PLT Tools: DrScheme Extension Manual

Robert Bruce Findler (robby@cs.rice.edu)

Version 202
August 2002

Copyright notice

Send us your Web links

If you use any parts or all of the PLT Scheme package (software, lecture notes) for one of your courses, for your research, or for your work, we would like to know about it. Furthermore, if you use it and publicize the fact on some Web page, we would like to link to that page. Please drop us a line at *scheme@plt-scheme.org*. Evidence of interest helps the DrScheme Project to maintain the necessary intellectual and financial support. We appreciate your help.

Contents

1. This Manual

This manual describes DrScheme's tools interface. It assumes familiarity with DrScheme, as described in *PLT DrScheme: Development Environment Manual*, the Framework, as described in *PLT Framework: GUI Application Framework*, MrEd as described in *PLT MrEd: Graphical Toolbox Manual*, and MzScheme as described in *PLT MzScheme: Language Manual*.

1.1 Thanks

Thanks to Eli Barzilay, John Clements, Cormac Flanagan, Max Halipern, Philippe Meunier, and Christian Queinnec, PLT at large, and many others for their feedback and help.

This manual was typest using LaTeX, SLaTeX, and `tex2page`. Some typesetting macros were originally taken from Julian Smart's *Reference Manual for wxWindows 1.60: a portable C++ GUI toolkit.*

This manual was typeset on August 27, 2002.

2. 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 Static Debugger, the Syntax Checker, the Stepper, and the teaching languages are all implemented as tools.

Libraries are for extensions of DrScheme that only want to add new functions and other values bound in the users namespace. See the DrScheme manual for more information on constructing libraries.

Tools rely heavily on MzScheme's units. See units, §35 in *PLT MzScheme: Language Manual* for information on how to construct units. They also require understanding of libraries and collections, §16 in *PLT MzScheme: Language Manual*.

When DrScheme starts up, it looks for tools by reading fields in the **info.ss** file of each top-level collection. DrScheme checks for these fields:

tools (*listof* (*listof* *string*[*subcollection-name*]))

tool-names (*listof* (*union* #f *string*))

tool-icons (*listof* (*union* #f (`cons` *string*[*filename*] (*listof* *string*[*collection-name*]))))

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 sufficies:

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

If the *tool-icons* or *tool-names* fields are present, they must be the same length as *tool*. They specify the path to an icon for each tool and the name of each tool. This information shows up in the about box, Help Desk's bug report form, and the icon appears on the splash screen as the tool is loaded at DrScheme's startup.

Each of *tools* files must contain a module that **provide**s *tool@*, which must be bound to a `unit/sig`, §35 in *PLT MzLib: Libraries Manual* 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

Phase 2 functions:

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

If the tools 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:

```
(module info (lib "infotab.ss" "setup")
  (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:

```
(module tool mzscheme
  (require (lib "tool.ss" "drscheme")
           (lib "mred.ss" "mred")
           (lib "unitsig.ss"))

  (provide tool@)

  (define tool@
    (unit/sig drscheme:tool-exports^
      (import drscheme:tool^)
      (define (phase1)
        (message-box "tool example" "phase1"))
      (define (phase2)
        (message-box "tool example" "phase2")))))
```

This tool just opens a window to indicate that it has been loaded.

2.1 Adding Languages to DrScheme

2.1.1 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, 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.

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.

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**.

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-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**, except not plain strings). 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.

The lists must have the same length.

As an example, the *Essentials of Programming Languages* language specification's **info.ss** looks like this:

```
(module info (lib "infotab.ss" "setup")
  (define name "EoPL Support")
  (define drscheme-language-modules
    (list '("eopl-lang.ss" "eopl")))
  (define drscheme-language-positions
    (list '("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 **eopl** collection. Additionally, the langauge dialog will contain Essentials of Programming Languages as a potential language.

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.

2.1.2 Adding Arbitrary Languages to DrScheme

With some additional work, any language that can be compiled to MzScheme's language 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-ba` and mixins, §3.2 in *PLT Framework: GUI Application Framework* `drscheme:language:simple-module-based-language->m` 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-configuratio` 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.

2.1.3 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 programs, 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 apriori 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.

2.2 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.

2.3 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

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
rawscmmethod1.

2.4 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 also 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.

3. Tools Reference

3.1 drscheme:debug:profile-definitions-text-mixin

Domain: drscheme:unit:definitions-text<%>

Domain: (class->interface text%)

Implements: drscheme:unit:definitions-text<%>

- **(instantiate** drscheme:debug:profile-definitions-text-mixin% () [**(line-spacing** _)] [**(tab-stops** _)] [**(auto-wrap** _)]) ⇒ drscheme:debug:profile-definitions-text-mixin% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

3.2 drscheme:debug:profile-interactions-text-mixin

Domain: drscheme:rep:text<%>

Implements: drscheme:rep:text<%>

3.3 drscheme:debug:profile-unit-frame-mixin

Domain: drscheme:frame:<%>

Domain: drscheme:unit:frame<%>

Implements: drscheme:frame:<%>

Implements: drscheme:unit:frame<%>

3.4 drscheme:frame:<%>

Extends: frame:text-info<%>

Extends: frame:editor<%>

Extends: drscheme:frame:basics<%>

add-show-menu-items

This method is called during the construction of the show 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.

- (**send** *a-drscheme:frame:* add-show-menu-items *show-menu*) ⇒ void
  show-menu : (is-a?/c menu%)

  Does nothing.

get-show-menu

returns the show menu, for use by the update-shown method.

See also add-show-menu-items.

- (**send** *a-drscheme:frame:* get-show-menu) ⇒ (instanceof menu%)

not-running

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:* not-running) ⇒ void

running

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:* running) ⇒ void

`update-shown`

This method is intended to be overridden. It's job is to update the `"Show"` 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 visiblity of the definitions and interaction `editor-canvas%`s.

Call this method whenever the state of the show menu might need to change.

- **(send** *a-drscheme:frame:* `update-shown`**)** ⇒ void

  Does nothing.

3.5 drscheme:frame:basics<%>

Extends: `frame:standard-menus<%>`

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

3.6 drscheme:frame:basics-mixin

Domain: `frame:standard-menus<%>`

Implements: `frame:standard-menus<%>`

Implements: `drscheme:frame:basics<%>`

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

`edit-menu:between-find-and-preferences`

This method is called between the addition of the find menu-item and before the addition of the preferences menu-item to the edit-menu menu. Override it to add additional menus at that point.

- **(send** *a-drscheme:frame:basics-mixin* `edit-menu:between-find-and-preferences`**)** ⇒ void

  Adds the `"Keybindings"` menu item to the edit menu.

`file-menu:between-open-and-revert`

This method is called between the addition of the open menu-item and before the addition of the revert menu-item to the file-menu menu. Override it to add additional menus at that point.

- **(send** *a-drscheme:frame:basics-mixin* `file-menu:between-open-and-revert` *file-menu***)** ⇒ void
  file-menu : (instance `menu%`)

  Adds an *Install .plt File...* menu item, which downloads and installs .plt files from the web, or installs them from the local disk.

`file-menu:new-callback`

This method is called when the new menu-item of the file-menu menu is selected.

- (**send** *a-drscheme:frame:basics-mixin* `file-menu:new-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Opens a new, empty DrScheme window.

`file-menu:new-string`

The result of this method is the name of this menu.

- (**send** *a-drscheme:frame:basics-mixin* `file-menu:new-string`) ⇒ string
  Returns the empty string.

`file-menu:open-callback`

This method is called when the open menu-item of the file-menu menu is selected.

- (**send** *a-drscheme:frame:basics-mixin* `file-menu:open-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Calls `handler:edit-file`.

`file-menu:open-string`

The result of this method is the name of this menu.

- (**send** *a-drscheme:frame:basics-mixin* `file-menu:open-string`) ⇒ string
  Returns the empty string.

`help-menu:about-callback`

This method is called when the about menu-item of the help-menu menu is selected.

- (**send** *a-drscheme:frame:basics-mixin* `help-menu:about-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Opens an about box for DrScheme.

`help-menu:about-string`

The result of this method is the name of this menu.

- (**send** *a-drscheme:frame:basics-mixin* `help-menu:about-string`) ⇒ string
  Returns the string `"DrScheme"`.

`help-menu:before-about`

This method is called before the addition of the about menu-item to the help-menu menu. Override it to add additional menus at that point.

- (**send** *a-drscheme:frame:basics-mixin* `help-menu:before-about` *help-menu*) ⇒ void
  help-menu : (instance menu%)

  Adds the Help Desk menu item and the Welcome to DrScheme menu item.

`help-menu:create-about?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

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

  Returns `#t`.

3.7 drscheme:frame:mixin

Domain: frame:text-info<%>

Domain: frame:editor<%>

Domain: drscheme:frame:basics<%>

Implements: drscheme:frame:<%>

Implements: frame:text-info<%>

Implements: frame:editor<%>

Implements: drscheme:frame:basics<%>

Provides an implementation of drscheme:frame:<%>

3.8 drscheme:frame:name-message%

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*) ⇒ drscheme:frame:name-message% object
  parent : (instance (implements area-container<%>))

`set-message`

Sets the names that the button shows.

- **(send** *a-drscheme:frame:name-message* `set-message` *name short-name*) ⇒ void
  name : (union string #f)
  short-name : string

  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.

3.9 drscheme:get/extend:base-definitions-canvas% = drscheme:unit:definitions-canvas%

`drscheme:get/extend:base-definitions-canvas% = drscheme:unit:definitions-canvas%`

- **(instantiate** `drscheme:get/extend:base-definitions-canvas%` **() (parent _) [(editor _)] [(style _)] [(scrolls-per-page _)] [(label _)] [(wheel-step _)] [(line-count _)] [(enabled _)] [(vert-margin _)] [(horiz-margin _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])** ⇒ drscheme:get/extend:base-definitions-canvas% object
  parent : frame%, dialog%, panel%, or pane% object
  editor = #f : text% or pasteboard% object or #f
  style = null : list of symbols in `'(no-hscroll no-vscroll hide-hscroll hide-vscroll)`
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If a canvas is initialized with #f for *editor*, install an editor later with `set-editor`.

  The *style* list can contain the following flags:

  - 'no-hscroll — disallows horizontal scrolling
  - 'no-vscroll — disallows vertical scrolling
  - 'hide-hscroll — allows horizontal scrolling, but hides the horizontal scrollbar
  - 'hide-vscroll — allows vertical scrolling, but hides the vertical scrollbar

  While vertical scrolling of text editors is based on lines, horizontal scrolling and pasteboard vertical scrolling is based on a fixed number of steps per horizontal page. The *scrolls-per-page* argument sets this value.

  If provided, the *wheel-step* argument is passed on to the `wheel-step` method. The default wheel step can be overridden globally though the '|MrEd:wheelStep| preference; see "Preferences" (section 12, page 325).

  If *line-count* is not #f, it is passed on to the `set-line-count` method.

  For information about the *enabled* argument, see `window<%>`. For information about the *horiz-margin* and *vert-margin* arguments, see `subarea<%>`. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see `area<%>`.

3.10 drscheme:get/extend:base-definitions-text% = (drscheme:debug:profile-definitions-t drscheme:unit:definitions-text%)

drscheme:get/extend:base-definitions-text% = (drscheme:debug:profile-definitions-text-mixin drscheme:unit:definitions-text%)

- **(instantiate** drscheme:get/extend:base-definitions-text% () [**(line-spacing** _)] [**(tab-stops** _)] [**(auto-wrap** _)]) ⇒ drscheme:get/extend:base-definitions-text% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

3.11 drscheme:get/extend:base-interactions-canvas% = drscheme:unit:interactions-canvas%

drscheme:get/extend:base-interactions-canvas% = drscheme:unit:interactions-canvas%

- **(instantiate** drscheme:get/extend:base-interactions-canvas% () **(parent** _) [**(editor** _)] [**(style** _)] [**(scrolls-per-page** _)] [**(label** _)] [**(wheel-step** _)] [**(line-count** _)] [**(enabled** _)] [**(vert-margin** _)] [**(horiz-margin** _)] [**(min-width** _)] [**(min-height** _)] [**(stretchable-width** _)] [**(stretchable-height** _)]) ⇒ drscheme:get/extend:base-interactions-canvas% object
  parent : frame%, dialog%, panel%, or pane% object
  editor = #f : text% or pasteboard% object or #f
  style = null : list of symbols in '(no-hscroll no-vscroll hide-hscroll hide-vscroll)
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If a canvas is initialized with #f for *editor*, install an editor later with set-editor.

  The *style* list can contain the following flags:

  – 'no-hscroll — disallows horizontal scrolling
  – 'no-vscroll — disallows vertical scrolling
  – 'hide-hscroll — allows horizontal scrolling, but hides the horizontal scrollbar
  – 'hide-vscroll — allows vertical scrolling, but hides the vertical scrollbar

While vertical scrolling of text editors is based on lines, horizontal scrolling and pasteboard vertical scrolling is based on a fixed number of steps per horizontal page. The *scrolls-per-page* argument sets this value.

If provided, the *wheel-step* argument is passed on to the wheel-step method. The default wheel step can be overridden globally though the '|MrEd:wheelStep| preference; see "Preferences" (section 12, page 325).

If *line-count* is not #f, it is passed on to the set-line-count method.

For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

3.12 drscheme:get/extend:base-interactions-text% = (drscheme:debug:profile-interactions drscheme:rep:text%)

drscheme:get/extend:base-interactions-text% = (drscheme:debug:profile-interactions-text-mixin drscheme:rep:text%)

- (make-object drscheme:get/extend:base-interactions-text% *context*) ⇒ drscheme:get/extend:base-interac object
  context : (implements drscheme:rep:context<%>)

3.13 drscheme:get/extend:base-unit-frame% = (drscheme:debug:profile-unit-frame-mixin drscheme:unit:frame%)

drscheme:get/extend:base-unit-frame% = (drscheme:debug:profile-unit-frame-mixin drscheme:unit:frame%)

- (**instantiate** drscheme:get/extend:base-unit-frame% () (**label** _) [(**parent** _)] [(**width** _)] [(**height** _)] [(**x** _)] [(**y** _)] [(**style** _)] [(**enabled** _)] [(**border** _)] [(**spacing** _)] [(**alignment** _)] [(**min-width** _)] [(**min-height** _)] [(**stretchable-width** _)] [(**stretchable-height** _)]) ⇒ drscheme:get/extend:base-unit-frame% object
  label : string (up to 200 characters)
  parent = #f : frame% object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in '(no-resize-border no-caption no-system-menu mdi-parent mdi-child)
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  The *label* string is displayed in the frame's title bar. If the frame's label is changed (see set-label), the title bar is updated.

  The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame

from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an exn:application:mismatch exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see window<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

3.14 drscheme:language:language<%>

Implementations of this interface are languages that DrScheme supports.

See §2.1 for an overview of adding languages to DrScheme.

config-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* config-panel *parent*) ⇒ (case-¿ (-¿ settings) (settings -¿ void))
  parent : (instanceof panel%)

create-executable

This method creates an executable in the given language. The *program-filename* is the name of the program to store in the executable and *exectuable-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-b

- (**send** *a-drscheme:language:language* `create-executable` *settings parent program-filename*) ⇒ void
  settings : settings
  parent : (union (instanceof `dialog%`) (instanceof `frame%`))
  program-filename : string

`default-settings`

Specifies the default settings for this language.

- (**send** *a-drscheme:language:language* `default-settings`) ⇒ settings

`default-settings?`

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

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

`front-end`

`front-end` method reads, parses, and optionally compiles a program in the language. The first argument to the method specifies the location of the unparsed input program. The selector `drscheme:language:text/pos-text` extracts the `text%` text and the `drscheme:language:text/pos-start` and `drscheme:language:text/pos-end` selectors extract the range in the text that should be considered program text. The second argument is the current settings for the language. The `front-end` 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 and interactions window. Notably, it is not called for programs that are `load`ed or `eval`ed. See and 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—, §12.6.1 in *PLT MzScheme: Language Manual* or *expand-top-level-with-compile-time-evals* and the use of the expanded code dictates which applies.

- (**send** *a-drscheme:language:language* `front-end` *input settings*) ⇒ (-¿ (union sexp syntax eof))
  input : text/pos
  settings : settings

`get-language-name`

Returns the name of the language as shown in the REPL when executing programs in the language.

- (**send** *a-drscheme:language:language* `get-language-name`) ⇒ string

`get-language-numbers`

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

- **(send** *a-drscheme:language:language* `get-language-numbers`**)** ⇒ (cons number (listof number))

`get-language-position`

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-position`**)** ⇒ (cons string (listof string))

`get-one-line-summary`

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

- **(send** *a-drscheme:language:language* `get-one-line-summary`**)** ⇒ string

`get-style-delta`

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* `get-style-delta`**)** ⇒ (union #f (instanceof `style-delta%`) (listof (list (instanceof `style-delta%`) number number)))

`marshall-settings`

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

- **(send** *a-drscheme:language:language* `marshall-settings` *settings***)** ⇒ writable
  settings : settings

`on-execute`

The `on-execute` method is called before any evaluation happens during execution. Use this method to initialize MzScheme's parameters, §7.4 in *PLT MzScheme: Language Manual* 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")`
- `break-enabled` is `#t`
- `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 current-load parameter is set to a procedure calls the language's `front-end` method, instead of jus using `read`.
- 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.

- (**send** *a-drscheme:language:language* `on-execute` *settings run-in-user-thread*) ⇒ vod
  settings : settings
  run-in-user-thread : ((-¿ void) -¿ void)

`render-value`

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` *value settings port port-write*) ⇒ void
  value : TST
  settings : settings
  port : port
  port-write : (union #f ((instanceof `snip%`)-¿ void))

`render-value/format`

This method is used to turn values into strings for displaying. The final argument is used because ports aren't enough to support the printing of snips into a `text%`. Calling the third argument with a snip inserts the snip into an `text%` that corresponds to displaying something in the port. If the third argument is `#f`, we are in a context that cannot display snips, so this methods are expected to flatten any snips into strings. 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 vaue.

See also `render-value`.

- (**send** *a-drscheme:language:language* `render-value/format` *value settings port port-write width*) ⇒ void
  value : TST
  settings : settings
  port : port
  port-write : (union #f ((instanceof `snip%`) -¿ void))
  width : (union #f number)

`unmarshall-settings`

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

- (**send** *a-drscheme:language:language* `unmarshall-settings` *input*) ⇒ (union settings #f)
  input : writable

3.15 drscheme:language:module-based-language<%>

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* `:`) ⇒ procedure with the same type as `read-syntax`

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.

`config-panel`

This method is the same as config-panel.

- (**send** *a-drscheme:language:module-based-language* `config-panel` *parent*) ⇒ (case-¿ (-¿ settings) (settings -¿ void))
  parent : (instanceof panel%)

`default-settings`

This method is the same as default-settings.

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

`default-settings?`

This method is the same as default-settings?.

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

`get-init-code`

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-init-code` *settings*) ⇒ sexp
  settings : settings

`get-language-numbers`

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

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

`get-language-position`

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

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

`get-module`

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

The result is expected to be the specification of a module except as value, ie `quoted`.

See also `get-transformer-module`.

- **(send** *a-drscheme:language:module-based-language* `get-module`**)** ⇒ s-expression

`get-one-line-summary`

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-one-line-summary`**)** ⇒ string

`get-transformer-module`

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 specification of a module except as value, ie `quoted`.

See also `get-module`.

- **(send** *a-drscheme:language:module-based-language* `get-transformer-module`**)** ⇒ s-expression

`marshall-settings`

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

- **(send** *a-drscheme:language:module-based-language* `marshall-settings` *settings***)** ⇒ writable
  settings : settings

`on-execute`

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

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

`render-value`

This method is the same as `render-value`.

- **(send** *a-drscheme:language:module-based-language* `render-value` *value settings port port-write***)** ⇒ void
  value : TST

settings : settings
port : port
port-write : (union #f ((instanceof snip%) -¿ void))

`render-value/format`

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

- (**send** *a-drscheme:language:module-based-language* `render-value/format` *value settings port port-write*) ⇒ void
  value : TST
  settings : settings
  port : port
  port-write : (union #f ((instanceof snip%) -¿ void))

`unmarshall-settings`

This method is the same as unmarshall-settings.

- (**send** *a-drscheme:language:module-based-language* `unmarshall-settings` *input*) ⇒ (union settings #f)
  input : writable

`use-mred-launcher`

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-mred-launcher`) ⇒ boolean

`use-namespace-require/copy?`

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.

- (**send** *a-drscheme:language:module-based-language* `use-namespace-require/copy?`) ⇒ boolean
  Defaultly returns `#f`.

3.16 drscheme:language:module-based-language->language-mixin

Domain: drscheme:language:module-based-language<%>

Implements: drscheme:language:module-based-language<%>

Implements: drscheme:language:language<%>

`front-end`

`front-end` method reads, parses, and optionally compiles a program in the language. The first argument to the method specifies the location of the unparsed input program. The selector `drscheme:language:text/pos-text` extracts the `text%` text and the `drscheme:language:text/pos-start` and `drscheme:language:text/pos-end` selectors extract the range in the text that should be considered program text. The second argument is the current settings for the language. The `front-end` 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 and interactions window. Notably, it is not called for programs that are `load`ed or `eval`ed. See and 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—, §12.6.1 in *PLT MzScheme: Language Manual* or *expand-top-level-with-compile-time-evals* and the use of the expanded code dictates which applies.

- **(send** *a-drscheme:language:module-based-language-¿language-mixin* `front-end` *input settings*) ⇒ (-¿ (union sexp syntax eof))
  input : text/pos
  settings : settings

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

`get-language-name`

Returns the name of the language as shown in the REPL when executing programs in the language.

- **(send** *a-drscheme:language:module-based-language-¿language-mixin* `get-language-name`**)** ⇒ string

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

`on-execute`

The `on-execute` method is called before any evaluation happens during execution. Use this method to initialize MzScheme's parameters, §7.4 in *PLT MzScheme: Language Manual* 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")`
- `break-enabled` is `#t`
- `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 current-load parameter is set to a procedure calls the language's `front-end` method, instead of jus using `read`.
- 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.

- (**send** *a-drscheme:language:module-based-language-¿language-mixin* `on-execute` *settings run-in-user-thread*) ⇒ void
  settings : settings
  run-in-user-thread : ((-¿ void) -¿ void)

  Calls the super method.

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

3.17 drscheme:language:simple-module-based-language<%>

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.

`get-language-numbers`

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-numbers`) ⇒ (cons number (listof number))

`get-language-position`

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

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

`get-module`

This method specifies the module that defines the language.

This method replaces `front-end`.

The result is expected to be the specification of a module except as value, ie `quoted`.

- (**send** *a-drscheme:language:simple-module-based-language* `get-module`) ⇒ s-expression

`get-one-line-summary`

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-one-line-summary`) ⇒ string

3.18 drscheme:language:simple-module-based-language%

Implements: `drscheme:language:simple-module-based-language<%>`

- (`make-object drscheme:language:simple-module-based-language%` *module language-position language-numbers one-line-summary documentation-reference*) ⇒ `drscheme:language:simple-module-based-language%` object
  module : s-expression
  language-position : (cons string (listof string))
  language-numbers = (map (lambda (x) 0) language-position) : (cons number (listof number))
  one-line-summary = "" : string
  documentation-reference = #f : (union #f something-else)

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

`get-language-numbers`

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-numbers`) ⇒ (cons number (listof number))

  returns the corresponding init arg.

`get-language-position`

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

- **(send** *a-drscheme:language:simple-module-based-language* `get-language-position`**)** ⇒ s-expression
  returns the corresponding init arg.

`get-module`

This method specifies the module that defines the language.

This method replaces `front-end`.

The result is expected to be the specification of a module except as value, ie `quoted`.

- **(send** *a-drscheme:language:simple-module-based-language* `get-module`**)** ⇒ (cons string (listof string))
  returns the corresponding init arg.

`get-one-line-summary`

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-one-line-summary`**)** ⇒ string
  returns the corresponding initialization argument.

3.19 drscheme:language:simple-module-based-language->module-based-language-mixin

Domain: `drscheme:language:simple-module-based-language<%>`

Implements: `drscheme:language:module-based-language<%>`

Implements: `drscheme:language:simple-module-based-language<%>`

This mixin uses a struct definition for its settings:

```
(define-struct drscheme:language:simple-settings (case-sensitive printing-style fraction-style show-sha
;; case-sensitive  : boolean
;; printing-style  : (union 'constructor 'quasiquote 'write)
;; fraction-style  : (union 'mixed-fraction 'mixed-fraction-e 'repeating-decimal 'repeating-decimal-e)
;; show-sharing    : boolean
;; insert-newlines : boolean
;; debugging       : boolean
```

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.

`config-panel`

This method is the same as `config-panel`.

- **(send** *a-drscheme:language:simple-module-based-language-¿module-based-language-mixin* `config-panel` *parent***)** ⇒ (case-¿ (-¿ settings) (settings -¿ void))
  parent : (instanceof `panel%`)

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

  See also §3.19 for details of the simple-settings structure, this mixins `settings` type.

`default-settings`

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

- **(send** *a-drscheme:language:simple-module-based-language-¿module-based-language-mixin* `default-settings`**)** ⇒ settings

  The defaults for the settings are

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

  See also §3.19 for details of the simple-settings structure, this mixins `settings` type.

`default-settings?`

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

- **(send** *a-drscheme:language:simple-module-based-language-¿module-based-language-mixin* `default-settings?` *settings***)** ⇒ boolean
  settings : settings

`get-init-code`

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:simple-module-based-language-¿module-based-language-mixin* `get-init-code` *settings***)** ⇒ sexpression
  settings : settings

  Creates an sexpression 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.

`get-transformer-module`

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 specification of a module except as value, ie `quoted`.

See also `get-module`.

- **(send** *a-drscheme:language:simple-module-based-language-¿module-based-language-mixin* `get-transformer-module`**)** ⇒ s-expression

  Returns `'mzscheme`.

`marshall-settings`

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

- **(send** *a-drscheme:language:simple-module-based-language-¿module-based-language-mixin* `marshall-settings` *settings***)** ⇒ writable
  settings : settings

  Constructs a vector from the structure.

  See also §3.19 for details of the simple-settings structure, this mixins `settings` type.

`on-execute`

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

- **(send** *a-drscheme:language:simple-module-based-language-¿module-based-language-mixin* `on-execute` *settings run-in-user-thread***)** ⇒ void
  settings : settings
  run-in-user-thread : ((-¿ void) -¿ void)

  Sets the case sensitivity of the language.

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

  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 §3.19 for details of the simple-settings structure, this mixin's `settings` type.

`render-value`

This method is the same as `render-value`.

- **(send** *a-drscheme:language:simple-module-based-language-¿module-based-language-mixin* `render-value` *value settings port port-write***)** ⇒ void
  value : TST
  settings : settings
  port : port
  port-write : (union #f ((instanceof `snip%`) -¿ void))

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

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

See also §3.19 for details of the simple-settings structure, this mixins settings type.

render-value/format

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

- **(send** *a-drscheme:language:simple-module-based-language-¿module-based-language-mixin* render-value/format *value settings port port-write*) ⇒ void
  value : TST
  settings : settings
  port : port
  port-write : (union #f ((instanceof snip%) -¿ void))

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

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

  See also §3.19 for details of the simple-settings structure, this mixins settings type.

unmarshall-settings

This method is the same as unmarshall-settings.

- **(send** *a-drscheme:language:simple-module-based-language-¿module-based-language-mixin* unmarshall-settings *input*) ⇒ (union #f settings)
  input : writable

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

  See also §3.19 for details of the simple-settings structure, this mixins settings type.

use-mred-launcher

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:simple-module-based-language-¿module-based-language-mixin* use-mred-launcher) ⇒ boolean

  Returns #t.

3.20 drscheme:rep:context<%>

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

disable-evaluation

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.

See also enable-evaluation.

- **(send** *a-drscheme:rep:context* disable-evaluation**)** ⇒ void

enable-evaluation

This method must disable all user-sponsored evaluation. It is called once the user starts some evaluation to ensure that only one evaluation proceeds at a time.

See also enable-evaluation.

- **(send** *a-drscheme:rep:context* enable-evaluation**)** ⇒ void

ensure-rep-shown

- **(send** *a-drscheme:rep:context* ensure-rep-shown**)** ⇒ void
  This method is called to force the rep window to be visible when, for example, an error message is put into the rep.

get-breakables

Returns the last values passed to set-breakables.

- **(send** *a-drscheme:rep:context* get-breakables**)** ⇒ (values (union thread #f) (union custodian #f))

get-directory

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

- **(send** *a-drscheme:rep:context* get-directory**)** ⇒ : (union string #f)

needs-execution?

This method should return #t when the state of the program that the repl reflects has changed.

- **(send** *a-drscheme:rep:context* needs-execution?**)** ⇒ boolean

not-running

- **(send** *a-drscheme:rep:context* not-running**)** ⇒ void
  This method should update some display in the gui that indicates no evaluation is currently proceeding in the user's world.

reset-offer-kill

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* `reset-offer-kill`) ⇒ void

`running`

- (**send** *a-drscheme:rep:context* `running`) ⇒ void

  This method should update some display in the gui that indicates evaluation is currently proceeding in the user's world.

`set-breakables`

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* `set-breakables` *thread* *custodian*) ⇒ void
  thread : (union thread #f)
  custodian : (union custodian #f)

3.21 drscheme:rep:drs-bindings-keymap-mixin

Domain: `editor:keymap<%>`

Implements: `editor:keymap<%>`

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

`get-keymaps`

The keymaps returned from this method are chained to this `editor<%>`'s keymap.

- (**send** *a-drscheme:rep:drs-bindings-keymap-mixin* `get-keymaps`) ⇒ (listof (instanceof `keymap%`))

  Calls the super method and adds in a keymap with the drscheme-specific keybindings:

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

3.22 drscheme:rep:text<%>

3.23 drscheme:rep:text%

Implements: `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*) ⇒ drscheme:rep:text% object
  context : (implements drscheme:rep:context<%>)

after-delete

Called after a given range is deleted from the editor (and after the display is refreshed; use on-delete and begin-edit-sequence to avoid extra refreshes when after-delete modifies the editor).

See also can-delete? and on-edit-sequence.

No internals locks are set when this method is called.

- (**send** *a-drscheme:rep:text* after-delete *start end*) ⇒ void
  start : exact non-negative integer
  end : exact non-negative integer

  Resets any error highlighting in this editor.

after-insert

Called after items are inserted into the editor (and after the display is refreshed; use on-insert and begin-edit-sequence to avoid extra refreshes when after-insert modifies the editor).

See also can-insert? and on-edit-sequence.

No internals locks are set when this method is called.

- (**send** *a-drscheme:rep:text* after-insert *start len*) ⇒ void
  start : exact non-negative integer
  len : exact non-negative integer

  Resets any error highlighting in this editor.

cleanup-transparent-io

Resets the little I/O box so that the next I/O goes into a newly created box in the repl.

This method is called when a value is printed to the repl or a prompt is inserted into the repl.

This method expects to be called on DrScheme's main eventspace thread.

- (**send** *a-drscheme:rep:text* cleanup-transparent-io) ⇒ void

display-results

- (**send** *a-drscheme:rep:text* display-results *results*) ⇒ void
  results : (list-of TST)

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

`do-many-evals`

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.

- (**send** *a-drscheme:rep:text* `do-many-evals` *run-loop*) ⇒ void
  run-loop : (((-¿ void) -¿ void) -¿ void)

  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.

`do-many-text-evals`

This function evaluates all of the expressions in a text.

- (**send** *a-drscheme:rep:text* `do-many-text-evals` *text start end*) ⇒ void
  text : a `text%` object
  start : int
  end : int

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

`get-error-range`

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.

- (**send** *a-drscheme:rep:text* `get-error-range`) ⇒ (union #f (list (instanceof `text:basic%` number number)))

  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.

`get-this-err`

This method returns the standard error port that prints in the repl.

- (**send** *a-drscheme:rep:text* `get-this-err`) ⇒ port

`get-this-in`

- (**send** *a-drscheme:rep:text* `get-this-in`) ⇒ input-port

  returns the input port for this repl.

`get-this-out`

This method returns the standard output port that prints in the repl.

- (**send** *a-drscheme:rep:text* `get-this-out`) ⇒ port

`get-this-result`

This method returns a port used to write values that show up in the repl.

- **(send** *a-drscheme:rep:text* `get-this-result`**)** ⇒ port

`get-user-custodian`

This is the custodian controlling the user's program.

- **(send** *a-drscheme:rep:text* `get-user-custodian`**)** ⇒ (union #f custodian)

`get-user-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-eventspace`**)** ⇒ (union #f eventspace)

`get-user-namespace`

Returns the user's namespace. This method returns a new namespace each time execute is clicked.

- **(send** *a-drscheme:rep:text* `get-user-namespace`**)** ⇒ (union #f namespace)

`get-user-thread`

This method returns the thread that the users code runs in. It is returns a different result, each time the user executes the program.

It is `#f` before the first time the user click on the Execute button or the evaluation has been killed.

This thread has all of its parameters initialized according to the settings of the current execution. See parameters, §7.4 in *PLT MzScheme: Language Manual* for more information about parameters.

- **(send** *a-drscheme:rep:text* `get-user-thread`**)** ⇒ (union #f thread)

`hide-eof-icon`

- **(send** *a-drscheme:rep:text* `hide-eof-icon`**)** ⇒ void

  Hides the eof icon for the input port in this repl. See also `show-eof-icon`.

`highlight-error`

Call this method to highlight an error associated with this repl. See also `highlight-errors`, `reset-highlighting`, `highlight-error/line-col`and `highlight-error/forward-sexp`.

This method highlights a series of dis-contiguous ranges in the editor.

- (**send** *a-drscheme:rep:text* `highlight-error` *text start end*) ⇒ void
  text : text:basic<%>
  start : small-integer
  end : small-integer

`highlight-error/forward-sexp`

Call this method to highlight an error associated with this repl. This method uses the paren matching library in DrScheme to determine the end position of the error.

See also `reset-highlighting` and `highlight-error`.

- (**send** *a-drscheme:rep:text* `highlight-error/forward-sexp` *text start-loc*) ⇒ void
  text : (instance (implements text:basic<%>))
  start-loc : small-integer

`highlight-error/line-col`

Call this method to highlight an error associated with this repl. See also `reset-highlighting`, `highlight-error`and `highlight-error/forward-sexp`.

- (**send** *a-drscheme:rep:text* `highlight-error/line-col` *text start-line start-col end-line end-col*) ⇒ void
  text : (instance (implements text:basic<%>))
  start-line : number
  start-col : number
  end-line : number
  end-col : number

`highlight-errors`

Call this method to highlight an error associated with this repl. See also `reset-highlighting`, `highlight-error/line-col`and `highlight-error/forward-sexp`.

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` *locs*) ⇒ void
  locs : (listof (list (instance (implements text:basic<%>)) small-integer small-integer))

`initialize-console`

- (**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`.

`insert-prompt`

- (**send** *a-drscheme:rep:text* `insert-prompt`) ⇒ void
  Inserts a new prompt at the end of the text.

`kill-evaluation`

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

- **(send** *a-drscheme:rep:text* `kill-evaluation`**)** ⇒ void

`on-close`

This method is called when a frame that shows this buffer is closed.

- **(send** *a-drscheme:rep:text* `on-close`**)** ⇒ void
  Calls `shutdown`.
  Calls the super method.

`queue-output`

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

- **(send** *a-drscheme:rep:text* `queue-output` *thnk***)** ⇒ void
  thnk : (-¿ void?)

`reset-console`

- **(send** *a-drscheme:rep:text* `reset-console`**)** ⇒ void
  Kills the old eventspace, and creates a new parameterization

`reset-highlighting`

This method resets the highlighting being displayed for this repl. See also: `highlight-error`, `highlight-error/line-col` and `highlight-error/forward-sexp`.

- **(send** *a-drscheme:rep:text* `reset-highlighting`**)** ⇒ void

`run-in-evaluation-thread`

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`.

- **(send** *a-drscheme:rep:text* `run-in-evaluation-thread` *f***)** ⇒ void
  f : (-¿ void)
  Calls *f*, after switching to the user's thread.

`show-eof-icon`

- **(send** *a-drscheme:rep:text* `show-eof-icon`**)** ⇒ void
  Shows the eof icon for the input port in this repl. Clicking on the icon calls the `submit-eof` method.
  See also `hide-eof-icon`.

`shutdown`

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* `shutdown`**)** ⇒ void

`submit-eof`

- **(send** *a-drscheme:rep:text* `submit-eof`**)** ⇒ void
  Submits an eof to the input port for this repl.

`submit-eof`

- **(send** *a-drscheme:rep:text* `submit-eof`**)** ⇒ void

`this-err-write`

- **(send** *a-drscheme:rep:text* `this-err-write` *to-display*) ⇒ void
  to-display : (union string (instanceof (derivedfrom snip%)))
  displays *to-display* on the error port for this repl.

`this-out-write`

- **(send** *a-drscheme:rep:text* `this-out-write` *to-display*) ⇒ void
  to-display : (union string (instanceof (derivedfrom snip%)))
  displays *to-display* on the output port for this repl.

`this-result-write`

- **(send** *a-drscheme:rep:text* `this-result-write` *to-display*) ⇒ void
  to-display : (union string (instanceof (derivedfrom snip%)))
  displays *to-display* on the value display port for this repl.

`wait-for-io-to-complete`

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`**)** ⇒ void

`wait-for-io-to-complete/user`

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

- **(send** *a-drscheme:rep:text* `wait-for-io-to-complete/user`**)** ⇒ void

3.24 drscheme:unit:definitions-canvas%

Superclass: editor-canvas%

- **(instantiate drscheme:unit:definitions-canvas% () (parent _) [(editor _)] [(style _)] [(scrolls-per-page _)] [(label _)] [(wheel-step _)] [(line-count _)] [(enabled _)] [(vert-margin _)] [(horiz-margin _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])** ⇒ drscheme:unit:definitions-canvas% object
  parent : frame%, dialog%, panel%, or pane% object
  editor = #f : text% or pasteboard% object or #f
  style = null : list of symbols in '(no-hscroll no-vscroll hide-hscroll hide-vscroll)
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If a canvas is initialized with #f for *editor*, install an editor later with set-editor.

  The *style* list can contain the following flags:

  - 'no-hscroll — disallows horizontal scrolling
  - 'no-vscroll — disallows vertical scrolling
  - 'hide-hscroll — allows horizontal scrolling, but hides the horizontal scrollbar
  - 'hide-vscroll — allows vertical scrolling, but hides the vertical scrollbar

  While vertical scrolling of text editors is based on lines, horizontal scrolling and pasteboard vertical scrolling is based on a fixed number of steps per horizontal page. The *scrolls-per-page* argument sets this value.

  If provided, the *wheel-step* argument is passed on to the wheel-step method. The default wheel step can be overridden globally though the '|MrEd:wheelStep| preference; see "Preferences" (section 12, page 325).

  If *line-count* is not #f, it is passed on to the set-line-count method.

  For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

3.25 drscheme:unit:definitions-text<%>

This interface is implemented by the definitions text.

get-next-settings

This method returns the language-settings that will be used on the execute in this DrScheme window.

- **(send** *a-drscheme:unit:definitions-text* get-next-settings**)** ⇒ language-settings

3.26 drscheme:unit:definitions-text% = (drscheme:rep:drs-bindings-keymap-mixin (drscheme:unit:program-editor-mixin (scheme:text-mixin text:info%)))

drscheme:unit:definitions-text% = (drscheme:rep:drs-bindings-keymap-mixin (drscheme:unit:program-editor-
(scheme:text-mixin text:info%)))

Extends: drscheme:unit:definitions-text<%>

- **(instantiate** drscheme:unit:definitions-text% () [**(line-spacing** _)] [**(tab-stops** _)] [**(auto-wrap** _)]) ⇒ drscheme:unit:definitions-text% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

set-filename

Set the path name for the file to be saved from or reloaded into this editor. This method is also called when the filename changes through any method (such as load-file).

The filename of an editor can be changed by the system in response to file loading and saving method calls, and such changes do not go through this method; use on-load-file and on-save-file to monitor such filename changes.

- **(send** *a-drscheme:unit:definitions-text* set-filename *filename temporary?*) ⇒ void
  filename : string or #f
  temporary? = #f : boolean

  Calls update-save-message.

set-modified

Sets the modified state of the editor. Usually, the state is changed automatically after an insertion, deletion, or style change by calling this method. (This method is also called when the modification state changes through *any* method.) This method is usually not called when the state of the flag is not changing.

See also is-modified? and on-snip-modified.

- **(send** *a-drscheme:unit:definitions-text* set-modified *modified?*) ⇒ void
  modified? : boolean

  Calls update-save-button.

3.27 drscheme:unit:frame<%>

`clear-annotations`

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 override this method to clear annotations.

DrScheme call this method before a program is executed.

- **(send** *a-drscheme:unit:frame* `clear-annotations`**)** ⇒ void

  Does nothing.

`get-definitions-canvas`

- **(send** *a-drscheme:unit:frame* `get-definitions-canvas`**)** ⇒ (instanceof (derivedfrom drscheme:unit:definitions-c

  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.

`get-definitions-text`

- **(send** *a-drscheme:unit:frame* `get-definitions-text`**)** ⇒ (instaceof (derivedfrom 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.

`get-interactions-canvas`

- **(send** *a-drscheme:unit:frame* `get-interactions-canvas`**)** ⇒ (instanceof (derivedfrom drscheme:unit:interactions

  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.

`get-interactions-text`

- **(send** *a-drscheme:unit:frame* `get-interactions-text`**)** ⇒ (instanceof (derivedfrom 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-t procedure, it will use the extended class to create the text.

`get-special-menu`

Returns the `"Special"` menu.

- **(send** *a-drscheme:unit:frame* `get-special-menu`**)** ⇒ (is-a?/c `menu%`)

3.28 drscheme:unit:frame% = (drscheme:frame:mixin (drscheme:frame:basics-mixin frame:searchable%))

`drscheme:unit:frame% = (drscheme:frame:mixin (drscheme:frame:basics-mixin frame:searchable%))`

Extends: `drscheme:unit:frame<%>`

Extends: `drscheme:rep:context<%>`

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

- **(instantiate** `drscheme:unit:frame%` **() (label _) [(parent _)] [(width _)] [(height _)] [(x _)] [(y _)] [(style _)] [(enabled _)] [(border _)] [(spacing _)] [(alignment _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])** ⇒ `drscheme:unit:frame%` object
  label : string (up to 200 characters)
  parent = #f : `frame%` object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in `'(no-resize-border no-caption no-system-menu mdi-parent mdi-child)`
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  The *label* string is displayed in the frame's title bar. If the frame's label is changed (see `set-label`), the title bar is updated.

  The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

  If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by `current-eventspace` . Otherwise, *parent*'s eventspace is the new frame's eventspace.

  If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

  If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an exn:application:mismatch exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see window<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

add-show-menu-items

This method is called during the construction of the show 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.

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

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

break-callback

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

- (**send** *a-drscheme:unit:frame* break-callback) ⇒ void

  Breaks the user's evaluation started by the Execute button (or possibly a queued callback in the user's eventspace).

change-to-file

- (**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.

`clear-annotations`

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 override this method to clear annotations.

DrScheme call this method before a program is executed.

- **(send** *a-drscheme:unit:frame* `clear-annotations`**)** ⇒ void
  Clears any error highlighting.
- **(send** *a-drscheme:unit:frame* `clear-annotations`**)** ⇒ void

`disable-evaluation`

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.

See also `enable-evaluation`.

- **(send** *a-drscheme:unit:frame* `disable-evaluation`**)** ⇒ void
  Disables the execute button, and the execute menu item and `lock`s the interactions window, and the definitions window.

`disable-evaluation`

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.

See also `enable-evaluation`.

- **(send** *a-drscheme:unit:frame* `disable-evaluation`**)** ⇒ void

`edit-menu:between-select-all-and-find`

This method is called between the addition of the select-all menu-item and before the addition of the find menu-item to the edit-menu menu. Override it to add additional menus at that point.

- **(send** *a-drscheme:unit:frame* `edit-menu:between-select-all-and-find`**)** ⇒ void
  Adds the `"Split"` and `"Collapse"` menu items.

`enable-evaluation`

This method must disable all user-sponsored evaluation. It is called once the user starts some evaluation to ensure that only one evaluation proceeds at a time.

See also `enable-evaluation`.

- **(send** *a-drscheme:unit:frame* enable-evaluation**)** ⇒ void

  Enables the execute button, and the execute menu item and locks the interactions window and the definitions window.

ensure-defs-shown

Ensures that the definitions window is visible.

- **(send** *a-drscheme:unit:frame* ensure-defs-shown**)** ⇒ void

ensure-rep-shown

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

  Shows the interactions window

execute-callback

This method is called when the user clicks on the execute button or chooses the execute menu item.

- **(send** *a-drscheme:unit:frame* execute-callback**)** ⇒ void

  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.

file-menu:between-open-and-revert

This method is called between the addition of the open menu-item and before the addition of the revert menu-item to the file-menu menu. Override it to add additional menus at that point.

- **(send** *a-drscheme:unit:frame* file-menu:between-open-and-revert**)** ⇒ void

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

file-menu:between-print-and-close

This method is called between the addition of the print menu-item and before the addition of the close menu-item to the file-menu menu. Override it to add additional menus at that point.

- **(send** *a-drscheme:unit:frame* file-menu:between-print-and-close**)** ⇒ void

  Adds a menu item for printing the interactions.

file-menu:between-save-as-and-print

This method is called between the addition of the save-as menu-item and before the addition of the print menu-item to the file-menu menu. Override it to add additional menus at that point.

- **(send** *a-drscheme:unit:frame* file-menu:between-save-as-and-print**)** ⇒ void

  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.

`file-menu:print-string`

The result of this method is the name of this menu.

- **(send** *a-drscheme:unit:frame* `file-menu:print-string`**)** ⇒ void
  returns `"Definitions"`

`file-menu:save-as-string`

The result of this method is the name of this menu.

- **(send** *a-drscheme:unit:frame* `file-menu:save-as-string`**)** ⇒ void
  Returns `"Definitions"`.

`file-menu:save-string`

The result of this method is the name of this menu.

- **(send** *a-drscheme:unit:frame* `file-menu:save-string`**)** ⇒ void
  Returns `"Definitions"`.

`get-break-button`

Returns the break button. Mostly used for test suites.

- **(send** *a-drscheme:unit:frame* `get-break-button`**)** ⇒ (instanceof `button%`)

`get-button-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 `drscheme:unit:make-bitmap`.

- **(send** *a-drscheme:unit:frame* `get-button-panel`**)** ⇒ (instanceof `horizontal-panel%`)

`get-canvas`

Returns the canvas used to display the `editor<%>` in this frame.

- (**send** *a-drscheme:unit:frame* get-canvas) ⇒ (instanceof editor-canvas%)
  Returns the result of get-definitions-canvas.

get-canvas%

The result of this method is used to create the canvas for the editor<%> in this frame.

- (**send** *a-drscheme:unit:frame* get-canvas%) ⇒ (instanceof (derived-from canvas%))
  Returns the result of drscheme:get/extend:get-definitions-canvas.

get-definitions/interactions-panel-parent

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 inbetween.

- (**send** *a-drscheme:unit:frame* get-definitions/interactions-panel-parent) ⇒ (instanceof vertical-panel%)
  Returns the result of get-area-container

get-directory

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

- (**send** *a-drscheme:unit:frame* get-directory) ⇒ (union string #f)
  This is the directory that the file is saved in, or the directory DrScheme started up in, if the file has not been saved.

get-editor

Returns the editor in this frame.

- (**send** *a-drscheme:unit:frame* get-editor) ⇒ (instanceof editor<%>)
  Returns the result of get-definitions-text.

get-editor%

The result of this class is used to create the editor<%> in this frame.

Override this method to specify a different editor class.

- (**send** *a-drscheme:unit:frame* get-editor%) ⇒ (instanceof (derived-from editor<%>))
  Returns the result of drscheme:get/extend:get-definitions-text.

get-execute-button

Returns the execute button. Mostly used for test suites.

- (**send** *a-drscheme:unit:frame* get-execute-button) ⇒ (instanceof button%)

`get-text-to-search`

Override this method to specify which text to search.

- **(send** *a-drscheme:unit:frame* `get-text-to-search`**)** ⇒ a `text:searching%` object

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

`make-searchable`

- **(send** *a-drscheme:unit:frame* `make-searchable` *canvas***)** ⇒ void
  canvas : a `drscheme:unit:interactions-canvas%` object

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

`on-close`

Called just before the window is closed (e.g., by the window manager). This method is *not* called by `show` .

See also `can-close?`.

- **(send** *a-drscheme:unit:frame* `on-close`**)** ⇒ void

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

  Calls the super method.

`on-size`

Called when the window is resized. The window's new size (in pixels) is provided to the method. The size values are for the entire window, not just the client area.

- **(send** *a-drscheme:unit:frame* `on-size` *width height***)** ⇒ void
  width : exact integer in [0, 10000]
  height : exact integer in [0, 10000]

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

`reset-offer-kill`

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:unit:frame* `reset-offer-kill`**)** ⇒ void

`set-breakables`

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:unit:frame* `set-breakables` *thread custodian*) ⇒ void
  thread : (union thread #f)
  custodian : (union custodian #f)

`set-breakables`

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:unit:frame* `set-breakables`) ⇒ (values (union thread #f) (union custodian #f))

`still-untouched?`

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

- (**send** *a-drscheme:unit:frame* `still-untouched?`) ⇒ boolean

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

`update-save-button`

- (**send** *a-drscheme:unit:frame* `update-save-button` *modified?*) ⇒ void
  modified? : boolean

  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 visiblity for the save button.

  This method is called by the `set-modified` method.

`update-save-message`

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

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

`update-shown`

This method is intended to be overridden. It's job is to update the `"Show"` 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 visiblity of the definitions and interaction `editor-canvas%`s.

Call this method whenever the state of the show menu might need to change.

- (**send** *a-drscheme:unit:frame* `update-shown`) ⇒ void

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

3.29 drscheme:unit:interactions-canvas%

- **(instantiate drscheme:unit:interactions-canvas% () (parent _) [(editor _)] [(style _)] [(scrolls-per-page _)] [(label _)] [(wheel-step _)] [(line-count _)] [(enabled _)] [(vert-margin _)] [(horiz-margin _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])** ⇒ drscheme:unit:interactions-canvas% object
  parent : frame%, dialog%, panel%, or pane% object
  editor = #f : text% or pasteboard% object or #f
  style = null : list of symbols in '(no-hscroll no-vscroll hide-hscroll hide-vscroll)
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If a canvas is initialized with #f for *editor*, install an editor later with set-editor.

  The *style* list can contain the following flags:

  - 'no-hscroll — disallows horizontal scrolling
  - 'no-vscroll — disallows vertical scrolling
  - 'hide-hscroll — allows horizontal scrolling, but hides the horizontal scrollbar
  - 'hide-vscroll — allows vertical scrolling, but hides the vertical scrollbar

  While vertical scrolling of text editors is based on lines, horizontal scrolling and pasteboard vertical scrolling is based on a fixed number of steps per horizontal page. The *scrolls-per-page* argument sets this value.

  If provided, the *wheel-step* argument is passed on to the wheel-step method. The default wheel step can be overridden globally though the '|MrEd:wheelStep| preference; see "Preferences" (section 12, page 325).

  If *line-count* is not #f, it is passed on to the set-line-count method.

  For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

on-focus

Called when a window receives or loses the keyboard focus. If the argument is #t, the keyboard focus was received, otherwise it was lost.

Note that under X, keyboard focus can move to the menu bar when the user is selecting a menu item.

- **(send** *a-drscheme:unit:interactions-canvas* on-focus *on?*) ⇒ void
  on? : boolean

  When the focus is on, calls make-searchable with this.

3.30 drscheme:unit:program-editor-mixin

Domain: editor:basic<%>

Domain: (class->interface text%)

Implements: 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.

- **(instantiate** drscheme:unit:program-editor-mixin% () [(**line-spacing** _)] [(**tab-stops** _)] [(**auto-wrap** _)]) ⇒ drscheme:unit:program-editor-mixin% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

after-delete

Called after a given range is deleted from the editor (and after the display is refreshed; use on-delete and begin-edit-sequence to avoid extra refreshes when after-delete modifies the editor).

See also can-delete? and on-edit-sequence.

No internals locks are set when this method is called.

- **(send** *a-drscheme:unit:program-editor-mixin* after-delete *start len*) ⇒ void
  start : number
  len : number

  Calls the super method.

  Resets an error higlighting.

after-insert

Called after items are inserted into the editor (and after the display is refreshed; use on-insert and begin-edit-sequence to avoid extra refreshes when after-insert modifies the editor).

See also can-insert? and on-edit-sequence.

No internals locks are set when this method is called.

- (**send** *a-drscheme:unit:program-editor-mixin* `after-insert` *start len*) ⇒ void
  start : number
  len : number

  Calls the super method.

  Resets an error higlighting.

3.31 DrScheme Tools Functions

`drscheme:debug:add-prefs-panel`

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

  Adds the profiling preferences panel.

`drscheme:debug:get-cm-key`

- (`drscheme:debug:get-cm-key`) ⇒ any

  Returns a key used with *contination-mark-set->list*. The contination mark set attached to an exception record for the user's program may use this mark. If it does, each mark on the continuation is the same type as the input to `drscheme:debug:open-and-highlight-in-file`.

`drscheme:debug:hide-backtrace-window`

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

  Hides the backtrace window.

`drscheme:debug:make-debug-error-display-handler`

- (`drscheme:debug:make-debug-error-display-handler` *oedh*) ⇒ (string? (union any? exn?) . -> . any)
  oedh : (string? (union any? exn?) . -> . any)

  This function implements an error-display-handler in terms of another error-display-handler.

  This function is designed to work in conjunction with `drscheme:debug:make-debug-eval-handler`.

  See also MzScheme's MzLinkmz:p:error-display-handlererror-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.

`drscheme:debug:make-debug-eval-handler`

- (`drscheme:debug:make-debug-eval-handler` *odeh*) ⇒ (any? . -> . any?)
  odeh : (any? . -> . any?)

  This function implements an eval-handler in terms of another eval-handler.

  This function is designed to work in conjunction with `drscheme:debug:make-debug-error-display-handler`.

  See also MzScheme's MzLinkmz:p:eval-handlereval-handler parameter.

  The resulting eval-handler expands and annotates the input expression and then passes it to the input eval-handler, unless the input expression is already compiled, in which case it just hands it directly to the input eval-handler.

`drscheme:debug:open-and-highlight-in-file`

- (`drscheme:debug:open-and-highlight-in-file` *debug-info*) ⇒ void?
  debug-info : (cons/p (union symbol? (is-a?/c editor<%>)) (cons/p number? number?))

  This function opens a DrScheme to display *debug-info*. The first element in the cons indicates where the file is and the two number indicate a range of text to show.

  See also `drscheme:debug:get-cm-key`.

`drscheme:debug:profiling-enabled`

- (`drscheme:debug:profiling-enabled` *enabled?*) ⇒ void?
  enabled? : boolean?

- (`drscheme:debug:profiling-enabled`) ⇒ boolean?

  A parameter that controls if profiling information is recorded.

  Defaults to `\#f`.

  Only applies if `drscheme:debug:make-debug-eval-handler` has been added to the eval handler.

`drscheme:debug:show-backtrace-window`

- (`drscheme:debug:show-backtrace-window` *error-message* *dis*) ⇒ void?
  error-message : string?
  dis : (listof (cons/p (union symbol? (is-a?/c editor<%>)) (cons/p number? number?)))

  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, and *dis* is the debug information, extracted from the continuation mark in the exception record, using `drscheme:debug:get-cm-key`.

`drscheme:eval:build-user-eventspace/custodian`

- (`drscheme:eval:build-user-eventspace/custodian` *language-settings* *init* *kill-termination*) ⇒ (values 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` 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.)

`drscheme:eval:expand-program`

- (`drscheme:eval:expand-program` *input language-settings eval-compile-time-part? init kill-termination iter*) ⇒ void?
  input : (union port? drscheme:language:text/pos?)
  language-settings : drscheme:language-configuration:language-settings?
  eval-compile-time-part? : boolean?
  init : (-> void?)
  kill-termination : (-> void?)
  iter : ((union eof-object? syntax? (cons/p string? any?)) (-> 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 awscmexpand, §12.6.1 in *PLT MzScheme: Language Manual* 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`, §14.1 in *PLT MzScheme: Language Manual* then pass #f. Otherwise, if your tool just processes the expanded program, be sure to pass #t.

  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:expand-program/multiple`

- (`drscheme:eval:expand-program/multiple` *language-settings eval-compile-time-part? init kill-termination*) ⇒ ((union port? drscheme:language:text/pos?) ((union eof-object? syntax? (cons/p string? any?)) (-> any) . -> . any) . -> . 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.

`drscheme:eval:get-snip-classes`

- `(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:set-basic-parameters`

- `(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/cust`

  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`
    * `'(lib "mred.ss" "mred")`
  - `read-curly-brace-as-paren` is `#t`,
  - `read-square-bracket-as-paren` is `#t`,
  - `break-enabled` is `#t`
  - `error-print-width` is set to 250.
  - The current-load parameter is set to a procedure that calls the language's `front-end` method, instead of just using `read`.
  - `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:frame:calc-button-min-sizes`

- `(drscheme:frame:calc-button-min-sizes` *dc label*`)` ⇒ (values number? number?)
  dc : (is-a?/c `dc<%>`)
  label : string?

  Calculates the minimum width and height of a button label (when drawn with `drscheme:frame:draw-button-label`).

`drscheme:frame:draw-button-label`

- `(drscheme:frame:draw-button-label` *dc label width height inverted*`)` ⇒ void?
  dc : (is-a?/c `dc<%>`)
  label : (union false? string?)
  width : (>/c 5)
  height : (>/c 5)
  inverted : boolean?

  Draws a button label like the one for the (define ...) and filename buttons in the top-left corner of the DrScheme frame. Use this function to draw similar buttons. The basic idea is to create a `canvas%` object whose `on-paint` method is overridden to call this function. The *dc* should be canvas's `dc<%>` object, the *label* should be the string to display on the button. The *width* and *height* arguments should be the width and height of the button and *inverted?* should be `#t` when the button is being depressed.

  See `drscheme:frame:calc-button-min-sizes` for help calculating the min sizes of the button.

`drscheme:get/extend:extend-definitions-canvas`

- (`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:extend-definitions-text`

- (`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:extend-interactions-canvas`

- (`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:extend-interactions-text`

- (`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:extend-unit-frame`

- (`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-definitions-canvas`

- `(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:get-definitions-text`

- `(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:get-interactions-canvas`

- `(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:get-interactions-text`

- `(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:get-unit-frame`

- `(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.

`drscheme:help-desk:help-desk`

- `(drscheme:help-desk:help-desk)` ⇒ void?

- `(drscheme:help-desk:help-desk` *key* *lucky?* *type* *mode*`)` ⇒ void?
  key : string?
  lucky? : boolean?
  type : (symbols 'keyword 'keyword+index 'all)
  mode : (symbols 'exact 'contains 'regexp)

- (drscheme:help-desk:help-desk *key lucky? type*) ⇒ void?
  key : string?
  lucky? : boolean?
  type : (symbols 'keyword 'keyword+index 'all)

- (drscheme:help-desk:help-desk *key lucky?*) ⇒ void?
  key : string?
  lucky? : boolean?

  This function opens a help desk window, or brings an already open help desk window to the front. If an argument is specified, that key is searched for.

  If no arguments are supplied, this function opens a help-desk window to the starting page, or just brings a help-desk window to the front (without changing what page it is viewing).

  If any arguments are supplied, this function opens a help-desk window and searches for *key*, according to *lucky?*, *type*, and *mode*. If the third and fourth arguments are omitted, they default to `'keyword+index` and `'exact`, respectively.

drscheme:help-desk:open-url

- (drscheme:help-desk:open-url *url*) ⇒ void?
  url : string?

  Opens *url* in a new help desk window.

drscheme:language-configuration:add-language

- (drscheme:language-configuration:add-language *language*) ⇒ void?
  language : (is-a?/c drscheme:language:language<%>)

  This function can only be called in phase 2 (see section 2 for details).

  Adds *language* to the languages offerend by DrScheme.

drscheme:language-configuration:fill-language-dialog

- (drscheme:language-configuration:fill-language-dialog *panel button-panel language-setting*) ⇒ 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?

  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.

drscheme:language-configuration:get-settings-preferences-symbol

- (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:language-dialog`

- (drscheme:language-configuration:language-dialog *show-welcome?* *language-settings-to-show* *parent*) ⇒ drscheme:language-configuration:language-settings?
  show-welcome? : boolean?
  language-settings-to-show : drscheme:language-configuration:language-settings?
  parent = #t : (union false? (is-a?/c top-level-window<%>))

  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.

`drscheme:language-configuration:language-settings-language`

- (drscheme:language-configuration:language-settings-language *ls*) ⇒ (is-a?/c drscheme:language:language<
  ls : drscheme:language-configuration:language-settings?

  Extracts the language field of a language-settings.

`drscheme:language-configuration:language-settings-settings`

- (drscheme:language-configuration:language-settings-settings *ls*) ⇒ any?
  ls : drscheme:language-configuration:language-settings?

  Extracts the settings field of a language-settings.

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

- (drscheme:language-configuration:language-settings? *val*) ⇒ boolean?
  val : any?

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

`drscheme:language-configuration:make-language-settings`

- (drscheme:language-configuration:make-language-settings *language* *settings*) ⇒ drscheme:language-configuration:language-settings?
  language : (implementation?/c drscheme:language:language<%>)
  settings : any?

  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-configur

`drscheme:language:create-executable-gui`

- (`drscheme:language:create-executable-gui` *parent program-name show-type? show-base?*) ⇒ (union false? (list/p (symbols (quote no-show) (quote launcher) (quote stand-alone)) (symbols (quote no-show) (quote mred) (quote mzscheme)) string?))
  parent : (union false? (is-a?/c top-level-window<%>))
  program-name : (union false? string?)
  show-type? : boolean?
  show-base? : boolean?

  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, or launcher. If *show-base?* is `\#t`, the user is prompted about a choice of base binary: mzscheme or mred.

  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 `\#f`, the corresponding result will be `'no-show`, otherwise it will indicate the user's choice.

`drscheme:language:create-module-based-launcher`

- (`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 : string?
  executable-filename : string?
  module-language-spec : any?
  transformer-module-language-spec : any?
  init-code : any?
  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:create-module-based-stand-alone-executable`

- (`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 : string?
  executable-filename : string?
  module-language-spec : any?
  transformer-module-language-spec : any?
  init-code : any?
  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.

  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:extend-language-interface`

- (`drscheme:language:extend-language-interface` *interface default-implementation*) ⇒ void?
  interface : interface?
  default-implementation : ((implementation?/c drscheme:language:language<%>) . ->d . (lambda (%) (subclass?/c %)))

  This function can only be called in phase 1 (see section 2 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`

- (`drscheme:language:get-default-mixin`) ⇒ ((implementation?/c drscheme:language:language<%>) . ->d . (lambda (%) (subclass?/c %)))

  This function can only be called in phase 2 (see section 2 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`

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

  This function can only be called in phase 2 (see section 2 for details).

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

`drscheme:language:get-post-hash-bang-start`

- (`drscheme:language:get-post-hash-bang-start` *text*) ⇒ (>=/c 0)
  text : (is-a?/c `text%`)

  Returns the starting position of this text, skipping over #! if there is one. If there is no #!, returns 0.

`drscheme:language:make-simple-settings`

- (`drscheme:language:make-simple-settings` *case-sensitive printing-style fraction-style show-sharing insert-newlines debugging*) ⇒ drscheme:language:simple-settings?
  case-sensitive : boolean?
  printing-style : (symbols 'constructor 'quasiquote 'write 'current-print)
  fraction-style : (symbols 'mixed-fraction 'mixed-fraction-e 'repeating-decimal 'repeating-decimal-e)
  show-sharing : boolean?

insert-newlines : boolean?
debugging : (symbols 'none 'debug 'debug/profile)

Constructs a simple settings.

`drscheme:language:make-text/pos`

- (`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:put-executable`

- (`drscheme:language:put-executable` *parent program-filename mred? launcher?*) ⇒ (union false? string?)
  parent : (is-a?/c `top-level-window<%>`)
  program-filename : string?
  mred? : boolean?
  launcher? : boolean?

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

  The arguments *mred?* and *launcher?* indicate what type of executable this should be (and the dialog may be slightly different on some platforms, depending on these arguments).

`drscheme:language:simple-settings->vector`

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

  Constructs a vector whose first index is the symbol `'struct:simple-settings` and the other elements are the fields of *simple-settings*.

`drscheme:language:simple-settings-annotations`

- (`drscheme:language:simple-settings-annotations` *simple-settings*) ⇒ (symbols 'none 'debug 'debug/profile)
  simple-settings : drscheme:language:simple-settings?

  Extracts the debugging setting from a simple-settings.

`drscheme:language:simple-settings-case-sensitive`

- (`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-fraction-style`

- (`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-insert-newlines`

- (`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-printing-style`

- (`drscheme:language:simple-settings-printing-style` *simple-settings*) ⇒ (symbols 'constructor 'quasiquote 'write 'current-print)
simple-settings : drscheme:language:simple-settings?

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

`drscheme:language:simple-settings-show-sharing`

- (`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?`

- (`drscheme:language:simple-settings?` *val*) ⇒ boolean?
val : any?

Determines if *val* is a simple-settings.

`drscheme:language:text/pos-end`

- (`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-start`

- (`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-text`

- (`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?`

- `(drscheme:language:text/pos?` *val*`)` ⇒ boolean?
  val : any?

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

`drscheme:number-snip:make-fraction-snip`

- `(drscheme:number-snip:make-fraction-snip` *num* *show-prefix-in-decimal-view?*`)` ⇒ (is-a?/c snip%)
  num : number?
  show-prefix-in-decimal-view? : boolean?

  Makes a number snip for DrScheme's REPL that is in the fraction view state. The boolean indicates if a `#e` prefix appears on the number in the decimal state

  See also `drscheme:number-snip:make-repeating-decimal-snip`.

`drscheme:number-snip:make-repeating-decimal-snip`

- `(drscheme:number-snip:make-repeating-decimal-snip` *num* *show-prefix?*`)` ⇒ (is-a?/c snip%)
  num : number?
  show-prefix? : boolean?

  Makes a number snip for DrScheme's REPL that is in the decimal view state. The boolean indicates if a `#e` prefix appears on the number.

  See also `drscheme:number-snip:make-fraction-snip`.

`drscheme:rep:current-rep`

- `(drscheme:rep:current-rep)` ⇒ (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

`drscheme:rep:exn:locs-locs`

- `(drscheme:rep:exn:locs-locs` *loc*`)` ⇒ (listof (list/p (is-a?/c text:basic<%>) number? number?))
  loc : drscheme:rep:exn:locs?

  Extracts the loc field from the exn.

`drscheme:rep:exn:locs?`

- `(drscheme:rep:exn:locs?` *val*`)` ⇒ boolean?
  val : any?

  Determines if *val* is an exn:loc or not.

`drscheme:rep:get-drs-bindings-keymap`

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

  Returns a keymap that bindings 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:get-error-ranges`

- `(drscheme:rep:get-error-ranges)` ⇒ (union false? (cons/p (list/p any? number? number?) (listof (list/p any? number? number?))))

  Returns the currently highlighted error range, or #f if there is none.

`drscheme:rep:insert-error-in-text`

- `(drscheme:rep:insert-error-in-text` *text rep-text msg exn dir*`)` ⇒ void?
  text : (is-a?/c text%)
  rep-text : (is-a?/c drscheme:rep:text<%>)
  msg : string?
  exn : exn?
  dir : (union false? (and/f string? directory-exists?))

  Formats and inserts the error message described by *msg* and *exn* into the textThe *rep-text* argument is used to trigger the actual highlighting. The *msg* and *exn* arguments are expected to come from the `error-display-handler`, when the *error-print-source-location* parameter is set to #f.

  The *user-dir* argument is the current directory of the program where the error occurred. If it is a string, it is used to shorten the path the file where the error occurred.

`drscheme:rep:make-exn:locs`

- `(drscheme:rep:make-exn:locs` *message continuation-mark-set locs*`)` ⇒ drscheme:rep:exn:locs?
  message : string?
  continuation-mark-set : continuation-mark-set?
  locs : (listof (list/p (is-a?/c text:basic<%>) number? number?))

  Constructs an exn:loc. These exceptions are handled specially by DrScheme's REPL. The source locations inside them are highlighted by the default exception handler.

`drscheme:rep:reset-error-ranges`

- `(drscheme:rep:reset-error-ranges)` ⇒ void?

  Clears the current error highlighting.

`drscheme:rep:use-number-snip`

- `(drscheme:rep:use-number-snip)` ⇒ (any? . -> . boolean?)
- `(drscheme:rep:use-number-snip` *use-number-snip?*`)` ⇒ void?
  use-number-snip? : (any? . -> . boolean?)

  This is a parameter whose value is a predicate determines if DrScheme uses a mixed fraction snip, a repeating decimal snip, or a regular ASCII improper fraction for printing numbers.

  If the value of the parameter returns `#t`, a mixed improper fraction snip is used. If it returns `'repeating-decimal`, a repeating decimal snip is used. If it returns `#f`, an ASCII improper fraction is used.

  Its default value is:

```
(lambda (x)
  (if (and (number? x)
           (exact? x)
           (real? x)
           (not (integer? x)))
      #t
      #f))
```

The value of this parameter must not return `#t` more often than the above code, or else the snip implementation will fail. It may, however, return `#f` more often.

`drscheme:rep:which-number-snip`

- (`drscheme:rep:which-number-snip` *which-number-snip*) ⇒ void?
  which-number-snip : (number? . -> . (symbols 'mixed-fraction 'mixed-fraction-e 'repeating-decimal 'repeating-decimal-e))
- (`drscheme:rep:which-number-snip`) ⇒ (number? . -> . (symbols 'mixed-fraction 'mixed-fraction-e 'repeating-decimal 'repeating-decimal-e))

  This function is called if `drscheme:rep:use-number-snip` returns `#t` for some kind of snip. When that happens, this parameter determines what kind of snip to use.

  The symbol `'mixed-fraction` indicates a mixed fraction snip. The symbol `'repeating-decimal` indicates a decimal expansion, possibly with an overbar on a suffix of the decimal expansion indicating that suffix is repeated forever. Either symbol suffixed with `-e` is the same, except that an `#e` is prefixed to the number when viewed in decimal notation.

`drscheme:teachpack:install-teachpacks`

- (`drscheme:teachpack:install-teachpacks` *teachpack-cache*) ⇒ void?
  teachpack-cache : drscheme:teachpack:teachpack-cache?

  Installs the teachpack cache in the current namespace. Passing 'drscheme:teachpacks to `preferences:get` returns the user's currently selected TeachPacks.

`drscheme:teachpack:teachpack-cache-filenames`

- (`drscheme:teachpack:teachpack-cache-filenames` *teachpack-cache*) ⇒ (listof string?)
  teachpack-cache : drscheme:teachpack:teachpack-cache?

  Returns the list of filenames for the teachpacks in *teachpack-cache*.

  See also `drscheme:teachpack:install-teachpacks`.

`drscheme:teachpack:teachpack-cache?`

- (`drscheme:teachpack:teachpack-cache?` *val*) ⇒ boolean?
  val : any?

  Determines if *val* is a teachpack cache or not.

`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 section 2 for details)..

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

drscheme:unit:get-program-editor-mixin

- (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:make-bitmap

- (drscheme:unit:make-bitmap *button-name*) ⇒ ((is-a?/c area-container<%>) . -> . (is-a?/c bitmap%))
  button-name : string?
- (drscheme:unit:make-bitmap *text* *filename*) ⇒ ((is-a?/c area-container<%>) . -> . (is-a?/c bitmap%))
  text : string?
  filename : string?

  This function constructs a bitmap for a button label. It is used for the buttons on the top row of DrScheme's frame.

  When one argument is supplied, this function constructs a button from the image in the **icons** collection named by the *button-name* with .bmp added to the end of the name. The button's label is also *button-name*, but with the first letter capitalized.

  When two arguments are supplied, constructs a button with *text* as the button's label and where *filename* specifies the full path to the bitmap

  The area-container<%> argument is used to find the font for the label

drscheme:unit:open-drscheme-window

- (drscheme:unit:open-drscheme-window) ⇒ (is-a?/c drscheme:unit:frame%)
- (drscheme:unit:open-drscheme-window *filename*) ⇒ (is-a?/c drscheme:unit:frame%)
  filename : (union string? false?)

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

Index