style%` object (in `editor:get-standard-style-name`) used to indicate a covered region of code.

```
drracket:debug:test-coverage-off-style-name : string?
```

The name of the `style%` object (in `editor:get-standard-style-name`) used to indicate a region of code that tests (or any code, really) didn't cover.

```
(drracket:debug:profiling-enabled) → boolean?
(drracket:debug:profiling-enabled enabled?) → void?
  enabled? : boolean?
```

Determines if the profiling annotation is added by the result of `drracket:debug:make-debug-eval-handler`.

```
(drracket:debug:bug-info->ticket-url query) → url?
  query : (listof (cons/c symbol? (or/c #f string?)))
```

Builds a url that goes to the trac report system. The *query* argument is used as the url's query field.

```
drracket:debug:small-planet-bitmap : (is-a?/c bitmap%)
```

The icon used in the DrRacket REPL when an exception is raised that includes blame information blaming a PLaneT package. (Clicking the icon connects to the PLaneT bug report form.)

```
(drracket:debug:open-and-highlight-in-file  debug-info
                                            [edition-pair])
 → void?
  debug-info : (or/c srcloc? (listof srcloc?))
  edition-pair :  (or/c #f (cons/c (λ (x) (and (weak-box? x)
                                               (let ([v (weak-box-value x)])
                                                 (or (not v)
                                                     (is-a? v editor<%>))))))
                                   number?))
                = #f
```

This function opens a DrRacket to display *debug-info*. Only the src the position and the span fields of the srcloc are considered.

The *edition-pair* is used to determine if a warning message is shown when before opening the file. If the *edition-pair* is not #f, it is compared with the result of get-edition-number of the editor that is loaded to determine if the file has been edited since the source location was recorded. If so, it puts up a warning dialog message to that effect.

```
(drracket:debug:show-backtrace-window/edition-pairs
 error-message
 dis
 editions-pairs
 defs
 ints)
→ void?
 error-message : string?
 dis : (listof srcloc?)
 editions-pairs : (listof
                   (or/c
                    #f
                    (cons/c (λ (x)
                              (and (weak-box? x)
                                   (let ([v (weak-box-value x)])
                                     (or (not v)
                                         (is-a? v editor<%>))))))
                            number?)))
 defs : (or/c #f (is-a?/c drracket:unit:definitions-text<%>))
 ints : (or/c #f (is-a?/c drracket:rep:text<%>))
```

Same as drracket:debug:show-backtrace-window/edition-pairs/two, where the

dis2 and *editions-pairs2* arguments are both '()

```
(drracket:debug:show-backtrace-window/edition-pairs/two
 error-message
 dis1
 editions-pairs1
 dis2
 editions-pairs2
 defs
 ints)
→ void?
 error-message : string?
 dis1 : (listof srcloc?)
 editions-pairs1 : (listof
                    (or/c
                     #f
                     (cons/c (λ (x)
                               (and (weak-box? x)
                                    (let ([v (weak-box-value x)])
                                      (or (not v)
                                          (is-a? v editor<%>)))))
                             number?)))
 dis2 : (listof srcloc?)
 editions-pairs2 : (listof
                    (or/c
                     #f
                     (cons/c (λ (x)
                               (and (weak-box? x)
                                    (let ([v (weak-box-value x)])
                                      (or (not v)
                                          (is-a? v editor<%>)))))
                             number?)))
 defs : (or/c #f (is-a?/c drracket:unit:definitions-text<%>))
 ints : (or/c #f (is-a?/c drracket:rep:text<%>))
```

Shows the backtrace window you get when clicking on the bug in DrRacket's REPL.

The *error-message* argument is the text of the error, *dis1* and *dis2* are the stack-trace information, extracted from the continuation mark in the exception record, using `errortrace-key` and using `continuation-mark-set->context`.

The `editions1` and `editions2` arguments indicate the editions of any editors that are open editing the files corresponding to the source locations. The lists must have the same length as *dis1* and *dis2*.

The *defs* argument should be non-`#f` if there are possibly stacktrace frames that contain

unsaved versions of the definitions window from DrRacket. Similarly, the *ints* argument should be non-#f if there are possibly stacktrace frames that contain unsaved versions of the interactions window.

Use drracket:rep:current-rep to get the rep during evaluation of a program.

```
(drracket:debug:get-error-color) → (is-a?/c color%)
```

Returns the background color used to highlight errors in the definitions window (and other places, possibly). See also drracket:debug:get-error-color-name.

The result depends on the result of (color-prefs:white-on-black-color-scheme?).

```
(drracket:debug:get-error-color-name)
 → color-prefs:color-scheme-color-name?
```

Returns the name of the background color used to highlight errors in the definitions window (and other places, possibly).

```
(drracket:debug:show-backtrace-window  error-message
                                       dis
                                      [rep
                                       defs])        → void?
  error-message : string?
  dis : (or/c exn?
              (listof srcloc?))
  rep : (or/c #f (is-a?/c drracket:rep:text<%>)) = #f
  defs :  (or/c #f (is-a?/c drracket:unit:definitions-text<%>))
       = #f
```

Shows the backtrace window you get when clicking on the bug in DrRacket's REPL.

If *dis* is a list of srcloc?, then this function simply calls drracket:debug:show-backtrace-window/edition-pairs, passing *error-message*, *dis*, and a list of #f that is as long as *dis*.

If *dis* is an exn:fail?, then this function calls drracket:debug:show-backtrace-window/edition-pairs/two, extracting the builtin stack trace (via continuation-mark-set->context) and an errortrace stack trace from the continuation marks in exn.

18 drracket:rep

```
drracket:rep:text<%> : interface?
```

```
drracket:rep:text% : class?
  superclass: racket:text%
  extends: drracket:rep:text<%>
```

This class implements a read-eval-print loop for DrRacket. User submitted evaluations in DrRacket are evaluated asynchronously, in an eventspace created for the user. No evaluations carried out by this class affect the implementation that uses it.

```
(make-object drracket:rep:text% context)
 → (is-a?/c drracket:rep:text%)
  context : (implements drracket:rep:context<%>)
```

```
(send a-drracket:rep:text after-delete) → void?
```

Overrides `after-delete` in `mode:host-text-mixin`.

Resets any error highlighting in this editor.

```
(send a-drracket:rep:text after-insert) → void?
```

Overrides `after-insert` in `mode:host-text-mixin`.

Resets any error highlighting in this editor.

```
(send a-drracket:rep:text display-results results) → void?
  results : (list-of TST)
```

This displays each of the elements of *results* in the interactions window, except those elements of *results* that are void. Those are just ignored.

```
(send a-drracket:rep:text evaluate-from-port
 port
 complete-program?
 cleanup)
→ any
 port : input-port?
 complete-program? : boolean?
 cleanup : (-> void)
```

Evaluates the program in the *port* argument. If *complete-program?* is #t, this method calls the front-end/complete-program to evaluate the program. If it is #f, it calls front-end/interaction method. When evaluation finishes, it calls *cleanup* on the user's main thread.

Just before calling *cleanup*, this invokes the thunk in drracket:rep:after-expression (if any). It takes the value of the drracket:rep:after-expression parameter on the DrRacket main thread, but invokes the thunk on the user's thread.

This method must be called from the DrRacket main thread.

```
(send a-drracket:rep:text after-many-evals) → any
```

Refine this method with augment.

Called from the DrRacket main thread after evaluate-from-port finishes (no matter how it finishes).

If the call to evaluate-from-port was from the call that sets up the initial read-eval-print loop, then the value of drracket:rep:module-language-initial-run will be #t; otherwise it will be #f.

```
(send a-drracket:rep:text on-execute run-on-user-
thread) → any
  run-on-user-thread : (-> any)
```

Use *run-on-user-thread* to initialize the user's parameters, etc.

Called from the DrRacket thread after the language's on-execute method has been invoked, and after the special values have been setup (the ones registered via drracket:language:add-snip-value).

Do not print to current-output-port or current-error-port during the dynamic extent of the thunk passed to *run-on-user-thread* because this can deadlock. IO is still, in general, fine, but the current-error-port and current-output-port are set to the user's ports that print into the interactions window and are not in a good state during those calls.

```
(send a-drracket:rep:text get-error-range)
 → (or/c false/c (list/c (is-a?/c text:basic%) number? number?))
```

Specification: Indicates the highlighted error range. The state for the error range is shared across all instances of this class, so there can only be one highlighted error region at a time.

Default implementation: If #f, no region is highlighted. If a list, the first element is the editor where the range is highlighted and the second and third are the beginning and ending regions, respectively.

```
(send a-drracket:rep:text get-user-custodian)
 → (or/c false/c custodian?)
```

This is the custodian controlling the user's program.

```
(send a-drracket:rep:text get-user-eventspace)
 → (or/c false/c eventspace?)
```

This is the user's eventspace. The result of `get-user-thread` is the main thread of this eventspace.

```
(send a-drracket:rep:text get-user-language-settings)
 → language-settings
```

Returns the user's language-settings for the most recently run program. Consider using `get-next-settings` instead, since the user may have selected a new language since the program was last run.

```
(send a-drracket:rep:text get-user-namespace)
 → (or/c false/c namespace?)
```

Returns the user's namespace. This method returns a new namespace each time Run is clicked.

```
(send a-drracket:rep:text get-user-thread)
 → (or/c false/c thread?)
```

This method returns the thread that the user's code runs in. It returns a different result each time the user runs the program.

It is `#f` before the first time the user click on the Run button or the evaluation has been killed.

This thread has all of its parameters initialized according to the settings of the current execution. See §11.3.2 "Parameters" for more information about parameters.

```
(send a-drracket:rep:text highlight-errors locs
                                          [error-arrows])
 → void?
  locs : (listof srcloc?)
  error-arrows : (or/c #f (listof srcloc?)) = #f
```

Call this method to highlight errors associated with this repl. See also `reset-highlighting`, and `highlight-errors/exn`.

This method highlights a series of dis-contiguous ranges in the editor.

It puts the caret at the location of the first error.

```
(send a-drracket:rep:text highlight-
errors/exn exn) → void?
  exn : exn
```

Highlights the errors associated with the exn (only syntax and read errors – does not extract any information from the continuation marks)

See also highlight-errors.

```
(send a-drracket:rep:text on-highlighted-
errors loc/s) → void?
  loc/s : (or/c srcloc? (listof srcloc?))
```

This method is called when an error is highlighted in a DrRacket window.

If the input is a list of srcloc? objects, then all of them are highlighted, and they are all of the errors known to DrRacket at this point.

If a single one is passed, then user probably typed the . menu shortcut to highlight a single error and there may be other errors known to DrRacket.

Errors are made known to DrRacket via highlight-errors.

```
(send a-drracket:rep:text initialize-console) → void?
```

This inserts the "Welcome to DrRacket" message into the interactions buffer, calls reset-console, insert-prompt, and clear-undos.

Once the console is initialized, this method calls first-opened. Accordingly, this method should not be called to initialize a REPL when the user's evaluation is imminent. That is, this method should be called when new tabs or new windows are created, but not when the Run button is clicked.

This method calls the first-opened from the user's eventspace's main thread and, when first-opened returns, it enqueue's a callback that ends an edit sequence on the REPL and calls clear-undos. Accordingly, if the first-opened method does not return, the interactions text will be in an unclosed edit sequence.

```
(send a-drracket:rep:text insert-prompt) → void?
```

Inserts a new prompt at the end of the text.

```
(send a-drracket:rep:text kill-evaluation) → void?
```

This method is called when the user chooses the kill menu item.

```
(send a-drracket:rep:text on-close) → void?
```

Overrides on-close in editor:basic<%>.

Calls shutdown.

Calls the super method.

```
(send a-drracket:rep:text queue-output thnk) → void?
  thnk : (-> void?)
```

Specification: This method queues thunks for DrRacket's eventspace in a special output-related queue.

```
(send a-drracket:rep:text reset-console) → void?
```

Kills the old eventspace, and creates a new parameterization for it.

```
(send a-drracket:rep:text reset-highlighting) → void?
```

This method resets the highlighting being displayed for this repl. See also: highlight-errors, and highlight-errors/exn.

```
(send a-drracket:rep:text run-in-evaluation-
thread f) → void?
  f : ( -> void)
```

Specification: This function runs its arguments in the user evaluation thread. This thread is the same as the user's eventspace main thread.

Default implementation: Calls *f*, after switching to the user's thread.

```
(send a-drracket:rep:text shutdown) → void?
```

Shuts down the user's program and all windows. Reclaims any resources the program allocated. It is expected to be called from DrRacket's main eventspace thread.

```
(send a-drracket:rep:text wait-for-io-to-complete) → void?
```

This waits for all pending IO in the rep to finish and then returns.

This method must only be called from the main thread in DrRacket's eventspace

```
(send a-drracket:rep:text wait-for-io-to-complete/user)
 → void?
```

This waits for all pending IO in the rep to finish and then returns.

This method must only be called from the main thread in the user's eventspace

```
drracket:rep:drs-bindings-keymap-mixin : (class? . -> . class?)
  argument extends/implements: editor:keymap<%>
```

This mixin adds some DrRacket-specific keybindings to the editor it is mixed onto.

```
(send a-drracket:rep:drs-bindings-keymap get-keymaps)
 → (listof (is-a?/c keymap%))
```

Overrides `get-keymaps` in `editor:keymap<%>`.

Calls the super method and adds in a keymap with the DrRacket-specific keybindings:

- f5 - Run
- c:x;o - toggles the focus between the definition and interactions windows.

```
drracket:rep:context<%> : interface?
```

Objects that match this interface provide all of the services that the `drracket:rep:text%` class needs to connect with its context.

```
(send a-drracket:rep:context clear-annotations) → void?
```

Specification: Call this method to clear any annotations in the text before executing or analyzing or other such activities that should process the program.

Tools that annotate the program text should augment this method to clear their own annotations on the program text.

DrRacket calls this method before a program is run (via the Run button).

Default implementation: Clears any error highlighting in the definitions window.

```
(send a-drracket:rep:context disable-evaluation) → void?
```

Call this method to disable evaluation GUI evaluation while some evaluation (or expansion) is taking place on another thread.

Override this method if you add a GUI-based mechanism for initiating evaluation in the frame.

This method is also called when the user switches tabs.

See also `enable-evaluation`.

```
(send a-drracket:rep:context enable-evaluation) → void?
```

This method must disable the GUI controls that start user-sponsored evaluation. It is called once the user starts some evaluation to ensure that only one evaluation proceeds at a time.

It is also called when the user switches tabs.

See also `disable-evaluation`.

```
(send a-drracket:rep:context ensure-rep-shown rep) → void?
  rep : (is-a?/c drracket:rep:text<%>)
```

This method is called to force the rep window to be visible when, for example, an error message is put into the rep. Also ensures that the appropriate tab is visible, if necessary.

```
(send a-drracket:rep:context get-breakables)
 → (or/c thread? false/c)
    (or/c custodian? false/c)
```

Returns the last values passed to `set-breakables`.

```
(send a-drracket:rep:context get-directory) → path?
```

The result of this method is used as the initial directory for the user's program to be evaluated in.

```
(send a-drracket:rep:context needs-execution)
 → (or/c string? false/c)
```

This method should return an explanatory string when the state of the program that the repl reflects has changed. It should return `#f` otherwise.

```
(send a-drracket:rep:context reset-offer-kill) → void?
```

The break button typically offers to kill if it has been pushed twice in a row. If this method is called, however, it ignores any prior clicks.

```
(send a-drracket:rep:context set-breakables thread
                                            custodian) → void?
  thread : (or/c thread false/c)
  custodian : (or/c custodian false/c)
```

Calling this method with a thread and a custodian means that the next time the break button is clicked, it will either break the thread or shutdown the custodian.

See also `get-breakables`.

```
(send a-drracket:rep:context update-
running running?) → void?
  running? : any/c
```

This method should update some display in the gui that indicates whether or not evaluation is currently proceeding in the user's world.

```
(drracket:rep:get-welcome-delta) → (is-a?/c style-delta%)
```

Returns a style delta that matches the style and color of the phrase “Welcome to” in the beginning of the interactions window.

```
(drracket:rep:get-dark-green-delta) → (is-a?/c style-delta%)
```

Returns a style delta that matches the style and color of the name of a language in the interactions window.

```
(drracket:rep:get-error-delta) → (is-a?/c style-delta%)
```

Returns a style delta that matches the style and color of errors that get shown in the interactions window.

```
(drracket:rep:get-drs-bindings-keymap) → (is-a?/c keymap%)
```

Returns a keymap that binds various DrRacket-specific keybindings. This keymap is used in the definitions and interactions window.

By default, binds C-x;o to a function that switches the focus between the definitions and interactions windows. Also binds f5 to Execute and f1 to Help Desk.

```
(drracket:rep:current-rep)
 → (or/c false/c (is-a?/c drracket:rep:text%))
```

This is a parameter whose value should not be set by tools. It is initialized to the repl that controls this evaluation in the user’s thread.

It only returns `#f` if the program not running in the context of a repl (eg, the test suite window).

```
(drracket:rep:current-value-port) → (or/c false/c port?)
```

This is a parameter whose value is a port that prints in the REPL in blue. It is used to print the values of toplevel expressions in the REPL.

It is only initialized on the user’s thread.

```
(drracket:rep:after-expression) → (or/c #f (-> any))
(drracket:rep:after-expression top-level-expression) → void?
  top-level-expression : (or/c #f (-> any))
```

This parameter is used by `evaluate-from-port`. When it is a thunk, then DrRacket invokes the thunk on the user's thread as the last thing it does (before cleaning up).

```
(drracket:rep:current-language-settings)
 → drracket:language-configuration:language-settings?
(drracket:rep:current-language-settings language-settings)
 → void?
  language-settings : drracket:language-configuration:language-settings?
```

This parameter is set (on the user's thread) to the `drracket:language-configuration:language-settings` for the currently running language.

```
(drracket:rep:module-language-initial-run) → boolean?
(drracket:rep:module-language-initial-run initial-run?) → void?
  initial-run? : boolean?
```

The value of this parameter is `#t` in the dynamic extent of the call to `evaluate-from-port` that sets up the initial read-eval-print loop (which doesn't run the user's program)

19 drracket:frame

```
drracket:frame:name-message% : class?
  superclass: canvas%
```

This class implements the little filename button in the top-left hand side of DrRacket's frame.

```
(make-object drracket:frame:name-message% parent)
 → (is-a?/c drracket:frame:name-message%)
  parent : (is-a?/c area-container<%>)
```

```
(send a-drracket:frame:name-message set-message name
                                                short-name)
 → void?
  name : (or/c string? false/c)
  short-name : string?
```

Specification: Sets the names that the button shows.

Default implementation: The string *short-name* is the name that is shown on the button and *name* is shown when the button is clicked on, in a separate window. If *name* is #f, a message indicating that the file hasn't been saved is shown.

```
drracket:frame:mixin : (class? . -> . class?)
  argument extends/implements: drracket:frame:basics<%>
                               frame:text-info<%>
                               frame:editor<%>
  result implements: drracket:frame:<%>
```

Provides an implementation of drracket:frame:<%>

```
drracket:frame:basics-mixin : (class? . -> . class?)
  argument extends/implements: frame:standard-menus<%>
  result implements: drracket:frame:basics<%>
```

Use this mixin to establish some common menu items across various DrRacket windows.

```
(send a-drracket:frame:basics edit-menu:between-find-and-
preferences edit-menu)
 → void?
  edit-menu : (is-a?/c menu%)
```

Overrides edit-menu:between-find-and-preferences in frame:standard-menus<%>.

Adds a separator-menu-item%. Next, adds the "Keybindings" menu item to the edit menu. Finally, if the current-eventspace-has-standard-menus? procedure returns #f, creates another separator-menu-item%.

```
(send a-drracket:frame:basics file-menu:between-open-and-
revert file-menu)
 → void?
  file-menu : (is-a?/c menu%)
```

Overrides file-menu:between-open-and-revert in frame:standard-menus<%>.

Adds an "Install .plt File..." menu item, which downloads and installs .plt files from the web, or installs them from the local disk. After that, calls the super method.

```
(send a-drracket:frame:basics file-menu:between-print-and-
close file-menu)
 → void?
  file-menu : (is-a?/c menu%)
```

Overrides file-menu:between-print-and-close in frame:standard-menus<%>.

Calls the super method. Then, creates a menu item for multi-file searching. Finally, adds a separator-menu-item%.

```
(send a-drracket:frame:basics file-menu:new-callback item
                                                      evt)
 → void?
  item : (is-a?/c menu-item%)
  evt : (is-a?/c control-event%)
```

Overrides file-menu:new-callback in frame:standard-menus<%>.

Opens a new, empty DrRacket window.

```
(send a-drracket:frame:basics file-menu:new-
string) → string?
```

Overrides file-menu:new-string in frame:standard-menus<%>.

Returns the empty string.

```
(send a-drracket:frame:basics file-menu:open-callback item
                                                       evt)
 → void?
  item : (is-a?/c menu-item%)
  evt : (is-a?/c control-event%)
```

Overrides `file-menu:open-callback` in `frame:standard-menus<%>`.

Calls `handler:edit-file`.

```
(send a-drracket:frame:basics file-menu:open-
string) → string?
```

Overrides `file-menu:open-string` in `frame:standard-menus<%>`.

Returns the empty string.

```
(send a-drracket:frame:basics get-additional-important-urls)
 → (listof (list string string))
```

Specification: Each string in the result of this method is added as a menu item to DrRacket's "Related Web Sites" menu item. The first string is the name of the menu item and the second string is a url that, when the menu item is chosen, is sent to the user's browser.

Default implementation: Returns the empty list by default.

```
(send a-drracket:frame:basics help-menu:about-callback item
                                                       evt)
 → void?
  item : (is-a?/c menu-item%)
  evt : (is-a?/c control-event%)
```

Overrides `help-menu:about-callback` in `frame:standard-menus<%>`.

Opens an about box for DrRacket.

```
(send a-drracket:frame:basics help-menu:about-string)
 → string?
```

Overrides `help-menu:about-string` in `frame:standard-menus<%>`.

Returns the string `"DrRacket"`.

```
(send a-drracket:frame:basics help-menu:before-about help-
menu)
 → void?
  help-menu : (is-a?/c menu%)
```

Overrides `help-menu:before-about` in `frame:standard-menus<%>`.

Adds the Help Desk menu item and the Welcome to DrRacket menu item.

```
(send a-drracket:frame:basics help-menu:create-about?)
 → boolean?
```

Overrides help-menu:create-about? in frame:standard-menus<%>.

Returns #t.

```
drracket:frame:basics<%> : interface?
  implements: frame:standard-menus<%>
```

This interface is the result of the drracket:frame:basics-mixin

```
drracket:frame:<%> : interface?
  implements: frame:editor<%>
              frame:text-info<%>
              drracket:frame:basics<%>
```

```
(send a-drracket:frame: add-show-menu-items show-
menu) → void?
  show-menu : (is-a?/c menu%)
```

Specification: This method is called during the construction of the View menu. This method is intended to be overridden with the overriding methods adding other Show/Hide menu items to the View menu.

See also set-show-menu-sort-key and get-show-menu. *Default implementation:* Does nothing.

```
(send a-drracket:frame: set-show-menu-sort-key item
                                                key) → void?
  item : (is-a?/c menu-item<%>)
  key : (and/c real? positive?)
```

Controls the ordering of items in the View menu.

The number determines the sorting order and where separators in the menu appear (smaller numbers first).

These are the numbers for many of the View menu items that come built-in to DrRacket:

Toolbar	1
Split	2

Collapse	3
Show Definitions	101
Show Interactions	102
Use Horizontal Layout	103
Show Log	205
Show Tracing	206
Hide Profile	207
Show Program Contour	301
Show Line Numbers	302
Show Module Browser	401

In addition, a separator is inserted for each 100. So, for example, a separator is inserted between Collapse and Show Definitions.

Note that the argument may be a rational number, effectively allowing insertion between any two menu items already in the menu. For this reason, avoid using 0, or any number is that 0 modulo 100.

```
(send a-drracket:frame: get-show-menu) → (is-a?/c menu%)
```

returns the View menu, for use by the update-shown method.

See also add-show-menu-items.

The method (and others) uses the word show to preserve backwards compatibility from when the menu itself was named the Show menu.

```
(send a-drracket:frame: update-shown) → void?
```

Specification: This method is intended to be overridden. It's job is to update the "View" menu to match the state of the visible windows. In the case of the standard DrRacket window, it change the menu items to reflect the visibility of the definitions and interaction editor-canvas%s.

Call this method whenever the state of the show menu might need to change.

See also get-show-menu.

Default implementation: Does nothing.

20 drracket:help-desk

```
(drracket:help-desk:help-desk
 [search-key
  search-context
  parent
  #:language-family language-family])
→ any
 search-key : (or/c #f string?) = #f
 search-context : (or/c #f string? (list/c string? string?))
                = #f
 parent : (or/c (is-a?/c frame%) (is-a?/c dialog%) #f) = #f
 language-family : string? = #f
```

if *search-key* is a string, performs a search in the docs with *search-key* and *search-context*. Otherwise, calls send-language-family-page. The *language-family* argument is passed along in either case.

The search may involve asking the user a question, in which case the dialog with the question uses *parent* as its parent.

```
(drracket:help-desk:goto-plt-license) → void?
```

Opens the user's web browser and points it at the license for PLT software.

21 drracket:eval

```
(drracket:eval:set-basic-parameters
 snipclasses
 [#:gui-modules? gui-modules])
→ void?
 snipclasses : (listof (is-a?/c snip-class%))
 gui-modules : boolean? = #t
```

Sets the parameters that are shared between the repl's initialization and drracket:eval:build-user-eventspace/custodian.

Specifically, it sets these parameters:

- current-namespace has been set to a newly created empty namespace. This namespace has the following modules shared (with namespace-attach-module) from DrRacket's original namespace:
  - racket/base
  - '#%foreign
  - mzlib/pconvert-prop
  - planet/terse-info

  If the gui-modules? parameter is a true value, then these modules are also shared:
  - mred/mred
  - mrlib/cache-image-snip
  - mrlib/image-core
  - mrlib/matrix-snip
- read-curly-brace-as-paren is #t;
- read-square-bracket-as-paren is #t;
- error-print-width is set to 250;
- current-ps-setup is set to a newly created ps-setup% object;
- the exit-handler is set to a parameter that kills the user's custodian; and
- the snip-class-list, returned by get-the-snip-class-list is initialized with all of the snipclasses in DrRacket's eventspace's snip-class-list.

```
(drracket:eval:get-snip-classes)
 → (listof (is-a?/c snip-class%))
```

Returns a list of all of the snipclasses in the current eventspace.

```
(drracket:eval:expand-program  input
                               language-settings
                               eval-compile-time-part?
                               init
                               kill-termination
                               iter
                              [#:gui-modules? gui-modules?])
 → void?
  input : (or/c input-port? drracket:language:text/pos?)
  language-settings : drracket:language-configuration:language-settings?
  eval-compile-time-part? : boolean?
  init : (-> void?)
  kill-termination : (-> void?)
  iter : (-> (or/c eof-object? syntax? (cons/c string? any/c))
             (-> any)
             any)
  gui-modules? : boolean? = #t
```

Use this function to expand the contents of the definitions window for use with external program processing tools.

This function uses drracket:eval:build-user-eventspace/custodian to build the user's environment. The arguments *language-settings*, *init*, *kill-termination*, and *gui-modules?* are passed to drracket:eval:build-user-eventspace/custodian.

The *input* argument specifies the source of the program.

The *eval-compile-time-part?* argument indicates if expand is called or if expand-top-level-with-compile-time-evals is called when the program is expanded. Roughly speaking, if your tool will evaluate each expression itself by calling eval then pass #f. Otherwise, if your tool just processes the expanded program, be sure to pass #t.

This function calls front-end/complete-program to expand the program. Unlike when the Run is clicked, however, it does not call front-end/finished-complete-program.

The first argument to *iter* is the expanded program (represented as syntax) or eof. The *iter* argument is called for each expression in the expanded program and once more with eof, unless an error is raised during expansion. It is called from the user's thread. If an

exception is raised during expansion of the user's program, *iter* is not called. Consider setting the exception-handler during *init* to handle this situation.

The second argument to *iter* is a thunk that continues expanding the rest of the contents of the definitions window. If the first argument to *iter* was eof, this argument is just the primitive void.

See also drracket:eval:expand-program/multiple.

```
(drracket:eval:traverse-program/multiple
 language-settings
 init
 kill-termination
 [#:gui-modules? gui-modules])
→ (-> (or/c input-port? drracket:language:text/pos?)
      (-> (or/c eof-object? syntax? (cons/c string? any/c))
          (-> any)
          any)
      boolean?
      void?)
 language-settings : drracket:language-configuration:language-settings?
 init : (-> void?)
 kill-termination : (-> void?)
 gui-modules : boolean? = #t
```

This function is similar to drracket:eval:expand-program/multiple The only difference is that it does not expand the program in the editor; instead the processing function can decide how to expand the program.

```
(drracket:eval:expand-program/multiple
 language-settings
 eval-compile-time-part?
 init
 kill-termination
 [#:gui-modules? gui-modules?])
→ (-> (or/c input-port? drracket:language:text/pos?)
      (-> (or/c eof-object? syntax? (cons/c string? any/c))
          (-> any)
          any)
      boolean?
      void?)
 language-settings : drracket:language-configuration:language-settings?
 eval-compile-time-part? : boolean?
 init : (-> void?)
 kill-termination : (-> void?)
```

```
  gui-modules? : boolean? = #t
```

This function is just like drracket:eval:expand-program except that it is curried and the second application can be used multiple times. Use this function if you want to initialize the user's thread (and namespace, etc) once but have program text that comes from multiple sources.

The extra boolean argument to the result function determines if drracket:language:language front-end/complete-program<%> or drracket:language:language front-end/interaction<%> is called.

```
(drracket:eval:build-user-eventspace/custodian
 language-settings
 init
 kill-termination
 [#:gui-modules? gui-modules?])
→ eventspace? custodian?
 language-settings : drracket:language-configuration:language-settings?
 init : (-> void?)
 kill-termination : (-> void?)
 gui-modules? : boolean? = #t
```

This function creates a custodian and an eventspace (on the new custodian) to expand the user's program. It does not kill this custodian, but it can safely be shutdown (with custodian-shutdown-all) after the expansion is finished.

It initializes the user's eventspace's main thread with several parameters:

- current-custodian is set to a new custodian.
- In addition, it calls drracket:eval:set-basic-parameters, passing the #:gui-modules? parameter along.

The *language-settings* argument is the current language and its settings. See drracket:language-configuration:language-settings for details on that structure.

If the program is associated with a DrRacket frame, get the frame's language settings from the get-next-settings method of drracket:unit:definitions-text<%>. Also, the most recently chosen language in the language dialog is saved via the framework's preferences. Apply preferences:get to drracket:language-configuration:get-settings-preferences-symbol for that *language-settings*.

The *init* argument is called after the user's parameters are all set, but before the program is run. It is called on the user's thread. The current-directory and current-load-relative-directory parameters are not set, so if there are appropriate directories, the *init* argument is a good place to set them.

The *kill-termination* argument is called when the main thread of the eventspace terminates, no matter if the custodian was shutdown, or the thread was killed. This procedure is also called when the thread terminates normally. This procedure is called from a new, dedicated thread (*i. e.*, not the thread created to do the expansion, nor the thread that drracket:eval:build-user-eventspace/custodian was called from.)

22 drracket:modes

```
(drracket:modes:add-mode
 name
 surrogate
 repl-submit
 matches-language
 [#:intended-to-edit-programs? intended-to-edit-programs?])
→ drracket:modes:mode?
 name : string?
 surrogate : (or/c #f (is-a?/c mode:surrogate-text<%>))
 repl-submit : (-> (is-a?/c drracket:rep:text%) number? boolean?)
 matches-language : (-> (or/c #f (listof string?)) boolean?)
 intended-to-edit-programs? : boolean? = #t
```

Adds a mode to DrRacket. Returns a mode value that identifies the mode.

The first argument, *name*, is the name of the mode, used in DrRacket's GUI to allow the user to select this mode.

The *surrogate* argument is set to the definitions text and the interactions text (via the `mode:host-text set-surrogate<%>` method) whenever this mode is enabled.

The *repl-submit* procedure is called whenever the user types a return in the interactions window. It is passed the interactions editor and the position where the last prompt occurs. If it returns `#t`, the text after the last prompt is treated as a program fragment and evaluated, according to the language settings. If it returns `#f`, the text is assumed to be an incomplete program fragment, and the keystroke is not treated specially.

The *matches-language* predicate is called whenever the language changes. If it returns `#t` this mode is installed. It is passed the list of strings that correspond to the names of the language in the language dialog.

The *intended-to-edit-programs?* boolean indicates if this mode is intended to be for editing programs (as opposed to some other kind of file content). If it is `#f`, online expansion is disabled and DrRacket won't look for `(module` at the front of the buffer to try to guess the intended filename.

Modes are tested in the opposite order that they are added. That is, the last mode to be added gets tested first when the filename changes or when the language changes.

See also `drracket:modes:get-modes`.

Changed in version 1.1 of package `drracket-core-lib`: Added the *intended-to-edit-programs?* argument.

```
(struct drracket:modes:mode (name
                             surrogate
                             repl-submit
                             matches-language
                             intended-to-edit-programs?))
  name : string?
  surrogate : (or/c #f (is-a?/c mode:surrogate-text<%>))
  repl-submit : (-> (is-a?/c drracket:rep:text%) number? boolean?)
  matches-language : (-> (or/c #f (listof string?)) boolean?)
  intended-to-edit-programs? : boolean?
```

See drracket:modes:add-mode for details on modes.

Changed in version 1.1 of package drracket-core-lib: Added the intended-to-edit-programs? field.

```
drracket:modes:struct:mode : struct-type?
```

An alias for struct:drracket:modes:mode.

```
(drracket:modes:get-modes) → (listof drracket:modes:mode?)
```

Returns all of the modes currently added to DrRacket.

Note that the *surrogate* field of the mode corresponding to the module language does not take into account the definitions-text-surrogate, so it may not be the actual class used directly in DrRacket, even when the mode is active.

See also drracket:modes:add-mode.

23 drracket:module-language-tools

```
(drracket:module-language-tools:add-opt-out-toolbar-button
 make-button
 id
 [#:number number])
→ void?
 make-button : (-> (is-a?/c top-level-window<%>)
                   (is-a?/c area-container<%>)
                   (is-a?/c switchable-button%))
 id : symbol?
 number : (or/c real? #f) = #f
```

Call this function to add another button to DrRacket's toolbar. When buttons are added this way, DrRacket monitors the `#lang` line at the top of the file; when it changes DrRacket queries the language to see if this button should be included. These buttons are "opt out", meaning that if the language doesn't explicitly ask to not have this button (or all such buttons), the button will appear. See `drracket:opt-out-toolbar-buttons` for more information.

The *number* argument is the same as the *number* argument to `register-toolbar-button`.

```
(drracket:module-language-tools:add-opt-in-toolbar-button
 make-button
 id
 [#:number number])
→ void?
 make-button : (-> (is-a?/c top-level-window<%>)
                   (is-a?/c area-container<%>)
                   (is-a?/c switchable-button%))
 id : symbol?
 number : (or/c real? #f) = #f
```

Like `drracket:module-language-tools:add-opt-out-toolbar-button`, but for buttons that should not be enabled by default, but instead explicitly opted in by languages via `drracket:opt-in-toolbar-buttons`.

Added in version 1.6 of package `drracket-core-lib`.

```
(drracket:module-language-tools:add-online-expansion-handler
 mod-path
 id
 local-handler)
```

```
→ void?
 mod-path : path-string?
 id : symbol?
 local-handler : (-> (is-a?/c drracket:unit:definitions-text<%>)
                     any/c
                     any)
```

Registers a pair of procedures with DrRacket's online expansion machinery. (See also drracket:module-language-tools:add-online-expansion-monitor.)

The procedure *id* from *mod-path* is loaded by dynamic-require into a specially designed separate place. When DrRacket detects that the editor has been modified, it sends the contents of the editor over to that separate place, expands the program there, and then supplies the fully expanded object to that first procedure. (The procedure is called in the same context as the expansion process.)

If the expansion raises an exception, then that exception is supplied as the first argument instead of the syntax object. If a non-exn? is raised, or if the expansion process is terminated (e.g. via custodian-shutdown-all called during expansion), then the expansion monitor is not notified.

The contract for that procedure is

```
(-> (or/c syntax? exn?) path? any/c custodian?
    any)
```

There are three other arguments:

- The path? argument is the path that was the current-directory when the code was expanded. This directory should be used as the current-directory when resolving module paths obtained from the syntax object.
- The third argument is the source object used in the syntax objects that come from the definitions window in DrRacket. It may be a path (if the file was saved), but it also might not be. Use equal? to compare it with the syntax-source field of syntax objects to determine if they come from the definitions window.
- Note that the thread that calls this procedure may be killed at any time: DrRacket may kill it when the user types in the buffer (in order to start a new expansion), but bizarro code may also create a separate thread during expansion that lurks around and then mutates arbitrary things.

  Some code, however, should be longer running, surviving such custodian shutdowns. To support this, the procedure called in the separate place is supplied with a more powerful custodian that is not shut down.

The result of the procedure is expected to be something that can be sent across a place-channel, which is then sent back to the original place where DrRacket itself is running and passed to the *local-handler* argument. At this point, the only code running is trusted code (DrRacket itself and other tools), but any long running computations may freeze DrRacket's GUI, since this procedure is invoked on DrRacket's eventspace's handler thread.

```
(drracket:module-language-tools:add-online-expansion-monitor
 mod-path
 id
 local-handler)
→ void?
 mod-path : path-string?
 id : symbol?
 local-handler : (-> (is-a?/c drracket:unit:definitions-text<%>)
                     (or/c drracket:module-language-tools:start?
                           any/c)
                     any)
```

Registers a pair of procedures with DrRacket's online expansion machinery.

Like drracket:module-language-tools:add-online-expansion-handler, the first two arguments specify a procedure that is called in the separate place designated for expansion.

The procedure is called before expansion starts and once it returns, expansion begins. The procedure should match this contract:

```
(-> (-> any/c void?)
    path? any/c custodian?
    any)
```

The first argument is a function that transmits its argument back to the DrRacket place, send it to the *local-handler* argument. The other three arguments are the same as the corresponding procedure used by drracket:module-language-tools:add-online-expansion-handler.

The expectation is that this procedure creates a thread and monitors the expansion process, sending back information to the main place while expansion is progressing.

The *local-handler* procedure is called each time the (-> any/c void?) procedure (described just above) is called. It is also called each time an expansion starts; it receives a value that returns #t from drracket:module-language-tools:start? in that case.

To help with debugging, DrRacket logs progress and recovers from some errors that happen when running the handler procedures. To monitor its progress, monitor the 'debug level of the logger with the topic 'drracket-background-compilation.

```
(drracket:module-language-tools:start? val) → boolean?
  val : any/c
```

Returns #t if this is a special (unique) value, used as discussed in drracket:module-language-tools:add-online-expansion-monitor. Returns #f otherwise.

```
(drracket:module-language-tools:register-online-expansion-pref func)
 → void?
  func : (-> (is-a?/c vertical-panel%) void?)
```

Registers *func* so that it is called while building the preferences panel. The function is passed a panel that contains other configuration controls for online expansion.

```
(drracket:module-language-tools:done? val) → boolean?
  val : any/c
```

Returns #t for drracket:module-language-tools:done and #f otherwise.

```
drracket:module-language-tools:done
 : drracket:module-language-tools:done?
```

Used to inform a monitor-based handler that the online expansion has finished.

24 drracket:module-language

```
drracket:module-language:module-language<%> : interface?
```

The only language that implements this interface is DrRacket's "Use the language declared in the source" language.

```
(send a-drracket:module-language:module-language get-users-
language-name)
 → string
```

Returns the name of the language that is declared in the source, as a string.

```
drracket:module-language-tools:definitions-text<%> : interface?
```

```
(send a-drracket:module-language-tools:definitions-
text move-to-new-language)
 → void?
```

This method is called when a new language is evident in the definitions window (by editing the `#lang` line.

```
(send a-drracket:module-language-tools:definitions-text get-
in-module-language?)
 → boolean?
```

Returns `#t` when the current language setting (from the language dialog) is "The Racket Language".

```
drracket:module-language-tools:tab<%> : interface?
```

This interface signals an implementation of a tab that specially handles programs beginning with `#lang`.

```
drracket:module-language-tools:frame<%> : interface?
```

This interface signals an implementation of a frame that specially handles programs beginning with `#lang`.

```
drracket:module-language-tools:definitions-text-mixin : (class? . -> .
class?)
```
argument extends/implements: `text:basic<%>`
`racket:text<%>`
`drracket:unit:definitions-text<%>`
result implements: `drracket:module-language-tools:definitions-text<%>`

```
drracket:module-language-tools:frame-mixin : (class? . -> . class?)
```
argument extends/implements: `drracket:unit:frame<%>`
result implements: `drracket:module-language-tools:frame<%>`

```
drracket:module-language-tools:tab-mixin : (class? . -> . class?)
```
argument extends/implements: `drracket:unit:tab<%>`
result implements: `drracket:module-language-tools:tab<%>`

```
(drracket:module-language:add-module-language) → any
```

Adds the module language to DrRacket. This is called during DrRacket's startup.

```
(drracket:module-language:module-language-put-file-mixin super%)
 → (implementation?/c text:basic<%>)
  super% : (implementation?/c text:basic<%>)
```

Extends *super%* by overriding the put-file method to use a default name from the buffer, if the buffer contains something like `(module name ...)`.

25 drracket:tracing

```
drracket:tracing:tab-mixin : (class? . -> . class?)
  argument extends/implements: drracket:unit:tab<%>
                               drracket:rep:context<%>
```

Tracks function call tracing information.

```
drracket:tracing:frame-mixin : (class? . -> . class?)
  argument extends/implements: drracket:frame:<%>
                               drracket:unit:frame<%>
```

Tracks function call tracing information.

```
(drracket:tracing:annotate stx) → syntax?
  stx : syntax?
```

Call this function to add tracing annotations to the a fully-expanded expression. When the program runs, DrRacket will pop open the tracing window to display the trace.

26 drracket:init

```
(drracket:init:original-error-display-handler message
                                              exn)    → any
  message : string?
  exn : any/c
```

This is the error-display-handler installed at the time that DrRacket starts up.

DrRacket sets the error-display-handler to one that shows an “Internal Error” dialog box.

27 Backwards Compatibility

This section lists the bindings that begin with `drscheme:` provided by the tools library; they are here for backwards compatibility and to provide links to the `drracket:` versions of the names.

`drscheme:tool^` : `any/c`

This is provided for backwards compatibility; new code should use `drracket:tool^` instead.

`drscheme:tool-exports^` : `any/c`

This is provided for backwards compatibility; new code should use `drracket:tool-exports^` instead.

`drscheme:debug:profile-definitions-text-mixin` : `any/c`

This is provided for backwards compatibility; new code should use `drracket:debug:profile-definitions-text-mixin` instead.

`drscheme:debug:profile-tab-mixin` : `any/c`

This is provided for backwards compatibility; new code should use `drracket:debug:profile-tab-mixin` instead.

`drscheme:debug:profile-unit-frame-mixin` : `any/c`

This is provided for backwards compatibility; new code should use `drracket:debug:profile-unit-frame-mixin` instead.

`drscheme:debug:test-coverage-interactions-text-mixin` : `any/c`

This is provided for backwards compatibility; new code should use `drracket:debug:test-coverage-interactions-text-mixin` instead.

`drscheme:debug:test-coverage-definitions-text-mixin` : `any/c`

This is provided for backwards compatibility; new code should use `drracket:debug:test-coverage-definitions-text-mixin` instead.

drscheme:debug:test-coverage-tab-mixin : any/c

This is provided for backwards compatibility; new code should use drracket:debug:test-coverage-tab-mixin instead.

drscheme:debug:test-coverage-frame-mixin : any/c

This is provided for backwards compatibility; new code should use drracket:debug:test-coverage-frame-mixin instead.

drscheme:unit:tab% : any/c

This is provided for backwards compatibility; new code should use drracket:unit:tab% instead.

drscheme:unit:frame% : any/c

This is provided for backwards compatibility; new code should use drracket:unit:frame% instead.

drscheme:unit:definitions-canvas% : any/c

This is provided for backwards compatibility; new code should use drracket:unit:definitions-canvas% instead.

drscheme:unit:get-definitions-text% : any/c

This is provided for backwards compatibility; new code should use drracket:unit:get-definitions-text% instead.

drscheme:unit:interactions-canvas% : any/c

This is provided for backwards compatibility; new code should use drracket:unit:interactions-canvas% instead.

drscheme:rep:drs-bindings-keymap-mixin : any/c

This is provided for backwards compatibility; new code should use drracket:rep:drs-bindings-keymap-mixin instead.

drscheme:rep:text% : any/c

This is provided for backwards compatibility; new code should use drracket:rep:text% instead.

drscheme:rep:text<%> : any/c

This is provided for backwards compatibility; new code should use drracket:rep:text<%> instead.

drscheme:frame:mixin : any/c

This is provided for backwards compatibility; new code should use drracket:frame:mixin instead.

drscheme:frame:basics-mixin : any/c

This is provided for backwards compatibility; new code should use drracket:frame:basics-mixin instead.

drscheme:language:language<%> : any/c

This is provided for backwards compatibility; new code should use drracket:language:language<%> instead.

drscheme:language:module-based-language<%> : any/c

This is provided for backwards compatibility; new code should use drracket:language:module-based-language<%> instead.

drscheme:language:simple-module-based-language<%> : any/c

This is provided for backwards compatibility; new code should use drracket:language:simple-module-based-language<%> instead.

drscheme:language:simple-module-based-language% : any/c

This is provided for backwards compatibility; new code should use drracket:language:simple-module-based-language% instead.

drscheme:language:simple-module-based-language->module-based-language-mixin
 : any/c

This is provided for backwards compatibility; new code should use drracket:language:simple-module-based-language->module-based-language-mixin instead.

drscheme:language:module-based-language->language-mixin : any/c

This is provided for backwards compatibility; new code should use drracket:language:module-based-language->language-mixin instead.

drscheme:tracing:tab-mixin : any/c

This is provided for backwards compatibility; new code should use drracket:tracing:tab-mixin instead.

drscheme:tracing:frame-mixin : any/c

This is provided for backwards compatibility; new code should use drracket:tracing:frame-mixin instead.

drscheme:module-language:module-language<%> : any/c

This is provided for backwards compatibility; new code should use drracket:module-language:module-language<%> instead.

drscheme:module-language-tools:frame-mixin : any/c

This is provided for backwards compatibility; new code should use drracket:module-language-tools:frame-mixin instead.

drscheme:module-language-tools:tab-mixin : any/c

This is provided for backwards compatibility; new code should use drracket:module-language-tools:tab-mixin instead.

drscheme:module-language-tools:definitions-text-mixin : any/c

This is provided for backwards compatibility; new code should use drracket:module-language-tools:definitions-text-mixin instead.

drscheme:frame:basics<%> : any/c

This is provided for backwards compatibility; new code should use drracket:frame:basics<%> instead.

drscheme:frame:<%> : any/c

This is provided for backwards compatibility; new code should use drracket:frame:<%> instead.

drscheme:unit:frame<%> : any/c

This is provided for backwards compatibility; new code should use drracket:unit:frame<%> instead.

drscheme:unit:definitions-text<%> : any/c

This is provided for backwards compatibility; new code should use drracket:unit:definitions-text<%> instead.

drscheme:unit:tab<%> : any/c

This is provided for backwards compatibility; new code should use drracket:unit:tab<%> instead.

drscheme:rep:context<%> : any/c

This is provided for backwards compatibility; new code should use drracket:rep:context<%> instead.

drscheme:module-language-tools:definitions-text<%> : any/c

This is provided for backwards compatibility; new code should use drracket:module-language-tools:definitions-text<%> instead.

```
drscheme:module-language-tools:tab<%> : any/c
```

This is provided for backwards compatibility; new code should use drracket:module-language-tools:tab<%> instead.

```
drscheme:module-language-tools:frame<%> : any/c
```

This is provided for backwards compatibility; new code should use drracket:module-language-tools:frame<%> instead.

```
drscheme:debug:error-display-handler/stacktrace
 : (->* (string? any/c)
        ((or/c false/c (listof srcloc?))
         #:definitions-text (or/c #f (is-a?/c drracket:unit:definitions-text<%>))
         #:interactions-text (or/c #f (is-a?/c drracket:rep:text<%>)))

        any/c)
```

This binding provided for backwards compatibility; new code should use drracket:debug:error-display-handler/stacktrace instead.

```
drscheme:debug:make-debug-error-display-handler
 : (-> (-> string? (or/c any/c exn?) any)
       (-> string? (or/c any/c exn?) any))
```

This binding provided for backwards compatibility; new code should use drracket:debug:make-debug-error-display-handler instead.

```
drscheme:debug:hide-backtrace-window : (-> void?)
```

This binding provided for backwards compatibility; new code should use drracket:debug:hide-backtrace-window instead.

```
drscheme:debug:add-prefs-panel : (-> void?)
```

This binding provided for backwards compatibility; new code should use drracket:debug:add-prefs-panel instead.

```
drscheme:debug:make-debug-compile-handler
 : (-> (-> any/c boolean? compiled-expression?)
       (-> any/c boolean? compiled-expression?))
```

This binding provided for backwards compatibility; new code should use drracket:debug:make-debug-compile-handler instead.

```
drscheme:debug:make-debug-eval-handler
 : (-> (-> any/c any) (-> any/c any))
```

This binding provided for backwards compatibility; new code should use drracket:debug:make-debug-eval-handler instead.

```
drscheme:debug:test-coverage-enabled : (parameter/c boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:debug:test-coverage-enabled instead.

```
drscheme:debug:test-coverage-on-style-name : string?
```

This binding provided for backwards compatibility; new code should use drracket:debug:test-coverage-on-style-name instead.

```
drscheme:debug:test-coverage-off-style-name : string?
```

This binding provided for backwards compatibility; new code should use drracket:debug:test-coverage-off-style-name instead.

```
drscheme:debug:profiling-enabled : (parameter/c boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:debug:profiling-enabled instead.

```
drscheme:debug:bug-info->ticket-url
 : (-> (listof (cons/c symbol? (or/c #f string?)))
       url?)
```

This binding provided for backwards compatibility; new code should use drracket:debug:bug-info->ticket-url instead.

```
drscheme:debug:small-planet-bitmap : (is-a?/c bitmap%)
```

This binding provided for backwards compatibility; new code should use drracket:debug:small-planet-bitmap instead.

```
drscheme:debug:open-and-highlight-in-file
 : (->* ((or/c srcloc? (listof srcloc?)))
        ((or/c #f (cons/c (λ (x) (and (weak-box? x)
                                      (let ([v (weak-box-value x)])
                                        (or (not v)
                                            (is-a? v editor<%>)))))
                          number?)))
        void?)
```

This binding provided for backwards compatibility; new code should use drracket:debug:open-and-highlight-in-file instead.

```
drscheme:debug:show-backtrace-window/edition-pairs
 : (-> string?
       (listof srcloc?)
       (listof
        (or/c
         #f
         (cons/c (λ (x)
                   (and (weak-box? x)
                        (let ([v (weak-box-value x)])
                          (or (not v)
                              (is-a? v editor<%>)))))
                 number?)))
       (or/c #f (is-a?/c drracket:unit:definitions-text<%>))
       (or/c #f (is-a?/c drracket:rep:text<%>))
       void?)
```

This binding provided for backwards compatibility; new code should use drracket:debug:show-backtrace-window/edition-pairs instead.

```
drscheme:debug:show-backtrace-window/edition-pairs/two
```

```
: (-> string?
     (listof srcloc?)
     (listof
      (or/c
       #f
       (cons/c (λ (x)
                 (and (weak-box? x)
                      (let ([v (weak-box-value x)])
                        (or (not v)
                            (is-a? v editor<%>)))))
               number?)))
     (listof srcloc?)
     (listof
      (or/c
       #f
       (cons/c (λ (x)
                 (and (weak-box? x)
                      (let ([v (weak-box-value x)])
                        (or (not v)
                            (is-a? v editor<%>)))))
               number?)))
     (or/c #f (is-a?/c drracket:unit:definitions-text<%>))
     (or/c #f (is-a?/c drracket:rep:text<%>))
     void?)
```

This binding provided for backwards compatibility; new code should use drracket:debug:show-backtrace-window/edition-pairs/two instead.

```
drscheme:debug:get-error-color : (-> (is-a?/c color%))
```

This binding provided for backwards compatibility; new code should use drracket:debug:get-error-color instead.

```
drscheme:debug:get-error-color-name
 : (-> color-prefs:color-scheme-color-name?)
```

This binding provided for backwards compatibility; new code should use drracket:debug:get-error-color-name instead.

```
drscheme:debug:show-backtrace-window
```

```
 : (->* (string?
         (or/c exn?
               (listof srcloc?)))
        ((or/c #f (is-a?/c drracket:rep:text<%>))
         (or/c #f (is-a?/c drracket:unit:definitions-text<%>)))
        void?)
```

This binding provided for backwards compatibility; new code should use drracket:debug:show-backtrace-window instead.

```
drscheme:eval:set-basic-parameters
 : (->* ((listof (is-a?/c snip-class%)))
        (#:gui-modules? boolean?)
        void?)
```

This binding provided for backwards compatibility; new code should use drracket:eval:set-basic-parameters instead.

```
drscheme:eval:get-snip-classes
 : (-> (listof (is-a?/c snip-class%)))
```

This binding provided for backwards compatibility; new code should use drracket:eval:get-snip-classes instead.

```
drscheme:eval:expand-program
 : (->* ((or/c input-port? drracket:language:text/pos?)
         drracket:language-configuration:language-settings?
         boolean?
         (-> void?)
         (-> void?)
         (-> (or/c eof-object? syntax? (cons/c string? any/c))
             (-> any)
             any))
        (#:gui-modules? boolean?)
        void?)
```

This binding provided for backwards compatibility; new code should use drracket:eval:expand-program instead.

```
drscheme:eval:traverse-program/multiple
```

```
: (->* (drracket:language-configuration:language-settings?
        (-> void?)
        (-> void?))
       (#:gui-modules? boolean?)
       (-> (or/c input-port? drracket:language:text/pos?)
           (-> (or/c eof-object? syntax? (cons/c string? any/c))
               (-> any)
               any)
           boolean?
           void?))
```

This binding provided for backwards compatibility; new code should use `drracket:eval:traverse-program/multiple` instead.

```
drscheme:eval:expand-program/multiple
: (->* (drracket:language-configuration:language-settings?
        boolean?
        (-> void?)
        (-> void?))
       (#:gui-modules? boolean?)
       (-> (or/c input-port? drracket:language:text/pos?)
           (-> (or/c eof-object? syntax? (cons/c string? any/c))
               (-> any)
               any)
           boolean?
           void?))
```

This binding provided for backwards compatibility; new code should use `drracket:eval:expand-program/multiple` instead.

```
drscheme:eval:build-user-eventspace/custodian
: (->* (drracket:language-configuration:language-settings?
        (-> void?)
        (-> void?))
       (#:gui-modules? boolean?)
       (values eventspace? custodian?))
```

This binding provided for backwards compatibility; new code should use `drracket:eval:build-user-eventspace/custodian` instead.

```
drscheme:get/extend:extend-unit-frame
: (->i ([mixin (make-mixin-contract drracket:unit:frame%)])
       ([before boolean?]
        #:name-for-changes [name-for-changes (or/c #f symbol?)])
       [result void?])
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:extend-unit-frame instead.

```
drscheme:get/extend:get-unit-frame
 : (-> (subclass?/c drracket:unit:frame%))
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:get-unit-frame instead.

```
drscheme:get/extend:extend-tab
 : (->i ([mixin (make-mixin-contract drracket:unit:tab<%>)])
        ([before boolean?]
         #:name-for-changes [name-for-changes (or/c #f symbol?)])
        [result void?])
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:extend-tab instead.

```
drscheme:get/extend:get-tab
 : (-> (implementation?/c drracket:unit:tab<%>))
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:get-tab instead.

```
drscheme:get/extend:extend-definitions-text
 : (->i ([mixin (make-mixin-contract drracket:unit:definitions-text<%>
                                     editor:standard-style-list<%>
                                     editor:info<%>
                                     racket:text<%>
                                     text:all-string-snips<%>
                                     text:file<%>
                                     text:info<%>
                                     text:wide-snip<%>)])
        ([before boolean?]
         #:name-for-changes [name-for-changes (or/c #f symbol?)])
        [result void?])
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:extend-definitions-text instead.

```
drscheme:get/extend:get-definitions-text
```

```
: (-> (and/c (implementation?/c drracket:unit:definitions-text<%>)
             (implementation?/c editor:standard-style-list<%>)
             (implementation?/c editor:info<%>)
             (implementation?/c racket:text<%>)
             (implementation?/c text:all-string-snips<%>)
             (implementation?/c text:file<%>)
             (implementation?/c text:info<%>)
             (implementation?/c text:wide-snip<%>)))
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:get-definitions-text instead.

```
drscheme:get/extend:extend-interactions-text
: (->i ([mixin (make-mixin-contract drracket:rep:text<%>)])
       ([before boolean?]
        #:name-for-changes [name-for-changes (or/c #f symbol?)])
       [result void?])
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:extend-interactions-text instead.

```
drscheme:get/extend:get-interactions-text
: (-> (implementation?/c drracket:rep:text<%>))
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:get-interactions-text instead.

```
drscheme:get/extend:extend-definitions-canvas
: (->i ([mixin (make-mixin-contract drracket:unit:definitions-canvas%)])
       ([before boolean?]
        #:name-for-changes [name-for-changes (or/c #f symbol?)])
       [result void?])
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:extend-definitions-canvas instead.

```
drscheme:get/extend:get-definitions-canvas
: (-> (subclass?/c drracket:unit:definitions-canvas%))
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:get-definitions-canvas instead.

```
drscheme:get/extend:extend-interactions-canvas
 : (->i ([mixin (make-mixin-contract drracket:unit:interactions-canvas%)])
        ([before boolean?]
         #:name-for-changes [name-for-changes (or/c #f symbol?)])
        [result void?])
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:extend-interactions-canvas instead.

```
drscheme:get/extend:get-interactions-canvas
 : (-> (subclass?/c drracket:unit:interactions-canvas%))
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:get-interactions-canvas instead.

```
drscheme:get/extend:disallow-re-extension! : (-> void?)
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:disallow-re-extension! instead.

```
drscheme:get/extend:allow-re-extension! : (-> void?)
```

This binding provided for backwards compatibility; new code should use drracket:get/extend:allow-re-extension! instead.

```
drscheme:help-desk:help-desk
 : (->* ()
        ((or/c #f string?)
         (or/c #f string? (list/c string? string?))
         (or/c (is-a?/c frame%) (is-a?/c dialog%) #f)
         #:language-family string?)
        any)
```

This binding provided for backwards compatibility; new code should use drracket:help-desk:help-desk instead.

```
drscheme:help-desk:goto-plt-license : (-> void?)
```

This binding provided for backwards compatibility; new code should use drracket:help-desk:goto-plt-license instead.

```
drscheme:language-configuration:get-languages
 : (-> (listof (is-a?/c drracket:language:language<%>)))
```

This binding provided for backwards compatibility; new code should use drracket:language-configuration:get-languages instead.

```
drscheme:language-configuration:add-language
 : (->* ((and/c (is-a?/c drracket:language:language<%>)
                drracket:language:object/c))
        (#:allow-executable-creation? boolean?)
        void?)
```

This binding provided for backwards compatibility; new code should use drracket:language-configuration:add-language instead.

```
drscheme:language-configuration:get-settings-preferences-symbol
 : (-> symbol?)
```

This binding provided for backwards compatibility; new code should use drracket:language-configuration:get-settings-preferences-symbol instead.

```
drscheme:language-configuration:language-settings?
 : (-> any/c boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:language-configuration:language-settings? instead.

```
drscheme:language-configuration:language-settings-language
 : (-> drracket:language-configuration:language-settings? (or/c (is-a?/c drracket:language:la
```

This binding provided for backwards compatibility; new code should use drracket:language-configuration:language-settings-language instead.

```
drscheme:language-configuration:language-settings-settings
 : (-> drracket:language-configuration:language-settings? any/c)
```

This binding provided for backwards compatibility; new code should use drracket:language-configuration:language-settings-settings instead.

```
drscheme:language-configuration:struct:language-settings
 : struct-type?
```

This binding provided for backwards compatibility; new code should use drracket:language-configuration:struct:language-settings instead.

```
drscheme:language-configuration:make-language-settings
 : procedure?
```

This binding provided for backwards compatibility; new code should use drracket:language-configuration:make-language-settings instead.

```
drscheme:language-configuration:language-dialog
 : (->* (boolean? drracket:language-configuration:language-settings?)
        ((or/c false/c (is-a?/c top-level-window<%>)))
        (or/c false/c drracket:language-configuration:language-settings?))
```

This binding provided for backwards compatibility; new code should use drracket:language-configuration:language-dialog instead.

```
drscheme:language-configuration:fill-language-dialog
 : (->*
    ((is-a?/c vertical-panel%)
     (is-a?/c area-container<%>)
     drracket:language-configuration:language-settings?)
    ((or/c false/c (is-a?/c top-level-window<%>))
     (-> symbol? void?))
    (values (-> (is-a?/c drracket:language:language<%>))
            (-> any/c)
            (-> any/c (is-a?/c mouse-event%) any)))
```

This binding provided for backwards compatibility; new code should use drracket:language-configuration:fill-language-dialog instead.

```
drscheme:language:register-capability
 : (->i ([s symbol?]
         [the-contract contract?]
         [default (the-contract) the-contract])
        ()
        [res void?])
```

This binding provided for backwards compatibility; new code should use drracket:language:register-capability instead.

```
drscheme:language:capability-registered? : (-> symbol? boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:language:capability-registered? instead.

```
drscheme:language:get-capability-default
 : (->i ([s (and/c symbol? drracket:language:capability-registered?)])
        ()
        [res (s) (drracket:language:get-capability-contract s)])
```

This binding provided for backwards compatibility; new code should use drracket:language:get-capability-default instead.

```
drscheme:language:get-capability-contract
 : (-> (and/c symbol? drracket:language:capability-registered?)
       contract?)
```

This binding provided for backwards compatibility; new code should use drracket:language:get-capability-contract instead.

```
drscheme:language:add-snip-value
 : (->* ((-> any/c boolean?)
         (-> any/c (is-a?/c snip%)))
        ((-> any/c))
        void?)
```

This binding provided for backwards compatibility; new code should use drracket:language:add-snip-value instead.

```
drscheme:language:extend-language-interface
 : (-> interface?
       (make-mixin-contract drracket:language:language<%>)
       void?)
```

This binding provided for backwards compatibility; new code should use drracket:language:extend-language-interface instead.

```
drscheme:language:get-default-mixin
 : (-> (make-mixin-contract drracket:language:language<%>))
```

This binding provided for backwards compatibility; new code should use drracket:language:get-default-mixin instead.

```
drscheme:language:get-language-extensions
 : (-> (listof interface?))
```

This binding provided for backwards compatibility; new code should use drracket:language:get-language-extensions instead.

```
drscheme:language:put-executable
 : ((is-a?/c top-level-window<%>)
    path?
    (or/c boolean? 'launcher 'standalone 'distribution)
    boolean?
    string?
    . -> . (or/c false/c path?))
```

This binding provided for backwards compatibility; new code should use drracket:language:put-executable instead.

```
drscheme:language:create-executable-gui
 : ((or/c #f (is-a?/c top-level-window<%>))
    (or/c #f string?)
    (or/c #t 'launcher 'standalone 'distribution)
    (or/c #t 'mzscheme 'mred)
    . -> .
    (or/c #f
          (list/c (or/c 'no-show 'launcher 'stand-alone 'distribution)
                  (or/c 'no-show 'mred 'mzscheme)
                  string?
                  (listof (cons/c symbol? any/c)))))
```

This binding provided for backwards compatibility; new code should use drracket:language:create-executable-gui instead.

```
drscheme:language:create-module-based-stand-alone-executable
 : ((or/c path? string?)
    (or/c path? string?) any/c any/c any/c boolean? boolean?
    . -> .
    void?)
```

This binding provided for backwards compatibility; new code should use drracket:language:create-module-based-stand-alone-executable instead.

```
drscheme:language:create-module-based-distribution
 : ((or/c path? string?)
    (or/c path? string?) any/c any/c any/c boolean? boolean?
    . -> .
    void?)
```

This binding provided for backwards compatibility; new code should use drracket:language:create-module-based-distribution instead.

```
drscheme:language:create-distribution-for-executable
 : ((or/c path? string?)
    boolean?
    (-> path? void?)
    . -> .
    void?)
```

This binding provided for backwards compatibility; new code should use drracket:language:create-distribution-for-executable instead.

```
drscheme:language:create-module-based-launcher
 : ((or/c path? string?) (or/c path? string?) any/c any/c any/c boolean? boolean?
                         . -> .
                         void?)
```

This binding provided for backwards compatibility; new code should use drracket:language:create-module-based-launcher instead.

```
drscheme:language:simple-module-based-language-convert-value
 : (-> any/c drracket:language:simple-settings? any)
```

This binding provided for backwards compatibility; new code should use drracket:language:simple-module-based-language-convert-value instead.

```
drscheme:language:setup-printing-parameters
 : (-> (-> any) drracket:language:simple-settings? (or/c number? 'infinity) any)
```

This binding provided for backwards compatibility; new code should use drracket:language:setup-printing-parameters instead.

```
drscheme:language:make-setup-printing-parameters
 : (-> (-> (-> any) drracket:language:simple-settings? (or/c number? 'infinity) any))
```

This binding provided for backwards compatibility; new code should use drracket:language:make-setup-printing-parameters instead.

```
drscheme:language:text/pos? : (-> any/c boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:language:text/pos? instead.

```
drscheme:language:text/pos-text
 : (-> drracket:language:text/pos? (is-a?/c text%))
```

This binding provided for backwards compatibility; new code should use drracket:language:text/pos-text instead.

```
drscheme:language:text/pos-start
 : (-> drracket:language:text/pos? exact-nonnegative-integer?)
```

This binding provided for backwards compatibility; new code should use drracket:language:text/pos-start instead.

```
drscheme:language:text/pos-end
 : (-> drracket:language:text/pos? exact-nonnegative-integer?)
```

This binding provided for backwards compatibility; new code should use drracket:language:text/pos-end instead.

```
drscheme:language:make-text/pos : procedure?
```

This binding provided for backwards compatibility; new code should use drracket:language:make-text/pos instead.

```
drscheme:language:struct:text/pos : struct-type?
```

This binding provided for backwards compatibility; new code should use drracket:language:struct:text/pos instead.

```
drscheme:language:simple-settings? : (-> any/c boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:language:simple-settings? instead.

```
drscheme:language:simple-settings-case-sensitive
 : (-> drracket:language:simple-settings? boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:language:simple-settings-case-sensitive instead.

```
drscheme:language:simple-settings-printing-style
 : (-> drracket:language:simple-settings? (or/c 'constructor 'quasiquote 'write 'trad-write '
```

This binding provided for backwards compatibility; new code should use drracket:language:simple-settings-printing-style instead.

```
drscheme:language:simple-settings-fraction-style
 : (-> drracket:language:simple-settings? (or/c 'mixed-fraction 'mixed-fraction-e 'repeating-
```

This binding provided for backwards compatibility; new code should use drracket:language:simple-settings-fraction-style instead.

```
drscheme:language:simple-settings-show-sharing
 : (-> drracket:language:simple-settings? boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:language:simple-settings-show-sharing instead.

```
drscheme:language:simple-settings-insert-newlines
 : (-> drracket:language:simple-settings? boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:language:simple-settings-insert-newlines instead.

```
drscheme:language:simple-settings-annotations
 : (-> drracket:language:simple-settings? (or/c 'none 'debug 'debug/profile 'test-coverage))
```

This binding provided for backwards compatibility; new code should use drracket:language:simple-settings-annotations instead.

```
drscheme:language:make-simple-settings : procedure?
```

This binding provided for backwards compatibility; new code should use drracket:language:make-simple-settings instead.

```
drscheme:language:struct:simple-settings : struct-type?
```

This binding provided for backwards compatibility; new code should use drracket:language:struct:simple-settings instead.

```
drscheme:language:simple-settings->vector
 : (drracket:language:simple-settings? . -> . vector?)
```

This binding provided for backwards compatibility; new code should use drracket:language:simple-settings->vector instead.

```
drscheme:modes:add-mode
 : (->* (string?
         (or/c #f (is-a?/c mode:surrogate-text<%>))
         (-> (is-a?/c drracket:rep:text%) number? boolean?)
         (-> (or/c #f (listof string?)) boolean?))
        (#:intended-to-edit-programs? boolean?)
        drracket:modes:mode?)
```

This binding provided for backwards compatibility; new code should use drracket:modes:add-mode instead.

```
drscheme:modes:mode? : (-> any/c boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:modes:mode? instead.

```
drscheme:modes:mode-name : (-> drracket:modes:mode? string?)
```

This binding provided for backwards compatibility; new code should use drracket:modes:mode-name instead.

```
drscheme:modes:mode-surrogate
 : (-> drracket:modes:mode? (or/c #f (is-a?/c mode:surrogate-text<%>)))
```

This binding provided for backwards compatibility; new code should use drracket:modes:mode-surrogate instead.

```
drscheme:modes:mode-repl-submit
 : (-> drracket:modes:mode? (-> (is-a?/c drracket:rep:text%) number? boolean?))
```

This binding provided for backwards compatibility; new code should use drracket:modes:mode-repl-submit instead.

```
drscheme:modes:mode-matches-language
 : (-> drracket:modes:mode? (-> (or/c #f (listof string?)) boolean?))
```

This binding provided for backwards compatibility; new code should use drracket:modes:mode-matches-language instead.

```
drscheme:modes:mode-intended-to-edit-programs?
 : (-> drracket:modes:mode? boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:modes:mode-intended-to-edit-programs? instead.

```
drscheme:modes:struct:mode : struct-type?
```

This binding provided for backwards compatibility; new code should use drracket:modes:struct:mode instead.

```
drscheme:modes:get-modes : (-> (listof drracket:modes:mode?))
```

This binding provided for backwards compatibility; new code should use drracket:modes:get-modes instead.

```
drscheme:module-language-tools:add-opt-out-toolbar-button
 : (->* ((-> (is-a?/c top-level-window<%>)
             (is-a?/c area-container<%>)
             (is-a?/c switchable-button%))
         symbol?)
        (#:number (or/c real? #f))
        void?)
```

This binding provided for backwards compatibility; new code should use drracket:module-language-tools:add-opt-out-toolbar-button instead.

```
drscheme:module-language-tools:add-opt-in-toolbar-button
 : (->* ((-> (is-a?/c top-level-window<%>)
             (is-a?/c area-container<%>)
             (is-a?/c switchable-button%))
         symbol?)
        (#:number (or/c real? #f))
        void?)
```

This binding provided for backwards compatibility; new code should use drracket:module-language-tools:add-opt-in-toolbar-button instead.

```
drscheme:module-language-tools:add-online-expansion-handler
 : (-> path-string? symbol? (-> (is-a?/c drracket:unit:definitions-text<%>)
                                any/c
                                any)
       void?)
```

This binding provided for backwards compatibility; new code should use drracket:module-language-tools:add-online-expansion-handler instead.

```
drscheme:module-language-tools:add-online-expansion-monitor
 : (-> path-string? symbol?
       (-> (is-a?/c drracket:unit:definitions-text<%>)
           (or/c drracket:module-language-tools:start?
                 any/c)
           any)
       void?)
```

This binding provided for backwards compatibility; new code should use drracket:module-language-tools:add-online-expansion-monitor instead.

```
drscheme:module-language-tools:start? : (-> any/c boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:module-language-tools:start? instead.

```
drscheme:module-language-tools:register-online-expansion-pref
 : (-> (-> (is-a?/c vertical-panel%) void?) void?)
```

This binding provided for backwards compatibility; new code should use drracket:module-language-tools:register-online-expansion-pref instead.

```
drscheme:module-language-tools:done? : (-> any/c boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:module-language-tools:done? instead.

```
drscheme:module-language-tools:done
 : drracket:module-language-tools:done?
```

This binding provided for backwards compatibility; new code should use drracket:module-language-tools:done instead.

```
drscheme:module-language:add-module-language : (-> any)
```

This binding provided for backwards compatibility; new code should use drracket:module-language:add-module-language instead.

```
drscheme:module-language:module-language-put-file-mixin
 : (-> (implementation?/c text:basic<%>) (implementation?/c text:basic<%>))
```

This binding provided for backwards compatibility; new code should use drracket:module-language:module-language-put-file-mixin instead.

```
drscheme:rep:get-welcome-delta : (-> (is-a?/c style-delta%))
```

This binding provided for backwards compatibility; new code should use drracket:rep:get-welcome-delta instead.

```
drscheme:rep:get-dark-green-delta : (-> (is-a?/c style-delta%))
```

This binding provided for backwards compatibility; new code should use drracket:rep:get-dark-green-delta instead.

```
drscheme:rep:get-error-delta : (-> (is-a?/c style-delta%))
```

This binding provided for backwards compatibility; new code should use drracket:rep:get-error-delta instead.

```
drscheme:rep:get-drs-bindings-keymap : (-> (is-a?/c keymap%))
```

This binding provided for backwards compatibility; new code should use drracket:rep:get-drs-bindings-keymap instead.

```
drscheme:rep:current-rep
 : (-> (or/c false/c (is-a?/c drracket:rep:text%)))
```

This binding provided for backwards compatibility; new code should use drracket:rep:current-rep instead.

```
drscheme:rep:current-value-port : (-> (or/c false/c port?))
```

This binding provided for backwards compatibility; new code should use drracket:rep:current-value-port instead.

```
drscheme:rep:after-expression : (parameter/c (or/c #f (-> any)))
```

This binding provided for backwards compatibility; new code should use drracket:rep:after-expression instead.

```
drscheme:rep:current-language-settings
 : (parameter/c drracket:language-configuration:language-settings?)
```

This binding provided for backwards compatibility; new code should use drracket:rep:current-language-settings instead.

```
drscheme:rep:module-language-initial-run
 : (parameter/c boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:rep:module-language-initial-run instead.

```
drscheme:unit:get-program-editor-mixin
 : (-> ((subclass?/c text%) . -> . (subclass?/c text%)))
```

This binding provided for backwards compatibility; new code should use drracket:unit:get-program-editor-mixin instead.

```
drscheme:unit:add-to-program-editor-mixin
 : (((subclass?/c text%) . -> . (subclass?/c text%)) . -> . void?)
```

This binding provided for backwards compatibility; new code should use drracket:unit:add-to-program-editor-mixin instead.

```
drscheme:unit:open-drscheme-window
 : (->* ()
        ((or/c string? #f) #:show? boolean?)
        (is-a?/c drracket:unit:frame%))
```

This binding provided for backwards compatibility; new code should use drracket:unit:open-drscheme-window instead.

```
drscheme:unit:add-search-help-desk-menu-item
 : (->* ((is-a?/c text%) (is-a?/c menu-item-container<%>) exact-nonnegative-integer?) ((-> an
```

This binding provided for backwards compatibility; new code should use drracket:unit:add-search-help-desk-menu-item instead.

```
drscheme:unit:teachpack-callbacks? : (-> any/c boolean?)
```

This binding provided for backwards compatibility; new code should use drracket:unit:teachpack-callbacks? instead.

```
drscheme:unit:teachpack-callbacks-get-names
 : (-> drracket:unit:teachpack-callbacks? (-> any/c (listof string?)))
```

This binding provided for backwards compatibility; new code should use drracket:unit:teachpack-callbacks-get-names instead.

```
drscheme:unit:teachpack-callbacks-add
 : (-> drracket:unit:teachpack-callbacks? (-> any/c path-string? any/c))
```

This binding provided for backwards compatibility; new code should use drracket:unit:teachpack-callbacks-add instead.

```
drscheme:unit:teachpack-callbacks-remove
 : (-> drracket:unit:teachpack-callbacks? (-> path-string? any/c any/c))
```

This binding provided for backwards compatibility; new code should use drracket:unit:teachpack-callbacks-remove instead.

```
drscheme:unit:teachpack-callbacks-remove-all
 : (-> drracket:unit:teachpack-callbacks? (-> any/c any/c))
```

This binding provided for backwards compatibility; new code should use drracket:unit:teachpack-callbacks-remove-all instead.

```
drscheme:unit:struct:teachpack-callbacks : struct-type?
```

This binding provided for backwards compatibility; new code should use drracket:unit:struct:teachpack-callbacks instead.

```
drscheme:unit:make-teachpack-callbacks : procedure?
```

This binding provided for backwards compatibility; new code should use drracket:unit:make-teachpack-callbacks instead.

```
drscheme:unit:find-symbol
 : (-> (is-a?/c text%) exact-nonnegative-integer? string?)
```

This binding provided for backwards compatibility; new code should use drracket:unit:find-symbol instead.

```
drscheme:tracing:annotate : (-> syntax? syntax?)
```

This binding provided for backwards compatibility; new code should use drracket:tracing:annotate instead.

```
drscheme:init:original-error-display-handler
 : (-> string? any/c any)
```

This binding provided for backwards compatibility; new code should use drracket:init:original-error-display-handler instead.

Index

add-bkg-running-color (method of drracket:unit:tab<%>)
add-show-menu-items (method of drracket:frame:<%>)
add-show-menu-items (method of drracket:unit:frame%)
Adding Arbitrary Languages to DrRacket
Adding Languages to DrRacket
adding languages to DrRacket
Adding Module-based Languages to DrRacket
Adding New Toolbar Buttons
after-create-new-tab (method of drracket:unit:frame<%>)
after-delete (method of drracket:unit:program-editor-mixin)
after-delete (method of drracket:rep:text%)
after-insert (method of drracket:unit:program-editor-mixin)
after-insert (method of drracket:rep:text%)
after-many-evals (method of drracket:rep:text%)
after-set-next-settings (method of drracket:unit:definitions-text<%>)
Backwards Compatibility
begin-metadata-changes (method of drracket:unit:definitions-text<%>)
break button
break-callback (method of drracket:unit:frame%)
break-callback (method of drracket:unit:tab<%>)
breaking
can-close? (method of drracket:unit:tab<%>)
capability-value (method of drracket:language:language<%>)
change-to-file (method of drracket:unit:frame%)
change-to-tab (method of drracket:unit:frame%)
Check Syntax
Check Syntax Button
clear-annotations (method of drracket:rep:context<%>)
clear-annotations (method of drracket:unit:tab%)
close-current-tab (method of drracket:unit:frame<%>)
close-given-tab (method of drracket:unit:frame<%>)
close-ith-tab (method of drracket:unit:frame<%>)
Color Schemes
'color-lexer
'comment
Comments
config-panel (method of drracket:language:module-based-language<%>)
config-panel (method of drracket:language:language<%>)
config-panel (method of drracket:language:simple-module-based-language->module-based-language
'constant
Cooperating with Background Check Syntax
create-executable (method of drracket:language:language<%>)
create-new-tab (method of drracket:unit:frame<%>)
Creating New Kinds of DrRacket Frames
default-settings (method of drracket:language:module-based-language<%>)
default-settings (method of drracket:language:language<%>)
default-settings (method of drracket:language:simple-module-based-language->module-based-language
default-settings? (method of drracket:language:module-based-language<%>)
default-settings? (method of drracket:language:simple-module-based-language->module-based-language
default-settings? (method of drracket:language:language<%>)
Defaulting to Debugging or other Errortrace-

based Annotations
Definition Popup-Menu Navigation
Definitions Text Surrogate
'definitions-text-surrogate
disable-evaluation (method of drracket:unit:tab<%>)
disable-evaluation (method of drracket:rep:context<%>)
'disappeared-binding
'disappeared-use
display-results (method of drracket:rep:text%)
Documentation Language Family
'documentation-language-family
drracket capability, drscheme:tabify-menu-callback
drracket capability, drscheme:special:xml-menus
drracket capability, drscheme:special:slideshow-menu-item
drracket capability, drscheme:special:insert-text-box
drracket capability, drscheme:special:insert-large-letters
drracket capability, drscheme:special:insert-lambda
drracket capability, drscheme:special:insert-image
drracket capability, drscheme:special:insert-gui-tool
drracket capability, drscheme:special:insert-fraction
drracket capability, drscheme:special:insert-comment-box
drracket capability, drscheme:help-context-term
drracket capability, drscheme:define-popup
drracket capability, drscheme:autocomplete-words
drracket capability, drracket:language-menu-title
drracket capability, drracket:check-syntax-button
DrRacket Plugins
'drracket-background-compilation
drracket-tool-icons
drracket-tool-names
drracket-tool-urls
drracket-tools
drracket/syncheck-drracket-button
drracket/tool
drracket/tool-lib
'drracket:comment-delimiters
drracket:debug
drracket:debug:add-prefs-panel
drracket:debug:bug-info->ticket-url
drracket:debug:error-display-handler/stacktrace
drracket:debug:get-error-color
drracket:debug:get-error-color-name
drracket:debug:hide-backtrace-window
drracket:debug:make-debug-compile-handler
drracket:debug:make-debug-error-display-handler
drracket:debug:make-debug-eval-handler
drracket:debug:open-and-highlight-in-file
drracket:debug:profile-definitions-text-mixin
drracket:debug:profile-tab-mixin
drracket:debug:profile-unit-frame-mixin
drracket:debug:profiling-enabled
drracket:debug:show-backtrace-window
drracket:debug:show-backtrace-window/edition-pairs
drracket:debug:show-backtrace-window/edition-pairs/two
drracket:debug:small-planet-bitmap

drracket:debug:test-coverage-definitions-text-mixin
drracket:debug:test-coverage-enabled
drracket:debug:test-coverage-frame-mixin
drracket:debug:test-coverage-interactions-text-mixin
drracket:debug:test-coverage-off-style-name
drracket:debug:test-coverage-on-style-name
drracket:debug:test-coverage-tab-mixin
'drracket:default-extension
'drracket:default-filters
'drracket:default-instrumentation
'drracket:define-popup
drracket:eval
drracket:eval:build-user-eventspace/custodian
drracket:eval:expand-program
drracket:eval:expand-program/multiple
drracket:eval:get-snip-classes
drracket:eval:set-basic-parameters
drracket:eval:traverse-program/multiple
drracket:frame
drracket:frame:<%>
drracket:frame:basics-mixin
drracket:frame:basics<%>
drracket:frame:mixin
drracket:frame:name-message%
drracket:get/extend
drracket:get/extend:allow-re-extension!
drracket:get/extend:disallow-re-extension!
drracket:get/extend:extend-definitions-canvas
drracket:get/extend:extend-definitions-text
drracket:get/extend:extend-interactions-canvas
drracket:get/extend:extend-interactions-text
drracket:get/extend:extend-tab
drracket:get/extend:extend-unit-frame
drracket:get/extend:get-definitions-canvas
drracket:get/extend:get-definitions-text
drracket:get/extend:get-interactions-canvas
drracket:get/extend:get-interactions-text
drracket:get/extend:get-tab
drracket:get/extend:get-unit-frame
'drracket:grouping-position
drracket:help-desk
drracket:help-desk:goto-plt-license
drracket:help-desk:help-desk
'drracket:indentation
drracket:init
drracket:init:original-error-display-handler
'drracket:keystrokes
drracket:language
drracket:language-configuration
drracket:language-configuration:add-language
drracket:language-configuration:fill-language-dialog
drracket:language-configuration:get-languages
drracket:language-configuration:get-settings-preferences-symbol
drracket:language-configuration:language-dialog

drracket:language-configuration:language-settings (struct)
drracket:language-configuration:language-settings-language
drracket:language-configuration:language-settings-settings
drracket:language-configuration:language-settings?
drracket:language-configuration:make-language-settings
drracket:language-configuration:struct:language-settings
drracket:language:add-snip-value
drracket:language:capability-registered?
drracket:language:create-distribution-for-executable
drracket:language:create-executable-gui
drracket:language:create-module-based-distribution
drracket:language:create-module-based-launcher
drracket:language:create-module-based-stand-alone-executable
drracket:language:extend-language-interface
drracket:language:get-capability-contract
drracket:language:get-capability-default
drracket:language:get-default-mixin
drracket:language:get-language-extensions
drracket:language:language<%>
drracket:language:make-setup-printing-parameters
drracket:language:make-simple-settings
drracket:language:make-text/pos
drracket:language:module-based-language->language-mixin
drracket:language:module-based-language<%>
drracket:language:object/c
drracket:language:put-executable
drracket:language:register-capability
drracket:language:setup-printing-parameters
drracket:language:simple-module-based-language%
drracket:language:simple-module-based-language->module-based-language-mixin
drracket:language:simple-module-based-language-convert-value
drracket:language:simple-module-based-language<%>
drracket:language:simple-settings (struct)
drracket:language:simple-settings
drracket:language:simple-settings->vector
drracket:language:simple-settings-annotations
drracket:language:simple-settings-case-sensitive
drracket:language:simple-settings-fraction-style
drracket:language:simple-settings-insert-newlines
drracket:language:simple-settings-printing-style
drracket:language:simple-settings-show-sharing
drracket:language:simple-settings?
drracket:language:struct:simple-settings
drracket:language:struct:text/pos
drracket:language:text/pos (struct)

drracket:language:text/pos-end
drracket:language:text/pos-start
drracket:language:text/pos-text
drracket:language:text/pos?
drracket:modes
drracket:modes:add-mode
drracket:modes:get-modes
drracket:modes:mode (struct)
drracket:modes:mode-intended-to-edit-programs?
drracket:modes:mode-matches-language
drracket:modes:mode-name
drracket:modes:mode-repl-submit
drracket:modes:mode-surrogate
drracket:modes:mode?
drracket:modes:struct:mode
drracket:module-language
drracket:module-language-tools
drracket:module-language-tools:add-online-expansion-handler
drracket:module-language-tools:add-online-expansion-monitor
drracket:module-language-tools:add-opt-in-toolbar-button
drracket:module-language-tools:add-opt-out-toolbar-button
drracket:module-language-tools:definitions-text-mixin
drracket:module-language-tools:definitions-text<%>
drracket:module-language-tools:done
drracket:module-language-tools:done?
drracket:module-language-tools:frame-mixin
drracket:module-language-tools:frame<%>
drracket:module-language-tools:register-online-expansion-pref
drracket:module-language-tools:start?
drracket:module-language-tools:tab-mixin
drracket:module-language-tools:tab<%>
drracket:module-language:add-module-language
drracket:module-language:module-language-put-file-mixin
drracket:module-language:module-language<%>
'drracket:opt-in-toolbar-buttons
'drracket:opt-out-toolbar-buttons
'drracket:paren-matches
'drracket:quote-matches
'drracket:range-indentation
'drracket:range-indentation/reverse-choices
drracket:rep
drracket:rep:after-expression
drracket:rep:context<%>
drracket:rep:current-language-settings
drracket:rep:current-rep
drracket:rep:current-value-port
drracket:rep:drs-bindings-keymap-mixin
drracket:rep:get-dark-green-delta
drracket:rep:get-drs-bindings-keymap
drracket:rep:get-error-delta
drracket:rep:get-welcome-delta
drracket:rep:module-language-initial-run
drracket:rep:text%
drracket:rep:text<%>
'drracket:show-big-defs/ints-labels
'drracket:submit-predicate
drracket:tool-exports^
drracket:tool^

drracket:tool^
'drracket:toolbar-buttons
drracket:tracing
drracket:tracing:annotate
drracket:tracing:frame-mixin
drracket:tracing:tab-mixin
drracket:unit
drracket:unit:add-search-help-desk-menu-item
drracket:unit:add-to-program-editor-mixin
drracket:unit:definitions-canvas%
drracket:unit:definitions-text<%>
drracket:unit:find-symbol
drracket:unit:frame%
drracket:unit:frame<%>
drracket:unit:get-definitions-text%
drracket:unit:get-program-editor-mixin
drracket:unit:interactions-canvas%
drracket:unit:make-teachpack-callbacks
drracket:unit:open-drscheme-window
drracket:unit:program-editor-mixin
drracket:unit:struct:teachpack-callbacks
drracket:unit:tab%
drracket:unit:tab<%>
drracket:unit:teachpack-callbacks (struct)
drracket:unit:teachpack-callbacks-add
drracket:unit:teachpack-callbacks-get-names
drracket:unit:teachpack-callbacks-remove
drracket:unit:teachpack-callbacks-remove-all
drracket:unit:teachpack-callbacks?
drscheme-language-modules
drscheme-language-numbers
drscheme-language-one-line-summaries
drscheme-language-positions
drscheme-language-readers
drscheme-language-urls
drscheme/tool
drscheme/tool-lib
drscheme:debug:add-prefs-panel
drscheme:debug:bug-info->ticket-url
drscheme:debug:error-display-handler/stacktrace
drscheme:debug:get-error-color
drscheme:debug:get-error-color-name
drscheme:debug:hide-backtrace-window
drscheme:debug:make-debug-compile-handler
drscheme:debug:make-debug-error-display-handler
drscheme:debug:make-debug-eval-handler
drscheme:debug:open-and-highlight-in-file
drscheme:debug:profile-definitions-text-mixin
drscheme:debug:profile-tab-mixin
drscheme:debug:profile-unit-frame-mixin
drscheme:debug:profiling-enabled
drscheme:debug:show-backtrace-window
drscheme:debug:show-backtrace-window/edition-pairs
drscheme:debug:show-backtrace-window/edition-pairs/two
drscheme:debug:small-planet-bitmap
drscheme:debug:test-coverage-

definitions-text-mixin
drscheme:debug:test-coverage-enabled
drscheme:debug:test-coverage-frame-mixin
drscheme:debug:test-coverage-interactions-text-mixin
drscheme:debug:test-coverage-off-style-name
drscheme:debug:test-coverage-on-style-name
drscheme:debug:test-coverage-tab-mixin
drscheme:eval:build-user-eventspace/custodian
drscheme:eval:expand-program
drscheme:eval:expand-program/multiple
drscheme:eval:get-snip-classes
drscheme:eval:set-basic-parameters
drscheme:eval:traverse-program/multiple
drscheme:frame:<%>
drscheme:frame:basics-mixin
drscheme:frame:basics<%>
drscheme:frame:mixin
drscheme:get/extend:allow-re-extension!
drscheme:get/extend:disallow-re-extension!
drscheme:get/extend:extend-definitions-canvas
drscheme:get/extend:extend-definitions-text
drscheme:get/extend:extend-interactions-canvas
drscheme:get/extend:extend-interactions-text
drscheme:get/extend:extend-tab
drscheme:get/extend:extend-unit-frame
drscheme:get/extend:get-definitions-canvas
drscheme:get/extend:get-definitions-text
drscheme:get/extend:get-interactions-canvas
drscheme:get/extend:get-interactions-text
drscheme:get/extend:get-tab
drscheme:get/extend:get-unit-frame
drscheme:help-desk:goto-plt-license
drscheme:help-desk:help-desk
drscheme:init:original-error-display-handler
drscheme:language-configuration:add-language
drscheme:language-configuration:fill-language-dialog
drscheme:language-configuration:get-languages
drscheme:language-configuration:get-settings-preferences-symbol
drscheme:language-configuration:language-dialog
drscheme:language-configuration:language-settings-language
drscheme:language-configuration:language-settings-settings
drscheme:language-configuration:language-settings?
drscheme:language-configuration:make-language-settings
drscheme:language-configuration:struct:language-settings
drscheme:language:add-snip-value
drscheme:language:capability-registered?

drscheme:language:create-distribution-for-executable
drscheme:language:create-executable-gui
drscheme:language:create-module-based-distribution
drscheme:language:create-module-based-launcher
drscheme:language:create-module-based-stand-alone-executable
drscheme:language:extend-language-interface
drscheme:language:get-capability-contract
drscheme:language:get-capability-default
drscheme:language:get-default-mixin
drscheme:language:get-language-extensions
drscheme:language:language<%>
drscheme:language:make-setup-printing-parameters
drscheme:language:make-simple-settings
drscheme:language:make-text/pos
drscheme:language:module-based-language->language-mixin
drscheme:language:module-based-language<%>
drscheme:language:put-executable
drscheme:language:register-capability
drscheme:language:setup-printing-parameters
drscheme:language:simple-module-based-language%
drscheme:language:simple-module-based-language->module-based-language-mixin
drscheme:language:simple-module-based-language-convert-value
drscheme:language:simple-module-based-language<%>
drscheme:language:simple-settings->vector
drscheme:language:simple-settings-annotations
drscheme:language:simple-settings-case-sensitive
drscheme:language:simple-settings-fraction-style
drscheme:language:simple-settings-insert-newlines
drscheme:language:simple-settings-printing-style
drscheme:language:simple-settings-show-sharing
drscheme:language:simple-settings?
drscheme:language:struct:simple-settings
drscheme:language:struct:text/pos
drscheme:language:text/pos-end
drscheme:language:text/pos-start
drscheme:language:text/pos-text
drscheme:language:text/pos?
drscheme:modes:add-mode
drscheme:modes:get-modes
drscheme:modes:mode-intended-to-edit-programs?
drscheme:modes:mode-matches-language
drscheme:modes:mode-name
drscheme:modes:mode-repl-submit
drscheme:modes:mode-surrogate
drscheme:modes:mode?
drscheme:modes:struct:mode
drscheme:module-language-tools:add-online-expansion-handler
drscheme:module-language-tools:add-online-expansion-monitor
drscheme:module-language-tools:add-opt-in-toolbar-button
drscheme:module-language-

tools:add-opt-out-toolbar-button
drscheme:module-language-tools:definitions-text-mixin
drscheme:module-language-tools:definitions-text<%>
drscheme:module-language-tools:done
drscheme:module-language-tools:done?
drscheme:module-language-tools:frame-mixin
drscheme:module-language-tools:frame<%>
drscheme:module-language-tools:register-online-expansion-pref
drscheme:module-language-tools:start?
drscheme:module-language-tools:tab-mixin
drscheme:module-language-tools:tab<%>
drscheme:module-language:add-module-language
drscheme:module-language:module-language-put-file-mixin
drscheme:module-language:module-language<%>
'drscheme:opt-out-toolbar-buttons
drscheme:rep:after-expression
drscheme:rep:context<%>
drscheme:rep:current-language-settings
drscheme:rep:current-rep
drscheme:rep:current-value-port
drscheme:rep:drs-bindings-keymap-mixin
drscheme:rep:get-dark-green-delta
drscheme:rep:get-drs-bindings-keymap
drscheme:rep:get-error-delta
drscheme:rep:get-welcome-delta
drscheme:rep:module-language-initial-run
drscheme:rep:text%
drscheme:rep:text<%>
drscheme:tool-exports^
drscheme:tool^
'drscheme:toolbar-buttons
drscheme:tracing:annotate
drscheme:tracing:frame-mixin
drscheme:tracing:tab-mixin
drscheme:unit:add-search-help-desk-menu-item
drscheme:unit:add-to-program-editor-mixin
drscheme:unit:definitions-canvas%
drscheme:unit:definitions-text<%>
drscheme:unit:find-symbol
drscheme:unit:frame%
drscheme:unit:frame<%>
drscheme:unit:get-definitions-text%
drscheme:unit:get-program-editor-mixin
drscheme:unit:interactions-canvas%
drscheme:unit:make-teachpack-callbacks
drscheme:unit:open-drscheme-window
drscheme:unit:struct:teachpack-callbacks
drscheme:unit:tab%
drscheme:unit:tab<%>
drscheme:unit:teachpack-callbacks-add
drscheme:unit:teachpack-callbacks-get-names
drscheme:unit:teachpack-callbacks-remove
drscheme:unit:teachpack-callbacks-remove-all
drscheme:unit:teachpack-callbacks?
edit-menu:between-find-

and-preferences (method of drracket:frame:basics-mixin)
edit-menu:between-select-all-and-find (method of drracket:unit:frame%)
Editor Modes
enable-evaluation (method of drracket:rep:context<%>)
enable-evaluation (method of drracket:unit:tab<%>)
end-metadata-changes (method of drracket:unit:definitions-text<%>)
ensure-defs-shown (method of drracket:unit:frame<%>)
ensure-rep-hidden (method of drracket:unit:frame<%>)
ensure-rep-shown (method of drracket:rep:context<%>)
ensure-rep-shown (method of drracket:unit:frame<%>)
'error
evaluate-from-port (method of drracket:rep:text%)
execute-callback (method of drracket:unit:frame%)
Expanding the User's Program Text and Breaking
expanding user programs
Extending the Existing DrRacket Classes
extra-repl-information (method of drracket:language:language<%>)
file-menu:between-open-and-revert (method of drracket:frame:basics-mixin)
file-menu:between-open-and-revert (method of drracket:unit:frame%)
file-menu:between-print-and-close (method of drracket:unit:frame%)
file-menu:between-print-and-close (method of drracket:frame:basics-mixin)
file-menu:between-save-as-and-print (method of drracket:unit:frame%)
file-menu:new-callback (method of drracket:frame:basics-mixin)
file-menu:new-string (method of drracket:frame:basics-mixin)
file-menu:open-callback (method of drracket:frame:basics-mixin)
file-menu:open-string (method of drracket:frame:basics-mixin)
file-menu:print-string (method of drracket:unit:frame%)
file-menu:save-as-string (method of drracket:unit:frame%)
file-menu:save-string (method of drracket:unit:frame%)
Filename Extensions
find-matching-tab (method of drracket:unit:frame%)
first-opened (method of drracket:language:language<%>)
front-end/complete-program (method of drracket:language:module-based-language->language-mixin)
front-end/complete-program (method of drracket:language:language<%>)
front-end/finished-complete-program (method of drracket:language:language<%>)
front-end/interaction (method of drracket:language:module-based-language->language-mixin)
front-end/interaction (method of drracket:language:language<%>)
General-purpose Modes
get-additional-important-urls (method of drracket:frame:basics-mixin)
get-break-button (method of drracket:unit:frame%)
get-breakables (method of drracket:unit:tab<%>)
get-breakables (method of drracket:rep:context<%>)
get-button-panel (method of drracket:unit:frame%)
get-canvas (method of drracket:unit:frame%)
get-canvas% (method of drracket:unit:frame%)
get-comment-character (method of drracket:language:language<%>)
get-current-tab (method of

drracket:unit:frame<%>)
get-definitions-canvas (method of drracket:unit:frame<%>)
get-definitions-text (method of drracket:unit:frame<%>)
get-definitions/interactions-panel-parent (method of drracket:unit:frame%)
get-defs (method of drracket:unit:tab<%>)
get-directory (method of drracket:unit:tab<%>)
get-directory (method of drracket:rep:context<%>)
get-drscheme-language-positions
get-editor (method of drracket:unit:frame%)
get-editor% (method of drracket:unit:frame%)
get-enabled (method of drracket:unit:tab<%>)
get-error-range (method of drracket:rep:text%)
get-execute-button (method of drracket:unit:frame%)
get-frame (method of drracket:unit:tab<%>)
get-in-module-language? (method of drracket:module-language-tools:definitions-text<%>)
get-init-code (method of drracket:language:simple-module-based-language->module-based-language-mixin)
get-init-code (method of drracket:language:module-based-language<%>)
get-insert-menu (method of drracket:unit:frame<%>)
get-interactions-canvas (method of drracket:unit:frame<%>)
get-interactions-text (method of drracket:unit:frame<%>)
get-ints (method of drracket:unit:tab<%>)
get-keymaps (method of drracket:rep:drs-bindings-keymap-mixin)
get-language-menu (method of drracket:unit:frame<%>)
get-language-name (method of drracket:language:language<%>)
get-language-name (method of drracket:language:module-based-language->language-mixin)
get-language-numbers (method of drracket:language:language<%>)
get-language-numbers (method of drracket:language:module-based-language<%>)
get-language-numbers (method of drracket:language:simple-module-based-language%)
get-language-numbers (method of drracket:language:simple-module-based-language<%>)
get-language-position (method of drracket:language:simple-module-based-language<%>)
get-language-position (method of drracket:language:language<%>)
get-language-position (method of drracket:language:module-based-language<%>)
get-language-position (method of drracket:language:simple-module-based-language%)
get-language-url (method of drracket:language:language<%>)
get-last-touched (method of drracket:unit:tab<%>)
get-metadata (method of drracket:language:language<%>)
get-metadata-lines (method of drracket:language:language<%>)
get-module (method of drracket:language:module-based-language<%>)
get-module (method of drracket:language:simple-module-based-language<%>)
get-module (method of drracket:language:simple-module-based-language%)
get-next-settings (method of drracket:unit:definitions-text<%>)
get-one-line-summary (method of drracket:language:module-based-language<%>)
get-one-line-summary (method of drracket:language:simple-module-based-language%)
get-one-line-summary (method of drracket:language:simple-module-based-language<%>)
get-one-line-summary (method of drracket:language:language<%>)
get-reader (method of drracket:language:simple-module-based-language%)

get-reader (method of drracket:language:simple-module-based-language<%>)
get-reader (method of drracket:language:module-based-language<%>)
get-reader-module (method of drracket:language:language<%>)
get-show-menu (method of drracket:frame:<%>)
get-style-delta (method of drracket:language:language<%>)
get-tab (method of drracket:unit:definitions-text<%>)
get-tab-count (method of drracket:unit:frame<%>)
get-tab-filename (method of drracket:unit:frame<%>)
get-tabs (method of drracket:unit:frame<%>)
get-text-to-search (method of drracket:unit:frame%)
get-transformer-module (method of drracket:language:simple-module-based-language->module-based-language-mixin)
get-transformer-module (method of drracket:language:module-based-language<%>)
get-user-custodian (method of drracket:rep:text%)
get-user-eventspace (method of drracket:rep:text%)
get-user-language-settings (method of drracket:rep:text%)
get-user-namespace (method of drracket:rep:text%)
get-user-thread (method of drracket:rep:text%)
get-users-language-name (method of drracket:module-language:module-language<%>)
help-menu:about-callback (method of drracket:frame:basics-mixin)
help-menu:about-string (method of drracket:frame:basics-mixin)
help-menu:before-about (method of drracket:frame:basics-mixin)
help-menu:create-about? (method of drracket:frame:basics-mixin)
hide-profile-gui (method of drracket:debug:profile-unit-frame-mixin)
highlight-errors (method of drracket:rep:text%)
highlight-errors/exn (method of drracket:rep:text%)
htdp-file-prefix?
htdp-save-file-prefix
'identifiers-as-disappeared-uses?
Implementing DrRacket Plugins
Indentation
initialize-console (method of drracket:rep:text%)
insert-prompt (method of drracket:rep:text%)
is-current-tab? (method of drracket:unit:tab<%>)
is-running? (method of drracket:unit:tab<%>)
Keystrokes
'keyword
kill-evaluation (method of drracket:rep:text%)
lang/htdp-langs-save-file-prefix
Language Extensions
make-drracket:language-configuration:language-settings
make-drracket:language:simple-settings
make-drracket:language:text/pos
make-drracket:unit:teachpack-callbacks
make-searchable (method of drracket:unit:frame%)
marshall-settings (method of drracket:language:module-based-language<%>)
marshall-settings (method of drracket:language:simple-module-based-language->module-based-language
marshall-settings (method of drracket:language:language<%>)
metadata->settings (method of drracket:language:language<%>)
modes
'mouse-over-tooltips

move-current-tab-left (method of drracket:unit:frame<%>)
move-current-tab-right (method of drracket:unit:frame<%>)
move-to-new-language (method of drracket:module-language-tools:definitions-text<%>)
needs-execution (method of drracket:rep:context<%>)
next-tab (method of drracket:unit:frame<%>)
on-close (method of drracket:unit:tab<%>)
on-close (method of drracket:unit:frame%)
on-close (method of drracket:rep:text%)
on-execute (method of drracket:language:module-based-language->language-mixin)
on-execute (method of drracket:language:module-based-language<%>)
on-execute (method of drracket:language:language<%>)
on-execute (method of drracket:rep:text%)
on-execute (method of drracket:language:simple-module-based-language->module-based-language-mixin)
on-highlighted-errors (method of drracket:rep:text%)
on-size (method of drracket:unit:frame%)
on-tab-change (method of drracket:unit:frame<%>)
online-check-syntax logger
open-in-new-tab (method of drracket:unit:frame<%>)
Opting In to Language-Specific Toolbar Buttons
Opting Out of Standard Toolbar Buttons
order-manuals (method of drracket:language:language<%>)
'original-for-check-syntax
'other
'parenthesis
phase1
phase2
Plugin Capabilities
prev-tab (method of drracket:unit:frame<%>)
queue-output (method of drracket:rep:text%)
racket mode
register-capability-menu-item (method of drracket:unit:frame<%>)
register-toolbar-button (method of drracket:unit:frame<%>)
register-toolbar-buttons (method of drracket:unit:frame<%>)
remove-bkg-running-color (method of drracket:unit:tab<%>)
render-value (method of drracket:language:simple-module-based-language->module-based-language
render-value (method of drracket:language:language<%>)
render-value (method of drracket:language:module-based-language<%>)
render-value/format (method of drracket:language:language<%>)
render-value/format (method of drracket:language:simple-module-based-language->module-based-language
render-value/format (method of drracket:language:module-based-language<%>)
reopen-closed-tab (method of drracket:unit:frame<%>)
reorder-tabs (method of drracket:unit:frame<%>)
REPL Submit Predicate
reset-console (method of drracket:rep:text%)
reset-highlighting (method of drracket:rep:text%)
reset-offer-kill (method of drracket:unit:tab<%>)
reset-offer-kill (method of drracket:rep:context<%>)
run-in-evaluation-thread (method of drracket:rep:text%)
scheme mode
set-breakables (method of drracket:unit:tab<%>)
set-breakables (method of drracket:rep:context<%>)
set-filename (method of drracket:unit:definitions-text<%>)
set-message (method of

drracket:frame:name-message%)
set-modified (method of drracket:unit:definitions-text<%>)
set-needs-execution-message (method of drracket:unit:definitions-text<%>)
set-next-settings (method of drracket:unit:definitions-text<%>)
set-show-menu-sort-key (method of drracket:frame:<%>)
Show Big "Definitions" and "Interactions" Labels
show-profile-gui (method of drracket:debug:profile-unit-frame-mixin)
shutdown (method of drracket:rep:text%)
Signatures
sort-toolbar-buttons-panel (method of drracket:unit:frame<%>)
still-untouched? (method of drracket:unit:frame%)
'string
struct:drracket:language-configuration:language-settings
struct:drracket:language:simple-settings
struct:drracket:language:text/pos
struct:drracket:modes:mode
struct:drracket:unit:teachpack-callbacks
'sub-range-binders
'sub-range-binding
'sub-range-binding
'symbol
syncheck-bitmap
syncheck-drracket-button
syncheck:button-callback
Syntax Coloring
Syntax Properties that Check Syntax Looks For
Teaching Languages
Tool support for #lang-based Languages
Toolbar Buttons
touched (method of drracket:unit:tab<%>)
unmarshall-settings (method of drracket:language:simple-module-based-language->module-based-language
unmarshall-settings (method of drracket:language:language<%>)
unmarshall-settings (method of drracket:language:module-based-language<%>)
unregister-toolbar-button (method of drracket:unit:frame<%>)
update-running (method of drracket:rep:context<%>)
update-save-button (method of drracket:unit:frame%)
update-save-message (method of drracket:unit:frame%)
update-shown (method of drracket:unit:frame%)
update-shown (method of drracket:frame:<%>)
use-mred-launcher (method of drracket:language:simple-module-based-language->module-based-language
use-mred-launcher (method of drracket:language:module-based-language<%>)
use-namespace-require/copy? (method of drracket:language:module-based-language<%>)
View menu
wait-for-io-to-complete (method of drracket:rep:text%)
wait-for-io-to-complete/user (method of drracket:rep:text%)