Plugins: Extending DrScheme

Version 4.2.3

Robert Bruce Findler

December 1, 2009

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

This manual describes DrScheme's tools interface. It assumes familiarity with PLT Scheme, as described in ***Guide****: PLT Scheme*, DrScheme, as described in ***DrScheme****: PLT Programming Environment*, and the Framework, as described in ***Framework****: PLT GUI Application Framework*.

Contents

Thanks

Thanks especially to Eli Barzilay, John Clements, Matthias Felleisen, Cormac Flanagan, Matthew Flatt, Max Hailperin, Philippe Meunier, Christian Queinnec, PLT at large, and many others for their feedback and help.

1 Implementing DrScheme Tools

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

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

- **tools**: `(listof (listof string [subcollection-name]))`
- **tool-names**: `(listof (union #f string))`
- **tool-icons**:

  ```
  (listof (union #f
                 string[relative-pathname]
                 (cons string[filename]
                       (listof string[collection-name]))))
  ```
- **tool-urls**: `(listof (union #f string [url]))`

The `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.ss` file resides. As an example, if there is only one tool named `tool.ss`, this suffices:

```
(define tools (list (list "tool.ss")))
```

If the `tool-icons` or `tool-names` fields are present, they must be the same length as `tools`. The `tool-icons` 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, Help Desk's bug report form, and the splash screen as the tool is loaded at DrScheme's startup.

Each of `tools` files must contain a module that `provides` `tool@`, which must be bound to a `unit`. The unit must import the `drscheme:tool^` signature, which is provided by the `tool.ss` library in the `drscheme` collection. The `drscheme:tool^` signature contains all of the names listed in this manual. The unit must export the `drscheme:tool-exports^` signature.

The drscheme:tool-exports^ signature contains two names: phase1 and phase2. These names must be bound to thunks. 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 DrScheme's drscheme: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 drscheme: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.

Phase 1 functions:

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

Phase 2 functions:

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

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

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

```
#lang setup/infotab
(define name "Tool Name")
(define tools (list (list "tool.ss")))
```

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

```
#lang scheme/gui
(require drscheme/tool)

(provide tool@)
```

```
(define tool@
  (unit
    (import drscheme:tool^)
    (export drscheme: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.

2 Adding Languages to DrScheme

2.1 Adding Module-based Languages to DrScheme

If a language can be implemented as a module (see `module` for details) and the standard language settings are sufficient, simply create an `info.ss` file in the collection where the module is saved. 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.ss` 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.
- **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 MzScheme'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.ss` 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.ss` looks like this:

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

This `info.ss` file indicates that there is a single language in this collection. The module that implements the language is the `eopl-lang.ss` file in the same directory as the `info.ss` 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 DrScheme.

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.

2.2 Adding Arbitrary Languages to DrScheme

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

Each language is a class that implement the `drscheme:language:language<%>` interface. DrScheme also provides two simpler interfaces: `drscheme:language:module-based-language<%>` and `drscheme:language:simple-module-based-language<%>`, and mixins `drscheme:language:simple-module-based-language->module-based-language-mixin` and `drscheme:language:module-based-language->language-mixin` that build implementations of `language^`s from these simpler interfaces.

Once you have an implementation of the `drscheme:language:language^` interface, call `drscheme:language-configuration:add-language` to add the language to DrScheme.

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. DrScheme maintains the current settings for each language.

2.3 Language Extensions

Some tools may require additional functionality from the drscheme:language:language interface. The drscheme:language:extend-language-interface function and the drscheme: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 drscheme:language:extend-language-interface to extend the drscheme: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 drscheme: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.

3 Creating New Kinds of DrScheme Frames

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

4 Extending the Existing DrScheme Classes

Each of the names:

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

is bound to an extender function. In order to change the behavior of drscheme, 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 it's argument and return a classes derived from that class as its result. For example:

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

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

5 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, DrScheme's tool interface provides `drscheme: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 DrScheme 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, DrScheme provides the `set-breakables` method. This method controls what behavior the Break button has.

6 Editor Modes

DrScheme provides support for multiple editor modes. Tools register modes via `drscheme:modes:add-mode`. Each mode is visible in the Modes submenu of the Edit menu. Initially, DrScheme only supports two modes: scheme mode and text mode.

DrScheme automatically selects a mode for each open file based on the file's extension. If the file ends with `.txt`, DrScheme uses text mode. Otherwise, DrScheme uses Scheme mode.

7 #lang-specific tools

If the result of read-language for a language is a function, DrScheme will query it to determine if there are any new toolbar buttons to be used when editing files in this language (when DrScheme's language is set to the Module language).

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

```
(listof (list/c string?
                (is-a?/c bitmap%)
                (-> (is-a?/c drscheme:unit:frame<%>) any)))
```

which is then used to create new toolbar buttons, one for each list in the first. The string is the label on the button; the bitmap is the icon (it should be 16x16), and the function is called when the button is clicked.

8 Language-specific capabilities

Drscheme'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. `drscheme: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 DrScheme.

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

9 drscheme:get/extend

```
(drscheme:get/extend:extend-tab mixin) → void?
  mixin : (make-mixin-contract drscheme:unit:tab<%>)
(drscheme:get/extend:extend-tab mixin
                                before?) → void?
  mixin : (make-mixin-contract drscheme:unit:tab<%>)
  before? : boolean?
```

This class implements the tabs in drscheme. 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)

The argument, `before`, controls if the mixin is applied before or after already installed mixins. If unsupplied, this is the same as supplying `#t`.

```
(drscheme:get/extend:extend-interactions-text mixin) → void?
  mixin : (make-mixin-contract drscheme:rep:text<%>)
(drscheme:get/extend:extend-interactions-text mixin
                                              before?) → void?
  mixin : (make-mixin-contract drscheme:rep:text<%>)
  before? : boolean?
```

This text is used in the bottom window of drscheme frames.

The argument, `before`, controls if the mixin is applied before or after already installed mixins. If unsupplied, this is the same as supplying `#t`.

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

Once this function is called, `drscheme:get/extend:extend-interactions-text` raises an error, disallowing any more extensions.

```
(drscheme:get/extend:extend-definitions-text mixin) → void?
  mixin : (make-mixin-contract drscheme:unit:definitions-text<%>)
(drscheme:get/extend:extend-definitions-text mixin
                                             before?) → void?
  mixin : (make-mixin-contract drscheme:unit:definitions-text<%>)
  before? : boolean?
```

This text is used in the top window of drscheme frames.

The argument, `before`, controls if the mixin is applied before or after already installed mixins. If unsupplied, this is the same as supplying #f.

```
(drscheme:get/extend:get-definitions-text)
 → (implementation?/c drscheme:unit:definitions-text<%>)
```

Once this function is called, drscheme:get/extend:extend-definitions-text raises an error, disallowing any more extensions.

```
(drscheme:get/extend:extend-interactions-canvas mixin) → void?
  mixin : (make-mixin-contract drscheme:unit:interactions-canvas%)
(drscheme:get/extend:extend-interactions-canvas mixin
                                                before?)
 → void?
  mixin : (make-mixin-contract drscheme:unit:interactions-canvas%)
  before? : boolean?
```

This canvas is used in the bottom window of drscheme frames.

The argument, before, controls if the mixin is applied before or after already installed mixins. If unsupplied, this is the same as supplying #f.

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

Once this function is called, drscheme:get/extend:extend-interactions-canvas raises an error, disallowing any more extensions.

```
(drscheme:get/extend:extend-definitions-canvas mixin) → void?
  mixin : (make-mixin-contract drscheme:unit:definitions-canvas%)
(drscheme:get/extend:extend-definitions-canvas mixin
                                               before?) → void?
  mixin : (make-mixin-contract drscheme:unit:definitions-canvas%)
  before? : boolean?
```

This canvas is used in the top window of drscheme frames.

The argument, before, controls if the mixin is applied before or after already installed mixins. If unsupplied, this is the same as supplying #f.

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

Once this function is called, drscheme:get/extend:extend-definitions-canvas raises an error, disallowing any more extensions.

```
(drscheme:get/extend:extend-unit-frame mixin) → void?
  mixin : (make-mixin-contract drscheme:unit:frame%)
(drscheme:get/extend:extend-unit-frame mixin
                                       before?) → void?
  mixin : (make-mixin-contract drscheme:unit:frame%)
  before? : boolean?
```

This is the frame that implements the main drscheme window.

The argument, `before`, controls if the mixin is applied before or after already installed mixins. If unsupplied, this is the same as supplying `#f`.

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

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

10 drscheme:unit

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

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

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

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

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

Overrides `enable-evaluation` in `drscheme: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-drscheme:unit:tab get-breakables)
 → (or/c thread? false/c)
   (or/c custodian? false/c)
```

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

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

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

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

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

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

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

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

Returns the frame that this tab is inside.

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

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

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

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

Indicates if this tab is the currently active tab.

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

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

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

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

```
(send a-drscheme:unit:tab set-breakables thread
                                          custodian) → void?
```

```
thread : (or/c thread? false/c)
custodian : (or/c custodian? false/c)
```

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

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

The base class that implements the tab's functionality.

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

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

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

Clears any error highlighting.

```
drscheme: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-drscheme:unit:program-editor after-delete start
                                                  len)
 → void?
  start : number
  len : number
```

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

Calls the super method.

Resets an error highlighting.

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

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

Calls the super method.

Resets an error highlighting.

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

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

Passes all arguments to `super-init`.

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

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

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

Passes all arguments to `super-init`.

```
(new drscheme:unit:frame%) → (is-a?/c drscheme:unit:frame%)
```

Passes all arguments to `super-init`.

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

Overrides <method not found>.

Adds the “Show Definitions”, “Show Interactions” and “Show Contour” menu items.

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

Loads this file into this already created frame. In normal DrScheme 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-drscheme:unit:frame edit-menu:between-select-all-and-find)
 → void?
```

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

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

```
(send a-drscheme: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 `do-many-text-evals` 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-drscheme:unit:frame file-menu:between-open-and-revert)
 → void?
```

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

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

```
(send a-drscheme:unit:frame file-menu:between-print-and-close)
 → void?
```

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

Adds a menu item for printing the interactions.

```
(send a-drscheme:unit:frame file-menu:between-save-as-and-print)
 → void?
```

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-drscheme:unit:frame file-menu:print-string) → void?
```

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

returns `"Definitions"`

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

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

Returns `"Definitions"`.

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

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

Returns `"Definitions"`.

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

Returns the break button. Mostly used for test suites.

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

This panel goes along the top of the drscheme 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-drscheme:unit:frame get-canvas)
 → (is-a?/c editor-canvas%)
```

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

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

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

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

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

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

(send *a-drscheme:unit:frame* get-definitions/interactions-panel-parent)
→ void?

Specification: This method is provided so that tools can add area-container<%>s to the drscheme 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-drscheme:unit:frame* get-editor) → (is-a?/c editor<%>)

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

Returns the result of get-definitions-text.

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

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

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

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

Returns the Run button. Mostly used for test suites.

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

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

(send *a-drscheme:unit:frame* on-close) → void?

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

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

Calls the super method.

(send *a-drscheme:unit:frame* on-size) → void?

Overrides on-size in window<%>.

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

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

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

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

Overrides <method not found>.

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

```
drscheme:unit:frame<%> : interface?
```

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

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

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

Ensures that the definitions window is visible.

(send *a-drscheme:unit:frame* ensure-rep-hidden) → void?

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

(send *a-drscheme:unit:frame* ensure-rep-shown) → void?

Shows the interactions window

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

Returns the currently active tab.

(send *a-drscheme:unit:frame* close-current-tab) → void?

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

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

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

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

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

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

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

Specification: Returns the Insert menu.

(send *a-drscheme:unit:frame* get-interactions-canvas)
→ (instanceof (derivedfrom drscheme:unit:interactions-canvas%))

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

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

(send *a-drscheme:unit:frame* get-interactions-text)
→ (instanceof (derivedfrom drscheme:rep:text%))

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

```
(send a-drscheme:unit:frame get-tabs)
 → (listof drscheme:unit:tab<%>)
```

Returns the list of tabs in this frame.

```
(send a-drscheme:unit:frame on-tab-change from-tab
                                           to-tab)  → void?
  from-tab : (is-a?/c drscheme:unit:tab<%>)
  to-tab : (is-a?/c drscheme: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-drscheme: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 cliked 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 `drscheme:language:register-capability`.

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

Registers the toolbar button `tb`. This is required so that the toolbar buttons properly switch orientation when the toolbar's position is moved.

```
(send a-drscheme:unit:frame register-toolbar-buttons tbs)
 → void?
  tbs : (listof (is-a?/c switchable-button%))
```

Simultaneously registers the toolbar buttons `tbs`. This is required so that the toolbar buttons properly switch orientation when the toolbar's position is moved.

```
(send a-drscheme: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.

```
drscheme:unit:definitions-text% : class?
  superclass: (drscheme:rep:drs-bindings-keymap-mixin (drscheme:unit:program-editor-mixin (schem
  extends: drscheme:unit:definitions-text<%>
```

```
(new drscheme:unit:definitions-text%)
 → (is-a?/c drscheme:unit:definitions-text%)
```

Passes all arguments to super-init.

```
(send a-drscheme:unit:definitions-text set-filename) → void?
```

Overrides <method not found>.

Calls update-save-message.

```
(send a-drscheme:unit:definitions-text set-modified) → void?
```

Overrides <method not found>.

Calls update-save-button.

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

This interface is implemented by the definitions text.

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

Augment this method to be notified when DrScheme is changing the buffer to insert metadata. The metadata is only inserted during saving, so tools that track changes to DrScheme 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-drscheme: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-drscheme: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 DrScheme window.

```
(send a-drscheme:unit:definitions-text get-port-name-identifier)
 → symbol
```

Returns an identifier that can be used as a port's name when the editor is not saved. (If it is saved, the filename of the editor should be used.)

```
(send a-drscheme:unit:definitions-text get-tab)
 → (instanceof drscheme:unit:tab%)
```

Returns the editor's enclosing tab.

```
(send a-drscheme:unit:definitions-text port-name-matches? id)
 → boolean?
  id : any
```

Indicates if the name of a port (which is also saved in the source field of an exception record) matches this editor.

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

Specification: This method, when called, puts this drscheme 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-drscheme: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`.

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

Initializes the visibility of the save button.

```
(drscheme: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 `drscheme:unit:add-to-program-editor-mixin`.

```
(drscheme: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 §1 “Implementing DrScheme Tools” for details).

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

```
(drscheme:unit:open-drscheme-window)
 → (is-a?/c drscheme:unit:frame%)
(drscheme:unit:open-drscheme-window filename)
 → (is-a?/c drscheme:unit:frame%)
  filename : (or/c string? false/c)
```

Opens a drscheme frame that displays *filename*, or nothing if *filename* is `#f` or not supplied.

11 drscheme:language

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

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

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

```
(send a-drscheme:language:simple-module-based-language get-language-numbers)
 → (cons 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-drscheme:language:simple-module-based-language get-language-position)
 → (cons string (listof string))
```

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

```
(send a-drscheme: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-drscheme:language:simple-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-drscheme: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.

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

```
 (make-object drscheme:language:simple-module-based-language%
  module
  language-position
 [language-numbers
  one-line-summary
  documentation-reference]
  reader
  language-id)
 → (is-a?/c drscheme:language:simple-module-based-language%)
  module : s-expression
  language-position : (cons string (listof string))
  language-numbers :  (cons number (listof number))
                   = (map (lambda (x) 0) language-position)
  one-line-summary : string? = ""
  documentation-reference : (or/c false/c 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-drscheme:language:simple-module-based-language get-language-numbers)
 → (cons number (listof number))
```

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

returns the corresponding init arg.

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

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

returns the corresponding init arg.

```
(send a-drscheme:language:simple-module-based-language get-module)
 → (cons string (listof string))
```

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

returns the corresponding init arg.

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

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

returns the corresponding initialization argument.

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

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

returns the corresponding init arg.

`drscheme:language:simple-module-based-language->module-based-language-mixin : (class? . -> . c`

argument extends/implements: `drscheme:language:simple-module-based-language<%>`

result implements: `drscheme:language:module-based-language<%>`

This mixin uses a struct definition for its settings:

```
 (define-struct drscheme:language:simple-settings
   (case-sensitive  ; boolean?
    printing-style  ; (symbols 'constructor 'quasiquote 'write
'print)
    fraction-style  ; (symbols 'mixed-fraction 'mixed-fraction-e
                    ;  'repeating-decimal 'repeating-decimal-e)
    show-sharing    ; boolean?
    insert-newlines ; boolean?
    annotations))   ; (symbols '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 `'write` is the default style, used in the MzScheme 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-drscheme:language:simple-module-based-language->module-based-language config-panel)
 → (case-> (-> settings) (settings -> void))
```

Overrides <method not found>.

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

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

```
(send a-drscheme:language:simple-module-based-language->module-based-language default-setting
 → settings
```

Overrides <method not found>.

The defaults for the settings are

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

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

```
(send a-drscheme:language:simple-module-based-language->module-based-language default-setting
 → boolean?
```

Overrides <method not found>.

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

Overrides <method not found>.

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-drscheme:language:simple-module-based-language->module-based-language get-transformer-
 → s-expression
```

Overrides <method not found>.

Returns 'mzscheme.

```
(send a-drscheme:language:simple-module-based-language->module-based-language marshall-setting
 → writable
```

Overrides <method not found>.

Constructs a vector from the structure.

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

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

Overrides <method not found>.

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

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

Overrides <method not found>.

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

```
(send a-drscheme:language:simple-module-based-language->module-based-language render-value/fo
 → void?
```

Overrides <method not found>.

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

```
(send a-drscheme:language:simple-module-based-language->module-based-language unmarshall-sett
 → (or/c false/c settings)
```

Overrides <method not found>.

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

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

```
(send a-drscheme:language:simple-module-based-language->module-based-language use-mred-launche
 → boolean?
```

Overrides <method not found>.

Returns #t.

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

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

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

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

This method is the same as default-settings.

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

This method is the same as default-settings?.

```
(send a-drscheme: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-drscheme:language:module-based-language get-language-numbers)
 → (cons number (listof number))
```

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

```
(send a-drscheme:language:module-based-language get-language-position)
 → (cons string (listof string))
```

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

```
(send a-drscheme: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-drscheme: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-drscheme: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-drscheme: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, ie 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-drscheme:language:module-based-language marshall-settings settings)
```

```
→ writable
 settings : settings
```

This method is the same as marshall-settings.

```
(send a-drscheme:language:module-based-language on-execute
 settings
 run-in-user-thread)
→ vod
 settings : settings
 run-in-user-thread : ((-> void) -> void)
```

This method is the same as on-execute.

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

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

```
(send a-drscheme:language:module-based-language unmarshall-settings input)
→ (or/c settings false/c)
 input : writable
```

This method is the same as unmarshall-settings.

```
(send a-drscheme: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 mred or the mzscheme binary.

```
(send a-drscheme: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 mzscheme primitive `namespace-require/copy` is used and if it returns `#f`, `namespace-require` is used.

Default implementation: Defaultly returns `#f`.

```
drscheme:language:module-based-language->language-mixin : (class? . -> . class?)
```

argument extends/implements: `drscheme:language:module-based-language<%>`
result implements: `drscheme:language:language<%>`

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

Overrides <method not found>.

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

Overrides <method not found>.

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-drscheme:language:module-based-language->language get-language-name)
 → string?
```

Overrides <method not found>.

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

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

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

Calls the super method.

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

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

Implementations of this interface are languages that DrScheme supports.

See §2 "Adding Languages to DrScheme" for an overview of adding languages to DrScheme.

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

Specification: Returns the language-specific value for some capability. See also `drscheme:language:register-capability`.

Default implementation: Defaultly returns the value from: `drscheme:language:get-capability-default`.

```
(send a-drscheme: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-drscheme: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 `drscheme:language:create-module-based-stand-alone-executable` and `drscheme:language:create-module-based-launcher`.

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

Specifies the default settings for this language.

```
(send a-drscheme:language:language default-settings? settings)
```

```
 → boolean?
  settings : settings
```

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

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

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

See also initialize-console.

```
 (send a-drscheme: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, parses, and optionally compiles a program in the language. The first argument contains all of the data to be read (until eof) and the second argument 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 third 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-drscheme: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 DrScheme window will be wedged.

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

Returns text to be used for the "Insert Large Letters" menu item in DrScheme. 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-drscheme: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 drscheme window.

```
(send a-drscheme:language:language get-language-numbers)
 → (cons 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 DrScheme 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-drscheme:language:language get-language-position)
```

```
 → (cons 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 Language")
```

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-drscheme:language:language get-language-url)
 → (or/c string? false/c)
```

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-drscheme: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 DrScheme, and removed from the buffer after it is opened in DrScheme.

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-drscheme: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-drscheme: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-drscheme:language:language get-reader-module)
 → (or/c sexp-representing-a-require-spec false/c)
```

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, DrScheme 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-drscheme: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 DrScheme 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-drscheme:language:language marshall-settings settings)
 → writable
  settings : settings
```

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

```
(send a-drscheme: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 DrScheme, 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-drscheme:language:language on-execute
 settings
 run-in-user-thread)
→ any
 settings : settings
 run-in-user-thread : ((-> any) -> any)
```

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

Use this method to initialize MzScheme's §10.3.2 "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 MzScheme:

- 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 DrScheme's original namespace:
  - 'mzscheme
  - '(lib "mred.ss" "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 DrScheme 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 DrScheme started up.
- The snip-class-list, returned by `get-the-snip-class-list` is initialized with all of the snipclasses in DrScheme'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-in-user-thread`* arguments accepts thunks and runs them on the user's eventspace's main thread. These thunks must not raise an exceptions (or drscheme itself will get stuck). In addition, the output ports are not yet functioning, so print outs should be directed to the original drscheme output port, if necessary.

```
(send a-drscheme: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-drscheme: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-drscheme:language:language render-value/format
  value
  settings
  port
  width)
 → void?
  value : TST
  settings : settings
  port : port
  width : (or/c number (symbols '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-drscheme:language:language unmarshall-settings input)
 → (or/c settings false/c)
  input : writable
```

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

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

```
(drscheme: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 DrScheme starts up:

- 'drscheme:check-syntax-button : boolean? = #t— controls the visiblity of the check syntax button
- 'drscheme:language-menu-title : string? = (string-constant scheme-menu-name)— controls the name of the menu just to the right of the language menu (defaultly named “Scheme”)
- 'drscheme:define-popup : (or/c (cons/c string? string?) false/c) = (cons "(define" "(define ...)")— 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
- 'drscheme:help-context-term : (or/c false/c string?) = #f— speci-

fies 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 false/c (-> (is-a?/c text%) number? number? void?)) = (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.

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

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

```
(drscheme:language:get-capability-default s)
```

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

Returns the default for a particular capability.

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

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

```
(drscheme: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.

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

This function can only be called in phase 1 (see §1 "Implementing DrScheme Tools" for details).

Each language added passed to `drscheme: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 `drscheme:language:get-default-mixin`.

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

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

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

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

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

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

```
(drscheme: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? (symbols 'launcher 'standalone 'distribution))
  mred? : boolean?
  title : string?
```

Calls the MrEd 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.

```
(drscheme:language:create-executable-gui parent
                                         program-name
                                         show-type
                                         show-base)
 → (or/c false/c
         (list/c (symbols 'no-show 'launcher 'stand-alone 'distribution)
                 (symbols 'no-show 'mred 'mzscheme)
                 string?))
  parent : (or/c false/c (is-a?/c top-level-window<%>))
  program-name : (or/c false/c string?)
  show-type : (or/c (λ (x) (eq? x #t)) (symbols 'launcher 'standalone 'distribution))
  show-base : (or/c (λ (x) (eq? x #t)) (symbols '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.

```
 (drscheme: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 identifer *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 MrEd or MzScheme stand-alone executable is created.

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

```
(drscheme: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 drscheme:language:create-module-based-stand-alone-executable, but packages the stand-alone executable into a distribution.

```
(drscheme: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 `drscheme: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.

```
(drscheme: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 `drscheme:language:create-module-based-stand-alone-executable`, except that it creates a launcher instead of a stand-alone executable.

```
(drscheme:language:text/pos-text text/pos) → (is-a?/c text%)
  text/pos : drscheme:language:text/pos?
```

Selects the `text%` from a text/pos.

```
(drscheme:language:text/pos-start text/pos) → number?
  text/pos : drscheme:language:text/pos?
```

Selects the starting position from a text/pos.

```
(drscheme:language:text/pos-end text/pos) → number?
  text/pos : drscheme:language:text/pos?
```

Selects the ending position from a text/pos.

```
(drscheme:language:text/pos? val) → boolean?
  val : any/c
```

Returns #t if val is a text/pos, and #f otherwise.

```
(drscheme:language:make-text/pos text
                                 start
                                 end)
 → drscheme:language:text/pos?
  text : (is-a?/c text%)
  start : number?
  end : number?
```

Constructs a text/pos.

```
(drscheme:language:simple-settings-case-sensitive simple-settings)
 → boolean?
  simple-settings : drscheme:language:simple-settings?
```

Extracts the case-sensitive setting from a simple-settings.

```
(drscheme:language:simple-settings-printing-style simple-settings)
 → (symbols 'constructor 'quasiquote 'write)
  simple-settings : drscheme:language:simple-settings?
```

Extracts the printing-style setting from a simple-settings.

```
(drscheme:language:simple-settings-fraction-style simple-settings)
 → (symbols 'mixed-fraction
            'mixed-fraction-e
            'repeating-decimal
            'repeating-decimal-e)
  simple-settings : drscheme:language:simple-settings?
```

Extracts the fraction-style setting from a simple-settings.

```
(drscheme:language:simple-settings-show-sharing simple-settings)
 → boolean?
  simple-settings : drscheme:language:simple-settings?
```

Extracts the show-sharing setting from a simple-settings.

```
(drscheme:language:simple-settings-insert-newlines simple-settings)
 → boolean?
  simple-settings : drscheme:language:simple-settings?
```

Extracts the insert-newline setting from a simple-settings.

```
(drscheme:language:simple-settings-annotations simple-settings)
 → (symbols 'none 'debug 'debug/profile 'test-coverage)
  simple-settings : drscheme:language:simple-settings?
```

Extracts the debugging setting from a simple-settings.

```
(drscheme:language:simple-settings? val) → boolean?
  val : any/c
```

Determines if *val* is a simple-settings.

```
(drscheme:language:make-simple-settings case-sensitive
                                        printing-style
                                        fraction-style
                                        show-sharing
                                        insert-newlines
                                        annotations)
 → drscheme:language:simple-settings?
  case-sensitive : boolean?
  printing-style : (symbols 'constructor 'quasiquote 'write)
  fraction-style : (symbols 'mixed-fraction 'mixed-fraction-e 'repeating-decimal 'repeating-de
  show-sharing : boolean?
  insert-newlines : boolean?
  annotations : (symbols 'none 'debug 'debug/profile 'test-coverage)
```

Constructs a simple settings.

```
(drscheme:language:simple-settings->vector simple-settings)
 → vector?
  simple-settings : drscheme:language:simple-settings?
```

Constructs a vector whose elements are the fields of *simple-settings*.

12 drscheme:language-configuration

```
(drscheme:language-configuration:get-languages)
 → (listof (is-a?/c drscheme: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 DrScheme.

```
(drscheme:language-configuration:add-language language) → void?
  language : (and/c (is-a?/c drscheme:language:language<%>) drscheme:language:object/c)
```

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

Adds *language* to the languages offerend by DrScheme.

```
(drscheme: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`.

```
 (drscheme:language-configuration:make-language-settings
  language
  settings)
 → drscheme:language-configuration:language-settings?
  language : (or/c (is-a?/c drscheme:language:language<%>) drscheme:language:object/c)
  settings : any/c
```

This is the constructor for a record consisting of two elements, a language and its settings.

The settings is a language-specific record that holds a value describing a parameterization of the language.

It has two selectors, `drscheme:language-configuration:language-settings-language` and `drscheme:language-configuration:language-settings-settings`, and a predicate, `drscheme:language-configuration:language-settings?`

```
(drscheme:language-configuration:language-settings-settings ls)
 → any/c
```

```
  ls : drscheme:language-configuration:language-settings?
```

Extracts the settings field of a language-settings.

```
(drscheme:language-configuration:language-settings-language ls)
 → (or/c (is-a?/c drscheme:language:language<%>) drscheme:language:object/c)
  ls : drscheme:language-configuration:language-settings?
```

Extracts the language field of a language-settings.

```
(drscheme:language-configuration:language-settings? val)
 → boolean?
  val : any/c
```

Determines if the argument is a language-settings or not.

```
 (drscheme:language-configuration:language-dialog
   show-welcome?
   language-settings-to-show
  [parent])
 → (or/c false/c drscheme:language-configuration:language-settings?)
  show-welcome? : boolean?
  language-settings-to-show : drscheme:language-configuration:language-settings?
  parent : (or/c false/c (is-a?/c top-level-window<%>)) = #t
```

Opens the language configuration dialog. See also drscheme:language-configuration:fill-language-dialog.

The *show-welcome?* argument determines if if a “Welcome to DrScheme” 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 (drscheme: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.

```
(drscheme:language-configuration:fill-language-dialog
  panel
  button-panel
  language-setting
 [re-center
  ok-handler])
→ drscheme:language-configuration:language-settings?
 panel : (is-a?/c vertical-panel%)
 button-panel : (is-a?/c area-container<%>)
 language-setting : drscheme: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.)

13 drscheme:debug

```
drscheme:debug:profile-unit-frame-mixin : (class? . -> . class?)
  argument extends/implements:  drscheme:frame:<%>
                                drscheme:unit:frame<%>
```

```
drscheme:debug:profile-interactions-text-mixin : (class? . -> . class?)
  argument extends/implements:  drscheme:rep:text<%>
```

```
drscheme:debug:profile-definitions-text-mixin : (class? . -> . class?)
  argument extends/implements:  drscheme:unit:definitions-text<%>
                                text%
```

```
 (drscheme: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 drscheme:unit:definitions-text<%>))
      = #f
 ints : (or/c #f (is-a?/c drscheme:rep:text<%>)) = #f
```

Displays the error message represented by the string, adding embellishments like those that appears in the DrScheme 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.

```
(drscheme: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 MzScheme's `error-display-handler` parameter.

If the current-error-port is the definitions window in drscheme, this error handler inserts some debugging annotations, calls *oedh*, and then highlights the source location of the runtime 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`.

```
(drscheme:debug:hide-backtrace-window) → void?
```

Hides the backtrace window.

```
(drscheme:debug:add-prefs-panel) → void?
```

Adds the profiling preferences panel.

```
(drscheme: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?/c v editor<%>)))))
                                  number?))
               = #f
```

This function opens a DrScheme 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.

```
(drscheme: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?/c v editor<%>)))))
                                             number?)))
  defs : (or/c #f (is-a?/c drscheme:unit:definitions-text<%>))
  ints : (or/c #f (is-a?/c drscheme:rep:text<%>))
```

Shows the backtrace window you get when clicking on the bug in DrScheme's REPL.

The *error-message* argument is the text of the error, *dis* is the debug information, extracted from the continuation mark in the exception record, using `errortrace-key`.

The `editions` argument indicates the editions of any editors that are open editing the files corresponding to the source locations

The *defs* argument should be non-`#f` if there are possibly stacktrace frames that contain unsaved versions of the definitions window from drscheme. Similarly, the *ints* argument should be non-`#f` if there are possibly stacktrace frames that contain unsaved versions of the interactions window.

Use `drscheme:rep:current-rep` to get the rep during evaluation of a program.

```
(drscheme:debug:show-backtrace-window  error-message
                                       dis
                                      [rep
                                       defs])        → void?
  error-message : string?
  dis : (or/c exn?
              (listof srcloc?)
              (non-empty-listof (cons/c string? (listof srcloc?))))
  rep : (or/c #f (is-a?/c drscheme:rep:text<%>)) = #f
  defs :  (or/c #f (is-a?/c drscheme:unit:definitions-text<%>))
       = #f
```

Shows the backtrace window you get when clicking on the bug in DrScheme's REPL.

This function simply calls drscheme:debug:show-backtrace-window/edition-pairs, using drscheme:debug:srcloc->edition/pair.

```
(drscheme:debug:srcloc->edition/pair srcloc
                                     ints
                                     defs)
 → (or/c #f (cons/c (let ([weak-box-containing-an-editor?
                           (λ (x) (and (weak-box? x)
                                       (let ([v (weak-box-value x)])
                                         (or (not v)
                                             (is-a?/c v editor<%>)))))])
                      weak-box-containing-an-editor?)
                    number?))
  srcloc : srcloc?
  ints : (or/c #f (is-a?/c drscheme:rep:text<%>))
  defs : (or/c #f (is-a?/c drscheme:unit:definitions-text<%>))
```

Constructs a edition pair from a source location, returning the current edition of the editor editing the source location (if any).

The *ints* and *defs* arguments are used to map source locations, in the case that the source location corresponds to the definitions window (when it has not been saved) or the interactions window.

14 drscheme:rep

```
drscheme:rep:text<%> : interface?
```

```
drscheme:rep:text% : class?
  superclass: scheme:text%
  extends: drscheme:rep:text<%>
```

This class implements a read-eval-print loop for DrScheme. User submitted evaluations in DrScheme 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 drscheme:rep:text% context)
 → (is-a?/c drscheme:rep:text%)
  context : (implements drscheme:rep:context<%>)
```

```
(send a-drscheme:rep:text after-delete) → void?
```

Overrides `after-delete` in `mode:host-text-mixin`.

Resets any error highlighting in this editor.

```
(send a-drscheme:rep:text after-insert) → void?
```

Overrides `after-insert` in `mode:host-text-mixin`.

Resets any error highlighting in this editor.

```
(send a-drscheme:rep:text display-results results) → void?
  results : (list-of TST)
```

This displays each of the elements of *results* in the interactions window, expect those elements of *results* that are void. Those are just ignored.

```
(send a-drscheme:rep:text do-many-evals run-loop) → void?
  run-loop : (((-> void) -> void) -> void)
```

Specification: Use this function to evaluate code or run actions that should mimic the user's interactions. For example, DrScheme uses this function to evaluate expressions in the definitions window and expressions submitted at the prompt.

Default implementation: The function *run-loop* is called. It is expected to loop, calling it's argument with a thunk that corresponds to the user's evaluation.

It should call it's argument once for each expression the user is evaluating. It should pass a thunk to it's argument that actually does the users's evaluation.

```
(send a-drscheme:rep:text do-many-text-evals
 text
 start
 end
 complete-program?)
→ void?
 text : (is-a?/c text%)
 start : int
 end : int
 complete-program? : any/c
```

Specification: This function evaluates all of the expressions in a text.

Default implementation: It evaluates all of the expressions in `text` starting at `start` and ending at `end`, calling `do-many-evals` to handle the evaluation.

The `complete-program?` argument determines if the `front-end/complete-program` method or the `front-end/interaction` method is called.

```
(send a-drscheme: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.

This method must be called from the drscheme main thread.

```
(send a-drscheme:rep:text after-many-evals) → any
```

Augments <method not found>.

Called from the drscheme main thread after `evaluate-from-port` finishes (no matter how it finishes).

```
(send a-drscheme:rep:text on-execute run-on-user-thread) → any
 run-on-user-thread : (-> any)
```

Augments <method not found>.

Called from the drscheme thread after the language's on-execute method has been invoked, and after the special values have been setup (the ones registered via drscheme:language:add-snip-value).

Use *run-on-user-thread* to initialize the user's parameters, etc.

```
(send a-drscheme: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-drscheme:rep:text get-user-custodian)
 → (or/c false/c custodian?)
```

This is the custodian controlling the user's program.

```
(send a-drscheme: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-drscheme: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-drscheme: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-drscheme:rep:text get-user-thread)
 → (or/c false/c thread?)
```

This method returns the thread that the users code runs in. It is 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 §10.3.2 "Parameters" for more information about parameters.

```
(send a-drscheme: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-drscheme: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-drscheme:rep:text initialize-console) → void?
```

This inserts the "Welcome to DrScheme" 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.

```
(send a-drscheme:rep:text insert-prompt) → void?
```

Inserts a new prompt at the end of the text.

```
(send a-drscheme:rep:text kill-evaluation) → void?
```

This method is called when the user chooses the kill menu item.

```
(send a-drscheme:rep:text on-close) → void?
```

Overrides on-close in editor:basic<%>.

Calls shutdown.

Calls the super method.

```
(send a-drscheme:rep:text queue-output thnk) → void?
  thnk : (-> void?)
```

Specification: This method queues thunks for drscheme's eventspace in a special output-related queue.

(send *a-drscheme:rep:text* reset-console) → void?

Kills the old eventspace, and creates a new parameterization for it.

(send *a-drscheme: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-drscheme:rep:text* run-in-evaluation-thread *f*) → void?
  f : (-> void)

Specification: This function runs it's arguments in the user evaluation thread. This thread is the same as the user's eventspace main thread.

See also do-many-evals.

Default implementation: Calls *f*, after switching to the user's thread.

(send *a-drscheme: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 DrScheme's main eventspace thread.

(send *a-drscheme: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 DrScheme's eventspace

---

(send *a-drscheme: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

---

drscheme:rep:drs-bindings-keymap-mixin : (class? . -> . class?)
  argument extends/implements: editor:keymap<%>

This mixin adds some drscheme-specific keybindings to the editor it is mixed onto.

(send *a-drscheme: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 drscheme-specific keybindings:

- f5 - Run
- c:x;o - toggles the focus between the definition and interactions windows.

`drscheme:rep:context<%> : interface?`

Objects that match this interface provide all of the services that the `drscheme:rep:text%` class needs to connect with it's context.

`(send a-drscheme: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.

DrScheme calls this method before a program is run (via the Run button).

Default implementation: Clears any error highlighting in the definitions window.

`(send a-drscheme: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-drscheme: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-drscheme:rep:context ensure-rep-shown rep) → void?`
`  rep : (is-a?/c drscheme: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-drscheme:rep:context get-breakables)
 → (or/c thread? false/c)
   (or/c custodian? false/c)
```

Returns the last values passed to `set-breakables`.

```
(send a-drscheme:rep:context get-directory)
 → (union string false/c)
```

The result of this method is used as the initial directory for the user's program to be evaluated in.

```
(send a-drscheme: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-drscheme: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-drscheme: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-drscheme: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.

```
(drscheme: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.

```
(drscheme: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.

```
(drscheme:rep:get-drs-bindings-keymap) → (is-a?/c keymap%)
```

Returns a keymap that binds various DrScheme-specific keybindings. This keymap is used in the definitions and interactions window.

Defaultly 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.

```
(drscheme:rep:current-rep)
 → (or/c false/c (is-a?/c drscheme: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).

```
(drscheme: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.

15 drscheme:frame

```
drscheme:frame:name-message% : class?
  superclass: canvas%
```

This class implements the little filename button in the top-right hand side of drscheme's frame.

```
(make-object drscheme:frame:name-message% parent)
 → (is-a?/c drscheme:frame:name-message%)
  parent : (is-a?/c area-container<%>)
```

```
(send a-drscheme: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.

```
drscheme:frame:mixin : (class? . -> . class?)
  argument extends/implements: drscheme:frame:basics<%>
                               frame:text-info<%>
                               frame:editor<%>
  result implements: drscheme:frame:<%>
```

Provides an implementation of `drscheme:frame:<%>`

```
drscheme:frame:basics-mixin : (class? . -> . class?)
  argument extends/implements: frame:standard-menus<%>
  result implements: drscheme:frame:basics<%>
```

Use this mixin to establish some common menu items across various DrScheme windows.

```
(send a-drscheme:frame:basics edit-menu:between-find-and-preferences)
 → void?
```

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-drscheme: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-drscheme: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-drscheme: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 DrScheme window.

```
(send a-drscheme:frame:basics file-menu:new-string) → string?
```

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

Returns the empty string.

```
(send a-drscheme: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-drscheme:frame:basics file-menu:open-string) → string?
```

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

Returns the empty string.

```
(send a-drscheme: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 DrScheme'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: Defaultly returns the empty list.

```
(send a-drscheme: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 DrScheme.

```
(send a-drscheme:frame:basics help-menu:about-string)
 → string?
```

Overrides `help-menu:about-string` in `frame:standard-menus<%>`.

Returns the string `"DrScheme"`.

```
(send a-drscheme: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 DrScheme menu item.

```
(send a-drscheme:frame:basics help-menu:create-about?)
 → boolean?
```

Overrides `help-menu:create-about?` in `frame:standard-menus<%>`.

Returns `#t`.

drscheme:frame:basics<%> : interface?
  implements: frame:standard-menus<%>

This interface is the result of the drscheme:frame:basics-mixin

drscheme:frame:<%> : interface?
  implements: frame:editor<%>
              frame:text-info<%>
              drscheme:frame:basics<%>

(send a-drscheme: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. It is expected to add other Show/Hide menu items to the show menu.

See also get-show-menu.

Default implementation: Does nothing.

(send a-drscheme: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-drscheme:frame: not-running) → void?

updates the status pane at the bottom of the window to show that evaluation is not taking place in the user's program.

(send a-drscheme:frame: running) → void?

updates the status pane at the bottom of the window to show that evaluation is taking place in the user's program.

(send a-drscheme: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 DrScheme 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.

16 drscheme:help-desk

```
(drscheme:help-desk:help-desk [search-key
                               search-context]) → any
  search-key : (or/c #f string?) = #f
  search-context : (or/c #f string? (list/c string? string?))
                 = #f
```

if *search-key* is a string, calls perform-search with *search-key* and *search-context*.

Otherwise, calls send-main-page with no arguments.

17 drscheme:eval

```
(drscheme:eval:set-basic-parameters snipclasses) → void?
  snipclasses : (listof (is-a?/c snip-class%))
```

sets the parameters that are shared between the repl's initialization and `drscheme: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 copied (with `namespace-attach-module`) from DrScheme's original namespace:
  - `'mzscheme`
  - `'mred`
- `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.
- The snip-class-list, returned by `get-the-snip-class-list` is initialized with all of the snipclasses in DrScheme's eventspace's snip-class-list.

```
(drscheme:eval:get-snip-classes)
 → (listof (is-a?/c snip-class%))
```

Returns a list of all of the snipclasses in the current eventspace.

```
(drscheme:eval:expand-program input
                              language-settings
                              eval-compile-time-part?
                              init
                              kill-termination
                              iter)                    → void?
  input : (or/c port? drscheme:language:text/pos?)
  language-settings : drscheme: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)
```

Use this function to expand the contents of the definitions window for use with external program processing tools.

This function uses `drscheme:eval:build-user-eventspace/custodian` to build the user's environment. The arguments *`language-settings`*, *`init`*, and *`kill-termination`* are passed to `drscheme: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 `drscheme:eval:expand-program/multiple`.

```
(drscheme:eval:traverse-program/multiple language-settings
                                          init
                                          kill-termination)
 → ((or/c port? drscheme:language:text/pos?)
     ((or/c eof-object? syntax? (cons/c string? any/c))
      (-> any)
      . -> .
      any)
     boolean?
     . -> .
     void?)
  language-settings : drscheme:language-configuration:language-settings?
```

```
init : (-> void?)
kill-termination : (-> void?)
```

This function is similar to drscheme: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.

```
(drscheme:eval:expand-program/multiple
 language-settings
 eval-compile-time-part?
 init
 kill-termination)
→ (-> (or/c port? drscheme:language:text/pos?)
       (-> (or/c eof-object? syntax? (cons/c string? any/c))
           (-> any)
           any)
       boolean?
       void?)
 language-settings : drscheme:language-configuration:language-settings?
 eval-compile-time-part? : boolean?
 init : (-> void?)
 kill-termination : (-> void?)
```

This function is just like drscheme: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 drscheme:language:language front-end/complete-program<%> or drscheme:language:language front-end/interaction<%> is called.

```
(drscheme:eval:build-user-eventspace/custodian
 language-settings
 init
 kill-termination)
→ eventspace? custodian?
 language-settings : drscheme:language-configuration:language-settings?
 init : (-> void?)
 kill-termination : (-> void?)
```

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 drscheme:eval:set-basic-parameters.

The *language-settings* argument is the current language and its settings. See drscheme:language-configuration:make-language-settings for details on that structure.

If the program is associated with a DrScheme frame, get the frame's language settings from the get-next-settings method of drscheme:unit:definitions-text<%>. Also, the most recently chosen language in the language dialog is saved via the framework's preferences. Apply preferences:get to drscheme: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 drscheme:eval:build-user-eventspace/custodian was called from.)

18 drscheme:modes

```
(drscheme:modes:add-mode name
                         surrogate
                         repl-submit
                         matches-language)
 → drscheme:modes:mode?
  name : string?
  surrogate : (or/c false/c (is-a?/c mode:surrogate-text<%>))
  repl-submit : ((is-a?/c drscheme:rep:text%) number? . -> . boolean?)
  matches-language : ((or/c false/c (listof string?)) . -> . boolean?)
```

Adds a mode to DrScheme. Returns a mode value that identifies the mode.

The first argument, *name*, is the name of the mode, used in DrScheme'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.

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 `drscheme:modes:get-modes`.

```
(drscheme:modes:mode? val) → boolean?
  val : any/c
```

Determines if *val* is a mode.

```
(drscheme:modes:get-modes) → (listof drscheme:modes:mode?)
```

Returns all of the modes currently added to DrScheme.

See also drscheme:modes:add-mode.

```
(drscheme:modes:mode-name mode) → string?
  mode : drscheme:modes:mode?
```

Extracts the name of the mode.

See also drscheme:modes:add-mode.

```
(drscheme:modes:mode-surrogate mode)
 → (or/c false/c (is-a?/c mode:surrogate-text<%>))
  mode : drscheme:modes:mode?
```

Extracts the surrogate of the mode.

See also drscheme:modes:add-mode.

```
(drscheme:modes:mode-repl-submit mode) → any
  mode : drscheme:modes:mode?
```

Extracts the repl submission predicate of the mode.

See also drscheme:modes:add-mode.

```
(drscheme:modes:mode-matches-language mode)
 → ((or/c false/c (listof string?)) . -> . boolean?)
  mode : drscheme:modes:mode?
```

Extracts the language matching predicate of the mode.

See also drscheme:modes:add-mode.

Index