

PLT DrScheme: Programming Environment Manual

PLT
scheme@cs.rice.edu

Version 100
August 1999

Copyright notice

Copyright ©1996-99 PLT, Rice University

Permission to make digital/hard copies and/or distribute this documentation for any purpose is hereby granted without fee, provided that the above copyright notice, author, and this permission notice appear in all copies of this documentation.

Send us your Web links

If you use any parts or all of the DrScheme 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@cs.rice.edu*. Evidence of interest helps the DrScheme Project to maintain the necessary intellectual and financial support. We appreciate your help.

Contents

| | | |
|----------|--|-----------|
| 1 | About DrScheme | 1 |
| 2 | Using DrScheme | 2 |
| 2.1 | Interface Essentials | 2 |
| 2.1.1 | Buttons | 2 |
| 2.1.2 | The Editor | 3 |
| 2.1.3 | The Interactions Window | 3 |
| 2.1.4 | Errors | 4 |
| 2.1.5 | Languages | 4 |
| 2.1.6 | Printed Results | 7 |
| 2.1.7 | Input and Output | 8 |
| 2.2 | Interface Reference | 9 |
| 2.2.1 | Menus | 9 |
| 2.2.2 | Preferences | 11 |
| 2.2.3 | Keyboard Shortcuts | 12 |
| 2.2.4 | DrScheme Files | 14 |
| 3 | Extending DrScheme | 16 |
| 3.1 | Libraries | 16 |
| 3.2 | Tools | 16 |
| 4 | Using DrScheme Jr | 17 |
| 5 | Frequently Asked Questions | 18 |
| 5.1 | Supported Operating Systems and Installation | 18 |
| 5.2 | Using DrScheme | 19 |
| 5.3 | Memory and Performance | 21 |

| | |
|-------------------------------|-----------|
| 5.4 Troubleshooting | 22 |
| Index | 23 |

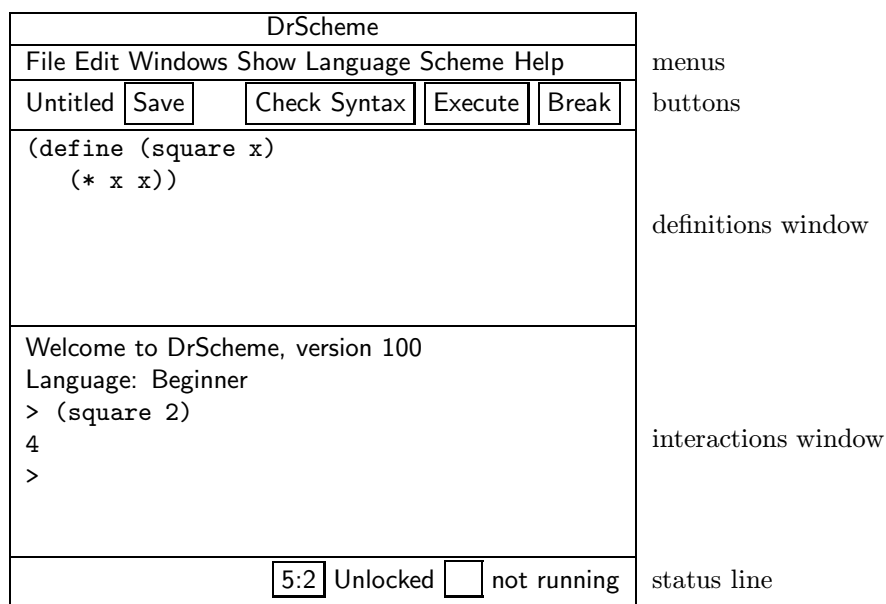
1. About DrScheme

DrScheme is a graphical environment for developing programs using the Scheme programming language. DrScheme runs under Windows 95/98/NT, MacOS, and Unix/X.

2. Using DrScheme

2.1 Interface Essentials

The DrScheme window has three parts: a row of buttons at the top, two editing panels in the middle, and a status line at the bottom.



The top editing panel, called the **definitions window**, is for defining Scheme programs. The above figure shows a program that defines the function `square`.

The bottom panel, called the **interactions window**, is for evaluating Scheme expressions interactively. The **Language** line in the interactions window indicates which primitives are available in the definitions and interactions windows. In the above figure, the language is **Beginner**, which is the default language.

Clicking the **Execute** button evaluates the program in the definitions window, making the program's definitions available in the interactions window. Given the definition of `square` as in the figure above, typing `(square 2)` in the interactions window produces the result 4.

The status line at the bottom of DrScheme's window provides information about the current line and position of the editing caret, whether the current file can be modified, and whether DrScheme is currently evaluating any expression. The recycling icon flashes while DrScheme is "recycling" internal resources, such as memory.

2.1.1 Buttons

The **Save** button appears whenever the definitions window is modified. Clicking the button saves the contents of the definitions window to a file. The current name of the file appears to the left of the **Save** button, but

a file-selection dialog appears if the file has never been saved before.

The **Step** button starts The Foot, which shows the evaluation of a program as a series of small steps. Each evaluation step replaces an expression in the program with an equivalent one using the evaluation rules of DrScheme. For example, a step might replace `(+ 1 2)` with `3`. These are the same rules used by DrScheme to evaluate a program. Clicking **Step** opens a new window that contains the program from the definitions window, plus three new buttons: **Next**, **Previous**, and **Home**. Clicking **Next** performs a single evaluation step, clicking **Previous** retraces a single step, and clicking **Home** returns to the initial program. The Foot works only for programs using the Beginner language level.

The **Check Syntax** button checks the syntax of the program in the definitions window and colorizes keywords and primitive function names. It also enables pop-up arrows that show the relationship between variable declarations and their uses when the mouse cursor is moved over a variable. Modifying the program turns off the arrows until **Check Syntax** is clicked again.

The **Execute** button evaluates the program in the definitions window and resets the interactions window.

The **Break** button interrupts an evaluation, or beeps if DrScheme is not evaluating anything. For example, after clicking **Execute** or entering an expression into the interactions window, click **Break** to cancel the evaluation. Click the **Break** button once to try to interrupt the evaluation gracefully; click the button twice to killing the evaluation immediately.

An **Analyze** button appears if you have installed the MrSpidey static debugger. Clicking the button starts the debugger on the program. See *PLT MrSpidey: Static Debugger Manual* for more information.

2.1.2 The Editor

DrScheme's editor provides special support for managing parentheses in a program. When the blinking caret is next to a parenthesis, DrScheme shades the region between the parenthesis and its matching parenthesis. This feature is especially helpful when for balancing parentheses to complete an expression. Furthermore, if you type a closing parenthesis `)` that should match an opening square bracket `[`, the editor automatically converts the `)` into a `]`. DrScheme beeps whenever a closing parenthesis does not match an opening parenthesis.

DrScheme also flashes quotation mark matches, just like parentheses. Beware, however, that DrScheme cannot distinguish between closing an opening quotation marks. Thus, when you type an opening quotation mark, DrScheme may flash a match to the previous closing quotation mark.

Although whitespace is not significant in Scheme, DrScheme encourages a particular format for Scheme code. When you type Enter or Return, the editor inserts a new line and automatically indents it. To make DrScheme re-indent an existing line, move the flashing caret to the line and hit the Tab key. (The caret can be anywhere in the line.) You can re-indent an entire region by selecting the region and typing Tab.

2.1.3 The Interactions Window

The interactions window lets you type an expression after the `>` prompt for immediate evaluation. You cannot modify any text before the last `>` prompt. To enter an expression, the flashing caret must appear after the last prompt, and also after the space following the prompt.

When you type a complete expression and hit Enter or Return, DrScheme evaluates the expression and prints the result. After printing the result, DrScheme creates a new prompt for another expression. Some expressions return a special "void" value; DrScheme never prints void, but instead produces a new prompt immediately.

If the expression following the current prompt is incomplete, then DrScheme will not try to evaluate it. In

that case, hitting Enter or Return produces a new, auto-indented line.

To copy the previous expression to the current prompt, type ESC-p (i.e., type Escape and then type p). Type ESC-p multiple times to cycle back through old expressions. Type ESC-n to cycle forward through old expressions. Also, if you move the flashing caret after an old expression and hit Enter or Return, DrScheme copies the expression to the current prompt.

Clicking the **Execute** button evaluates the program in the definitions window and makes the program's definitions available in the interactions window. Clicking **Execute** also resets the interactions window, erasing all old interactions and removing old definitions from the interaction environment. Although **Execute** erases old > prompts, ESC-p and ESC-n can still retrieve old expressions.

2.1.4 Errors

Whenever DrScheme encounters an error while evaluating an expression, it prints an error message in the interactions window and highlights the expression that triggered the error. The highlighted expression might be in the definitions window, or it might be after an old prompt in the interactions window.

For certain kinds of errors, DrScheme turns a portion of the error message into a hyperlink. Click the hyperlink to get help regarding a function or keyword related to the error.

2.1.5 Languages

DrScheme supports five languages:

- **Beginner** is a small version of Scheme that is tailored for beginning computer science students.
- **Intermediate** extends Beginner with programmer-defined datatypes and lexically-scoped bindings.
- **Advanced** extends Intermediate with mutable state.
- **MzScheme** is a practical dialect of Scheme that is described in *PLT MzScheme: Language Manual*.
- **MrEd** extends MzScheme with a GUI toolbox that is described in *PLT MrEd: Graphical Toolbox Manual*.

Programs written in standard Scheme generally require the Advanced, MzScheme, or MrEd language. The MzScheme Debug and MrEd Debug languages are the same as the MzScheme and MrEd languages, respectively, but the Debug languages provide source location information for syntax and run-time errors (with a small performance cost).

DrScheme always shows the current evaluation language in the top of the interactions window. To choose a different language, select the **Language|Configure Language...** menu item. After changing the language, click **Execute** to reset the language in the interactions window.

The table in Figure 2.1 summarizes the syntactic forms and procedures built into each language. Most lines in the table specify the syntactic forms and procedures that each language provides. Lines in the next-to-last section specify deviations from the standard Scheme language:

- **Case-sensitive identifiers and symbols** — In a case-sensitive language, the variable names `x` and `X` are distinct, and the symbols `'x` and `'X` are also distinct. In a case-insensitive language, `x` and `X` are equivalent and `'x` and `'X` represent the same value.

| | Beginner | Intermediate | Advanced | MzScheme | MrEd |
|--|----------|--------------|----------|----------------|----------------|
| <code>define</code> ¹ , <code>define-struct</code> | ✓ | ✓ | ✓ | ✓ | ✓ |
| <code>lambda</code> ² | ✓ | ✓ | ✓ | ✓ | ✓ |
| <code>cond</code> , <code>if</code> , <code>and</code> , <code>or</code> | ✓ | ✓ | ✓ | ✓ | ✓ |
| <code>nand</code> , <code>nor</code> | ✓ | ✓ | ✓ | † ¹ | † ¹ |
| <code>quote</code> 'd symbols | ✓ | ✓ | ✓ | ✓ | ✓ |
| MrSpidey annotations | ✓ | ✓ | ✓ | † ² | † ² |
| <code>quote</code> 'd lists, <code>quasiquote</code> , <code>unquote</code> | | ✓ | ✓ | ✓ | ✓ |
| <code>let</code> ¹ , <code>let*</code> ¹ , <code>letrec</code> ¹ | | ✓ | ✓ | ✓ | ✓ |
| <code>local</code> | | ✓ | ✓ | † ¹ | † ¹ |
| <code>set!</code> ¹ , <code>fluid-let</code> | | | ✓ | ✓ | ✓ |
| <code>begin</code> , <code>begin0</code> , <code>implicit begin</code> | | | ✓ | ✓ | ✓ |
| <code>when</code> , <code>unless</code> , <code>if without else</code> | | | ✓ | ✓ | ✓ |
| Named <code>let</code> , <code>delay</code> , <code>do</code> , <code>case</code> | | | ✓ | ✓ | ✓ |
| <code>recur</code> , <code>rec</code> , <code>evcase</code> | | | ✓ | † ¹ | † ¹ |
| Turtles, <code>split</code> , <code>split*</code> , <code>tprompt</code> | | | ✓ | | † ³ |
| <code>require-library</code> | ✓ | ✓ | ✓ | ✓ | ✓ |
| <code>define-macro</code> , <code>begin-elaboration-time</code> | ✓ | ✓ | ✓ | ✓ | ✓ |
| <code>time</code> | | ✓ | ✓ | ✓ | ✓ |
| <code>let/cc</code> , <code>parameterize</code> , <code>with-handlers</code> | | | ✓ | ✓ | ✓ |
| Other MzScheme syntax | | | | ✓ | ✓ |
| Scheme and MzScheme procedures | ✓ | ✓ | ✓ | ✓ | ✓ |
| MzLib core library procedures | ✓ | ✓ | ✓ | † ⁴ | † ⁴ |
| MrEd GUI classes | | | | | ✓ |
| Case-sensitive identifiers and symbols | ✓ | ✓ | ✓ | | |
| Procedures must take at least 1 argument | ✓ | ✓ | | | |
| Identifier required at function-call position | ✓ | ✓ | | | |
| Top-level required at function-call position | ✓ | | | | |
| <code>lambda</code> allowed only in definitions | ✓ | | | | |
| <code>quote</code> works only on symbols | ✓ | | | | |
| Unmatched <code>cond/case</code> is an error | ✓ | ✓ | ✓ | | |
| Conditional values must be <code>#t</code> or <code>#f</code> | ✓ | ✓ | | | |
| <code>=</code> , <code>...</code> , <code>+</code> , <code>*</code> , and <code>/</code> take at least 2 arguments | ✓ | ✓ | ✓ | | |
| <code>and</code> , <code>or</code> , <code>nand</code> , <code>nor</code> require at least 2 exprs | ✓ | ✓ | ✓ | | |
| Improper lists disallowed | ✓ | ✓ | ✓ | | |
| <code>define-struct</code> always omits supertype | ✓ | ✓ | | | |
| Constructor-style output | ✓ | ✓ | ✓ | | |
| <code>write</code> output | | | | ✓ | ✓ |
| Abbreviate <code>cons</code> constructor with <code>list</code> | | ✓ | ✓ | | |
| Show sharing in values | | | ✓ | | |
| Print inexact numbers with <code>#i</code> | ✓ | ✓ | ✓ | | |

¹ including `-values` forms² including `case-lambda`†¹ Use `(require-library "macro.ss")`†² Use `(require-library "spidey.ss")`†³ Use `(require-library "turtle.ss" "graphics")`†⁴ Use `(require-library "core.ss")`

Figure 2.1: Language Definitions

- **Procedures must take at least 1 argument** — In the Beginner and Advanced languages, defined procedures must consume at least one argument. Since the languages have no side-effects, zero-argument functions are not useful, and rejecting such function definitions helps detect confusing syntactic mistakes.
- **Identifier required at function call position** — In the Beginner and Advanced languages, procedure calls must be of the form (*identifier* ...). This restriction helps detect confusing misuses of parentheses, such as (1) or ((+ 3 4)), which is a common mistake among beginners who are used to the optional parentheses of algebra.
- **Top-level required at function call position** — In the Beginner language, procedure calls must be of the form (*top-level-identifier* ...). This restriction helps detect confusing misuses of parentheses, such as (*x*) where *x* is a function argument. DrScheme can detect such mistakes syntactically because Beginner does not support higher-order procedures.
- **lambda allowed only in definitions** — In the Beginner language, `lambda` (or `case-lambda`) may appear only in a definition and only as the value of the defined variable. This restriction is required by the evaluation rules used by the stepper tool.
- **quote works only on symbols** — In the Beginner and Advanced languages, `quote` and `'` can specify only symbols. This restriction avoids the need to explain to beginners why 1 and '1 are equivalent in standard Scheme.
- **Unmatched cond/case is an error** — In the Beginner, Intermediate, and Advanced languages, falling through a `cond` or `case` expression without matching a clause signals a run-time error. This convention helps detect syntactic and logical errors in programs.
- **Conditional values must be #t or #f** — In the Beginner and Advanced languages, an expression whose value is treated as a Boolean must return `#t` or `#f`. This restriction, which applies to `if`, `cond`, `and`, `or`, `nand`, and `nor` expressions, helps detect errors where a Boolean function application is omitted.
- **=, <=, <, >, >=, +, *, and / take at least two arguments** — In the Beginner, Intermediate, and Advanced languages, mathematical operators that are infix in algebra notation require at least two arguments in DrScheme. This restriction helps detect missing arguments to an operator.
- **and, or, nand, and nor require at least 2 expressions** — In the Beginner, Intermediate, and Advanced languages, the Boolean combination forms require at least two sub-expressions. This restriction helps detect missing or ill-formed sub-expressions in a Boolean expression.
- **Improper lists disallowed** — A **proper list** is either an empty list or a list created by `consing` onto a proper list. In the Beginner and Advanced languages, `cons` constructs only **proper lists**, signalling an error if the second argument is not a proper list. Since beginning students do not need improper lists, this restriction helps detect logical errors in recursive functions.
- **define-struct always omits supertype** — In Beginner and Intermediate, the `define-struct` syntactic form does not support the optional supertype expression. This restriction simplifies the syntax to help detect parenthesization mistakes.

Lines in the last section specify deviations from a traditional Scheme `read-eval-print` interface in the way that DrScheme prints interaction results:

- **Constructor-style output** — See Section 2.1.6.1.
- **write output** — Prints interaction results with `write`.
- **Abbreviate cons constructor with list** — Adjusts constructor-style output to print list values with `list` instead of a nested sequence of `conses`.

| Input Expression | Constructor output | write output |
|-------------------|--------------------|-------------------|
| (cons 1 2) | (cons 1 2) | (1 . 2) |
| (list 1 2) | (list 1 2) | (1 2) |
| '(1 2) | (list 1 2) | (1 2) |
| (vector 1 2 3) | (vector 1 2 3) | #(1 2 3) |
| (box 1) | (box 1) | #&1 |
| (lambda (x) x) | (lambda (a) ...) | #<procedure> |
| 'sym | 'sym | sym |
| (make-s 1 2) | (make-s 1 2) | #<structure:s> |
| '() | empty | () |
| #t | true | #t |
| #f | false | #f |
| add1 | add1 | #<primitive:add1> |
| (list (void)) | (list (void)) | (#<void>) |
| (make-weak-box 1) | (make-weak-box 1) | #<weak-box> |
| (delay 1) | (delay ...) | #<promise> |
| (regexp "a") | (regexp ...) | #<regexp> |

Figure 2.2: Comparison of constructor-style output to write

- **Show sharing in values** — Prints interaction results using the **shared** syntax, which exposes shared structure within a value. For example, the list created by (let ([l (list 0)]) (list 1 l)) prints as

```
(shared ((-1- (list 0))) (list -1- -1-))
```

rather than

```
(list (list 0) (list 0)).
```

- **Print inexact numbers with #i** — Prints inexact numbers with a leading **#i** to emphasize that they represent imprecise results (or even effectively incorrect results, depending on the intended calculation).

The **Language|Configure Language...** dialog contains a **Show Details** button that lets you configure certain details of the language specification. (Each option corresponds to one of the lines in the language table, but only a few of the lines in the figure have an option in the dialog.) Whenever the selected options do not match the default language specification, a **Custom** indicator appears next to the language-selection control at the top of the dialog.

2.1.6 Printed Results

2.1.6.1 CONSTRUCTOR-STYLE OUTPUT

DrScheme's **constructor-style output** treats **cons**, **vector**, and similar primitives as value constructors, rather than functions. It also treats **list** as shorthand for multiple **cons**'s ending with the empty list. Constructor-style printing is valuable for beginning computer science students, because output values look the same as input values.

Results printed in DrScheme's interactions window using constructor-style printing look different than results printed in traditional Scheme implementations, which use **write** to print results. The table in Figure 2.2 shows the differences between values printed in constructor style and values printed with **write**.

2.1.6.2 QUASIQUOTE-STYLE OUTPUT

Constructor-style output is inconvenient for printing S-expression results that represent programs. For example, the value `'(lambda (x) (lambda (y) (+ x y)))` prints as

```
(list 'lambda (list 'x) (list 'lambda (list 'y) (list '+ 'x 'y)))
```

with constructor-style printing.

DrScheme's **quasiquote-style output** combines the input–output invariance of constructor-style printing with the S-expression readability of `write`. It uses `quasiquote` to print lists, and uses `unquote` to escape back to constructor style printing for non-lists and non-symbols.

With quasiquote-style printing, the above example prints as:

```
'(lambda (x) (lambda (y) (+ x y)))
```

2.1.7 Input and Output

Many Scheme programs avoid explicit input and output operations, obtaining input via direct function calls in the interactions window, and producing output by returning values. Other Scheme programs explicitly print output for the user during evaluation using `write` or `display`, or explicitly request input from the user using `read` or `read-char`.

Explicit input and output appear in the interactions window, but within special boxes that separate explicit I/O from normal expressions and results. For example, evaluating

```
> (read)
```

in the interactions window produces a special box for entering input:

```
┌ _
```

(The underscore indicates the location of the flashing caret.) Type an number into the box and hit Enter, and that number becomes the result of the `(read)` expression. If you type 5, the overall interaction appears as follows:

```
> (read)
┌ 5
5
> _
```

The mouse cursor becomes a watch whenever DrScheme is evaluating expression, but you can still use the mouse to move the selection in an input box.

Output goes to the same box as input. If you execute the program

```
(define v (read))
(display v)
v
```

and provide the input S-expression `(1 2)`, the interactions window ultimately appears as follows:

```

(1 2)
(1 2)
(cons 1 (cons 2 empty)))
> _

```

In this example, `display` produces output immediately beneath the input you typed, but the final result was printed outside the box because it is the result of the program, rather than explicit output. (The above example assumes constructor-style printing. With traditional value printing, the final line outside the box would be `(1 2)`.)

Entering the same program line-by-line in the interactions window produces a different-looking result:

```

> (define v (read))
(1 2)
> (display v)
(1 2)
> v
(cons 1 (cons 2 empty))
> _

```

Although it is the same program as before, entering the program expression-by-expression demonstrates how each prompt creates its own I/O box.

2.2 Interface Reference

2.2.1 Menus

2.2.1.1 File

- **New** creates a new DrScheme window.
- **Open...** opens a find-file dialog for choosing a file to load into a definitions window.
- **Open URL...** opens a dialog for choosing a Uniform Resource Locator (URL) to open in a new Help Desk window.
- **Revert** re-loads the file that is currently in the definitions window. All changes since the file was last saved will be lost.
- **Save Definitions** saves the program in the definitions window. If the program has never been saved before, a save-file dialog appears.
- **Save Definitions As...** opens a save-file dialog for choosing a destination file to save the program in the definitions window. Subsequent saves write to the newly-selected file.
- **Save Definitions As Text...** is like **Save Definitions As...**, but the file is saved in plain-text format (see Section 2.2.4.1). Subsequent saves also write in plain-text format.
- **Save Interactions** saves the contents of the interactions window to a file. If the interaction contents have never been saved before, a save-file dialog appears.
- **Save Interactions As...** opens a save-file dialog for choosing a destination file to save the contents of the interactions window. Subsequent saves write to the newly-selected file.
- **Save Interactions As Text...** is like **Save Interactions As...**, but the file is saved in plain-text format (see Section 2.2.4.1). Subsequent saves are write in plain-text format.

- **Show Interactions History** opens a window that shows the history of expressions evaluated in the interactions window. Use ESC-p and ESC-n in the interactions window to cycle through the history.
- **Print Definitions...** opens a dialog for printing the current program in the definitions window.
- **Print Interactions...** opens a dialog for printing the contents of the interactions window.
- **Close** closes this DrScheme window. If this window is the only open DrScheme window, DrScheme quits.
- **Quit or Exit** exits DrScheme.

2.2.1.2 Edit

All Edit menu items operate on either the definitions or interactions window, depending on the location of the selection or blinking caret. Each window maintains its own Undo and Redo history.

- **Undo** reverses an editing action. Each window maintains a history of actions, so multiple Undo operations can reverse multiple editing actions.
- **Redo** reverses an Undo action. Each window maintains a history of Undo actions, so multiple Redo operations can reverse multiple Undo actions.
- **Cut** copies the selected text to the clipboard and deletes it from the window.
- **Copy** copies the selected text to the clipboard.
- **Paste** pastes the current clipboard contents into the window.
- **Delete** deletes the selected text.
- **Find** opens a search panel at the bottom of the frame.
- **Replace** opens a search panel at the bottom of the frame.
- **Insert Text Box** inserts a box into the window. The text box may contain arbitrary text, and it is treated as a value, like a number or symbol.
- **Insert Graphic Box** inserts a graphics box. The box is treated as a value.
- **Insert Image...** opens a find-file dialog for selecting an image file in GIF, BMP, XBM, or XPM format. The image is treated as a value.
- **Toggle Wrap Text** toggles between wrapped text and unwrapped text in the window.
- **Preferences** opens the preferences dialog. See section 2.2.2.

2.2.1.3 Windows

This menu contains an entry for each window in DrScheme. Selecting a menu item brings the corresponding window to the front.

2.2.1.4 View

Two of the following menu items appear at a time:

- **Show Interactions** shows interactions window.
- **Hide Interactions** hides interactions window.
- **Show Definitions** shows the definitions window.
- **Hide Definitions** hides the definitions window.

Note: whenever a program is executed, the interactions window is made visible if it is invisible.

2.2.1.5 Language

- **Configure Language** opens a dialog for selecting the current evaluation language. Click **Execute** to make the language active in the interactions window. See section 2.1.5 for more information about the languages.
- **Select Library...** opens a find-file dialog for choosing a library to extend the current language. Click **Execute** to make the library available in the interactions windows. See Section 3 for information on creating libraries.
- **Clear Library** clears the current library. Click **Execute** to clear the library from the interactions window.

2.2.1.6 Scheme

- **Execute** resets the interactions window and executes the program in the definitions window.
- **Break** breaks the current evaluation.
- **Indent** indents the selected text according to the standard Scheme formatting conventions. (Pressing the Tab key has the same effect.)
- **Indent All** indents all of the text in either the definitions or interactions window, depending on the location of the selection or blinking caret.
- **Comment Out** puts “;” characters at the beginning of each selected line of text, except for lines that already start with “;”.
- **Uncomment** removes all “;” characters at the start of each selected line of text. Uncommenting removes a “;” if it appears at the start of a line, if it follows only whitespace on its line, or if it follows another “;” that is removed at the same time.

2.2.2 Preferences

The preferences dialog comprises several panels:

- **Font**
This panel controls the main font used by DrScheme.
- **Indenting**
This panel controls which keywords DrScheme recognizes for indenting, and how each keyword is treated.

- General

- **Highlight between matching parens** — If checked, the editor marks the region between matching parenthesis with a gray background (in color) or a stipple pattern (in monochrome) when the flashing caret is next to a parenthesis.
- **Correct parens** — If checked, the editor automatically converts a typed “)” to “]” to match “[”, or it converts a typed “]” to “)” to match “(“.
- **Flash paren match** — If checked, typing a closing parenthesis, square bracket, or quotation mark flashes the matching open parenthesis/bracket/quote.
- **Auto-save files** — If checked, the editor generates autosave files (see Section 2.2.4.2) for files that have not been saved after five minutes.
- **Map delete to backspace** — If checked, the editor treats the Delete key like the Backspace key.
- **Use platform-specific file dialogs** — If checked, DrScheme uses the standard, platform-specific dialogs for its find-file and save-file dialogs. Otherwise, DrScheme uses dialogs that are the same across platforms.
- **Verify exit** — If checked, DrScheme consults the user before exiting.
- **Ask before changing save format** — If checked, DrScheme consults the user before saving a file in non-text format (see Section 2.2.4.1).
- **Wrap words in editor buffers** — If checked, DrScheme editors auto-wrap text lines by default. Changing this preference affects new windows only.
- **Show status-line** — If checked, DrScheme shows a status line at the bottom of each window.
- **Count line and column numbers from one** — If checked, the status line’s line:column counter counts from one. Otherwise, it counts from zero.
- **Enable keybindings in menus** — If checked, some DrScheme menu items have keybindings. Otherwise, no menu items have key bindings. Changing this preference affects new windows only.
- **Display line numbers in buffer; not character offsets** — If checked, the status line shows a line:column display for the current selection rather than the character offset into the text.
- **Use separate dialog for searching** — If checked, then selecting the Find menu item opens a separate dialog for searching and replacing. Otherwise, selecting Find opens an interactive search-and-replace panel at the bottom of a DrScheme window.

- General II

- **Only warn once when executions and interactions are not synchronized** — If checked, DrScheme warns the user on the first interaction after the definitions window, language, or library is changed without a corresponding click on **Execute**. Otherwise, the warning appears on every interaction.

- Check Syntax

This panel controls the font and color styles used by the **Check Syntax** colorizations.

2.2.3 Keyboard Shortcuts

Most key presses simply insert a character into the editor (“a”, “3”, “(”, etc.). Other keys and key combinations act as keyboard shortcuts that move the blinking caret, delete a line, copy the selection, etc. Keyboard shortcuts are usually triggered by key combinations using the Control, Meta, or Command key.

C-⟨key⟩ = This means press the Control key, hold it down and then press ⟨key⟩ and then release them both. For example: C-e (Control-E) moves the blinking caret to the end of the current line.

M-⟨key⟩ = Same as C-⟨key⟩, except with the Meta key. Depending on your keyboard, Meta may be called “Left”, “Right” or have a diamond symbol, but it’s usually on the bottom row next to the space bar. M-⟨key⟩ can also be performed as a two-character sequence: first, strike and release the Escape key, then strike ⟨key⟩. Under Windows and MacOS, meta is only available through the Escape key.

DEL = The Delete key.

SPACE = The Space bar.

Note: On most keyboards, “<” and “>” are shifted characters. So, to get M->, you actually have to type Meta-Shift->. That is, press and hold down both the Meta and Shift keys, and then strike “>”.

Note: Many of the key bindings can also be done with menu items.

Under Windows, some of these keybindings are actually standard menu items. Those keybindings will behave according to the menus, unless the **Enable Keybindings In Menus** preference is unchecked.

2.2.3.1 MOVING AROUND

- C-f move forward one character
- C-b move backward one character
- M-f move forward one word
- M-b move backward one word
- C-v move forward one page
- M-v move backward one page
- M-< move to beginning of file
- M-> move to end of file
- C-a move to beginning of line (left)
- C-e move to end of line (right)
- C-n move to next line (down)
- C-p move to previous line (up)
- M-C-f move forward one S-expression
- M-C-b move backward one S-expression
- M-C-u move up out of an S-expression
- M-C-d move down into a nested S-expression
- M-C-SPACE select forward S-expression
- M-C-p match parentheses backward

2.2.3.2 EDITING OPERATIONS

- C-d delete forward one character
- C-h delete backward one character
- M-d delete forward one word
- M-DEL delete backward one word
- C-k delete forward to end of line
- M-C-k delete forward one S-expression

- M-w copy selection to clipboard
- C-w delete selection to clipboard (cut)
- C-y paste from clipboard (yank)
- C-t transpose characters
- M-t transpose words
- C-_ undo
- C-+ redo
- C-x u undo

2.2.3.3 FILE OPERATIONS

- C-x C-s save file
- C-x C-w save file under new name

2.2.3.4 SEARCHING

- C-s search for string forward
- C-r search for string backward

2.2.3.5 INTERACTIONS

The interactions window has all of the same keyboard shortcuts as the definitions window plus a few more:

- M-p bring the previously executed expression down to the prompt.
- M-n bring the expression after the current expression in the expression history down to the prompt.

2.2.4 DrScheme Files

2.2.4.1 PROGRAM FILES

The standard extension for a Scheme program file is **.scm**. The extensions **.ss** and **.sch** are also acceptable.

DrScheme's editor can save a program file in two different formats:

- **Plain-text format** — All text editors can read this format. DrScheme saves a program in plain-text format by default, unless the program contains images or text boxes. (Plain-text format does not preserve images or text boxes.)

Plain-text format is platform-specific because different platforms have different newline conventions. However, most tools for moving files across platforms support a “text” transfer mode that adjusts newlines correctly.

- **Multimedia format** — This format is specific to DrScheme, and no other editor recognizes it. DrScheme saves a program in multimedia format by default when the program contains images, text boxes, or formatted text.

Multimedia format is platform-independent. Use a “binary” transfer mode when moving multimedia-format files across platforms. (Using “text” mode may corrupt the file.)

2.2.4.2 BACKUP AND AUTOSAVE FILES

When you modify an existing file in DrScheme and save it, DrScheme copies the old version of the file to a special backup file if no backup file exists. The backup file is saved in the same directory as the original file, and the backup file's name is generated from the original file's name:

- Under X and MacOS, a tilde (~) is added to the end of the file's name.
- Under Windows, the file's extension is replaced with **.bak**.

When a file in an active DrScheme editor is modified but not saved, DrScheme saves the file to a special autosave file after five minutes (in case of a power failure or catastrophic error). If the file is later saved, or if the user exists DrScheme without saving the file, the autosave file is removed. The autosave file is saved in the same directory as the original file, and the autosave file's name is generated from the original file's name:

- Under X and MacOS, a pound sign (#) is added to the start and end of the file's name, then a number is added after the ending pound sign, and then one more pound sign is appended to the name. The number is selected to make the autosave filename unique.
- Under Windows, the file's extension is replaced with a number to make the autosave filename unique.

2.2.4.3 MISCELLANEOUS FILES

On start-up, DrScheme reads configuration information from two files: a normal preferences file and a low-level setup file. The names of these files contain “MrEd” because they are part of a preferences system shared by DrScheme and other programs that execute in MrEd.

The name and location of the preferences file depends on the platform and user:

- Under X, preferences are stored in **.mred.prefs** in the user's home directory.
- Under Windows, if the HOMEDRIVE and HOMEPATH environment variables are defined, preferences are stored in **%HOMEDRIVE%\%HOMEPATH%\mred.pre**, otherwise preferences are stored in **mred.pre** in the directory containing the MrEd executable.

Windows NT: When DrScheme is launched under Windows NT and HOMEDRIVE and HOMEPATH are not set, NT automatically sets the variables to indicate the root directory of the main disk. Therefore, when HOMEDRIVE and HOMEPATH are not set in NT, the preferences file **mred.pre** is saved in the root directory.

- Under MacOS, preferences are stored in **MrEd Preferences** in the system preferences folder.

The low-level setup file configures font settings that are needed early in the MrEd boot process. (The information in the setup file is duplicated in the normal preferences file, which is loaded later.) The name and location of the low-level setup file depends on the platform and user:

- Under X, preferences are stored in **.mred.resources** in the user's home directory.
- Under Windows, if the HOMEDRIVE and HOMEPATH environment variables are defined, preferences are stored in **%HOMEDRIVE%\%HOMEPATH%\mred.ini**, otherwise preferences are stored in **mred.ini** in the directory containing the MrEd executable.
- Under MacOS, preferences are stored in **mred.fnt** in the system preferences folder.

3. Extending DrScheme

DrScheme supports two forms of extension to the programming environment:

- A **library** extends the set of produces that are built into a language in DrScheme. For example, a library might extend the Beginner language with a procedure for playing sounds.

Libraries are particularly useful in a classroom setting, where an instructor can provide a library that is designed for a specific exercise. To use the library, each student must download the library file and select it through the **Language|Set Library To...** menu item.

- A **tool** extends the set of utilities within the DrScheme environment. For example, DrScheme's **Check Syntax** button starts a syntax-checking tool, and the **Analyze** button starts the MrSpidey tool.

3.1 Libraries

A single Scheme source file defines a library (although the file is likely to access other files). The last expression in the file must return a signed unit (see units, §7 in *PLT MzScheme: Language Manual*) that imports the `plt:userspace^` signature. When a user selects the library into DrScheme, DrScheme adds the names in the unit's export signature to the user's namespace.

The `plt:userspace^` signature is defined in the `userspcs.ss` file of the `userspce` collection (see Library Collections and MzLib, §15 in *PLT MzScheme: Language Manual*). A library file is always loaded with the MrEd language, but in case-sensitive mode (so that a library can export case-sensitive names to the teaching languages).

As an example example, the following signed unit defines a library that adds the binding `four` to the user's namespace:

```
(unit/sig (four)
  (import plt:userspace^)
  (define four 4))
```

For more interesting examples, see the **teach** directory of the **lib** directory in the PLT installation.

3.2 Tools

A separate manual describes the mechanism for defining a tool. See *PLT Tools: DrScheme Extension Manual*.

4. Using DrScheme Jr

DrScheme Jr is a lightweight, text-only programming shell that provides the functionality of DrScheme's interactions window. Under Windows, DrScheme Jr runs within an "MS Dos" shell. Under MacOS, DrScheme Jr uses a text shell that is embedded into MzScheme. Under Unix, DrScheme Jr runs within a terminal window.

DrScheme Jr's `restart` function serves the same purpose as DrScheme's `Execute` button. Evaluating `(restart)` resets the current interaction and clears all definitions. Evaluating `(restart "file.scm")` resets the current interaction and evaluates the program in **file.scm**.

Set DrScheme Jr's language through a command-line flag when starting DrScheme Jr. For example, to set the language to Beginner:

- Under Windows, run `"DrScheme Jr" -l Beginner --save` in a command shell in the `plt` directory.
- Under MacOS, double-click on DrScheme Jr while holding down the Command key. A dialog appears with a text box; type `-l Beginner --save` into the text box and click OK.
- Under Unix, run `plt/bin/drscheme-jr -l Beginner --save` in a terminal.

The `--save` flag (note that there are two dashes) saves the language selection to a configuration file so that DrScheme Jr starts with the same language next time. (Run DrScheme Jr with the `--help` flag to determine the name of the configuration file.)

5. Frequently Asked Questions

5.1 Supported Operating Systems and Installation

Where can I get DrScheme and/or documentation?

Our web page provides documentation in PostScript, DVI, and HTML format:

<http://www.cs.rice.edu/CS/PLT/packages/drscheme/>

How much does DrScheme cost?

DrScheme is absolutely free for anyone to use. However, there are restrictions on the way that DrScheme can be modified and redistributed. Please read the GNU Library General Public License in the distribution for details.

What operating systems are supported for DrScheme?

Windows 95/NT, MacOS, and Unix with the X Window System.

How much memory is needed to run DrScheme?

To run DrScheme comfortably, your machine should have at least 20 MB of RAM and at least 32 MB of total memory.

I don't have that much memory. Are there any other PLT options?

DrScheme Jr is a text-only version of DrScheme. It requires less memory and provides the same language(s) as DrScheme (without the graphics library), but it implements only the interactions half of DrScheme's interface.

MrEd is PLT's raw graphical Scheme implementation (used to execute DrScheme). MrEd provides a minimal read-eval-print loop, but MrEd does not provide DrScheme's teaching languages, and error messages in MrEd do not provide a source code location.

MzScheme is PLT's Scheme implementation. The language is the same as MrEd without graphics. MzScheme provides little programming support, so its memory requirements are minimal (a few MB usually suffices).

The standard DrScheme distribution includes all of the above programs. Smaller distributions can be downloaded from

<http://www.cs.rice.edu/CS/PLT/packages/drschemejr/>

and

<http://www.cs.rice.edu/CS/PLT/packages/mzscheme/>

Does DrScheme run under DOS or Windows 3.1?

No.

Does DrScheme Jr (PLT's text-only Scheme) run under DOS or Windows 3.1?

No.

I don't have a network connection. Can I obtain DrScheme on floppy disks?

Send two SASFDs (self-addressed stamped floppy disks) to

DrScheme
Programming Languages Team
Rice University -- MS 132
Houston, TX 77005-1892
USA

Please indicate the desired operating system.

How do I install DrScheme?

Obtain a DrScheme distribution from the above address. For Windows, the distribution is an installer program; running this program installs DrScheme. For MacOS, the distributiton is a StuffIt archive; unpacking the archive mostly installs DrScheme, then run **Setup PLT** to complete the installation. For Unix/X, the distribution is a tarred and gzipped file; unpacking the archive and running **./install** installs DrScheme. In all cases, the final download page provides detailed, platform-specific installation instructions.

When I run `plt.exe` to install under Windows, it says "corrupt installation detected." What went wrong?

The real problem may be that the installer is unable to write the DrScheme files to your hard drive, or the installer may be unable to modify the **Start** menu. In this case, check to make sure there is disk space available, and contact your system maintainer to make sure that you have the appropriate access privileges.

How large is the distribution archive?

An average archive is around 2.5 MB.

How much disk space does DrScheme consume?

Around 12 MB in its normal configuration, not including the optional documentation. Some disk space (about 5MB) can be saved by deleting all files with the suffix **.zo**. DrScheme does not need the **.zo** files, but it starts up more slowly without them (so keep them unless you really need the disk space).

5.2 Using DrScheme

How do I find general help for DrScheme?

Select Help Desk in DrScheme's Help menu.

How do I run MrSpidey, DrScheme's program analyzer?

MrSpidey is distributed separately from the standard DrScheme distribution. Download MrSpidey from <http://www.cs.rice.edu/CS/PLT/packages/mrspidey/>

What happened to the Analyze button?

Starting with version 51, PLT distributes DrScheme without the MrSpidey analysis tool. See the previous answer for information about obtaining MrSpidey.

How do I customize DrScheme?

The Edit menu contains a Preferences item that opens the preferences dialog.

How do I turn off parenthesis-flashing and the gray background behind expressions?

Use the Edit|Preferences menu item.

What are the key bindings in DrScheme?

Some basic key bindings are listed in the DrScheme manual, which is accessible via the Help button in DrScheme. A more complete set of key bindings is listed in the MrEd toolbox manual, which is optional documentation that can be downloaded.

Can I change the key bindings in DrScheme?

Technically, yes, but that requires in-depth information about the way that DrScheme is implemented. (The necessary information is part of the MrEd toolbox manual.) DrScheme currently provides no simple way to adjust the keyboard mappings, other than to set the behavior of the Delete key (via the preferences dialog).

What do those yellow-and-black messages mean, and how do I get rid of them?

When text in the definitions window is modified, the current language is changed, or the current library is changed, DrScheme pessimistically assumes that some definition has been changed. In this case, expressions evaluated in the interaction window would use definitions that do not match those currently displayed in the definitions windows. A yellow-and-black message warns you about this potential inconsistency, and suggests that you resolve the inconsistency by clicking the Execute button. To suppress all but the first warning, see the General II panel in the Preferences dialog.

Why can't I type in the interaction window before the the current prompt?

To prevent accidental revisions of the interaction history, DrScheme disallows editing before the current prompt. While old expressions cannot be edited in place, you can copy old expressions to the current prompt by typing Esc-p. Alternatively, place the insertion caret at the end of any old expression in the interactions window and type Enter or Return to copy the expression down to the current prompt.

Is there a DrScheme compiler?

DrScheme is a compiler; each time the user loads a program or enters expression in the interactions window, DrScheme compiles and then executes the program or expression.

PLT's **mzc** transforms Scheme programs into C programs, and then uses a third-party C compiler to produce executable code. Under Windows, either Microsoft Visual C or gcc (a free compiler from Cygnus Solutions) works as the C compiler. Under MacOS, CodeWarrior works. Under Unix, most any compiler works.

For details, see the **mzc** documentation, available from:

<http://www.cs.rice.edu/CS/PLT/packages/doc/>

Can I produce stand-alone executables from Scheme code?

The **mzc** compiler can be used to produce stand-alone executables, but only with significant effort. See the **mzc** documentation for more information.

Can files saved in DrScheme be transferred between platforms?

DrScheme saves files in two formats: *text* and *multimedia*.

The text format is the usual platform-specific text format. Tools for moving files between platforms typically support a “text” transfer mode that adjusts newlines and carriage returns in the text as appropriate.

The multimedia format, used for saving files that contain pictures or formatted text, is platform-independent. Although no other program is able to read DrScheme’s special format, a multimedia-format file can be moved between different platforms (in “binary” mode) and DrScheme will read it correctly on the destination platform.

5.3 Memory and Performance

Does DrScheme really require at least 32 MB of memory?

Yes and no. Although DrScheme starts within 12 MB, using graphics, Help Desk, and other tools increases the amount of Memory DrScheme requires. Most users should find DrScheme usable within 24 MB. However, the total amount of memory needed for DrScheme depends on the kinds of programs the user executes, and we recommend 32 MB in general.

Why does analyzing a program with MrSpidey use even more memory?

MrSpidey is a large program, and it needs a large working memory to calculate the flow of values in a program.

Why is DrScheme so slow at parsing large programs (even compared to PLT’s own MzScheme)?

Unlike MzScheme, DrScheme maintains source location information as it parses programs. This information is used to highlight program text for syntax and run-time errors. Partly, the source-correlating parser and compiler is slower than a typical Scheme parser and compiler (such as MzScheme’s) because it works harder to maintain location information. Partly, it’s slower because DrScheme is implemented as a MzScheme program; MzScheme’s parser and compiler are considerably simpler, and can therefore be implemented in a low-level language that executes ten times faster than Scheme programs with our current technology. We are working on Scheme compiler technology that will narrow this gap and speed up DrScheme’s parser.

Why do programs run more slowly in DrScheme than in other Scheme implementations (including PLT’s own MzScheme)?

Programs run more slowly in DrScheme because DrScheme inserts extra checks into a program to provide information about the location of run-time errors. The MzScheme and MrEd languages (as opposed to MzScheme Debug and MrEd Debug) do not annotate programs in this way.

5.4 Troubleshooting

When I run DrScheme, it is very slow and the disk is constantly running. Why?

You do not have enough memory to run DrScheme. If DrScheme works well for a while, and then starts paging (using the disk a lot), then your memory configuration is borderline for DrScheme. If DrScheme usually works well and has only suddenly started this bad behavior, then perhaps you have written a program that consumes an infinite amount of memory.

My Macintosh has 32 MB of memory, but I am having trouble with DrScheme. What can I do?

Make sure you quit all other applications before starting DrScheme. Also, turn off any non-essential extensions. Select About this Macintosh in the Finder's Apple menu and verify that the system itself uses less than 10 MB.

I have successfully downloaded the installer program for Windows, but the installation fails. Why?

If the installer reports a message such as "corrupt installation detected", the real problem may be that the installer is unable to write the DrScheme files to your hard drive, or the installer may be unable to modify the Start menu. In this case, check to make sure there is disk space available, or contact your system maintainer to make sure that you have the appropriate access privileges.

When I run DrScheme under MacOS, it sometimes freezes the whole machine. Why? Are there any potential software conflicts?

You probably do not have enough memory to run DrScheme. There are no known conflicts between DrScheme and other software.

I think I found a bug. What should I do?

First, read this section to make sure your problem does not have a standard answer. If not, submit a bug report via

<http://www.cs.rice.edu/CS/PLT/Bugs/>

If you do not have access to a web browser, as a last resort send mail to
plt-bugs@cs.rice.edu

How do I send PLT a question?

If you have a question that is not answered in the documentation or this list of "Frequently Asked Questions", send mail to

scheme@cs.rice.edu

Index

- .bak**, 15
- .mred.prefs**, 15
- .mred.resources**, 15
- .sch**, 14
- .scm**, 14
- .ss**, 14
- #i**, 7
- >** prompt, 3

- Advanced language, 4
- Analyze button, 3
- Ask before changing save format preference, 12
- Auto-save files preference, 12
- autosave files, 15

- backup files, 15
- Beginner language, 4
- Break button, 3
- Break menu item, 11

- Check Syntax button, 3
- Check Syntax preferences, 12
- Clear Library menu item, 11
- Close menu item, 10
- Comment Out menu item, 11
- compiler, 20
- configuration files, 15
- Configure Language menu item, 11
- constructor-style output, 7
- Copy menu item, 10
- Correct parens preference, 12
- corrupt installation detected, 22
- Count line and column numbers from one preference, 12
- Cut menu item, 10

- definitions window, 2
- Delete menu item, 10
- disk requirements, 19
- display**, 8
- Display line numbers in buffer; not character off-sets preference, 12
- documentation
 - downloading, 18
- DrScheme Jr, 17
- DrScheme libraries, 16

- editor, 3
- Enable keybindings in menus preference, 12
- error highlighting, 4

- evaluating expressions, 3
- Execute button, 2, 4
- Execute menu item, 11
- execution speed, 21

- file extensions, 14
- Find menu item, 10, 12
- Flash paren match preference, 12
- flashing parenthesis matches, 3
- flashing quotation mark matches, 3
- font preference, 11
- formatting Scheme code, 3
- frequently asked questions, 18

- graphical interface, 2
 - details, 9
- gray highlight region, 3

- Hide Definitions menu item, 11
- Hide Interactions menu item, 11
- Highlight between matching parens preference, 12

- I/O, 8
- Indent All menu item, 11
- Indent menu item, 11
- Indenting preferences, 11
- indenting Scheme code, 3
- Insert Graphic Box menu item, 10
- Insert Image... menu item, 10
- Insert Text Box menu item, 10
- installation instructions, 19
- interactions window, 3
- Intermediate language, 4

- keyboard shortcuts, 12

- languages, 4
 - changing, 4
 - configuring, 7
 - extending, 16
- license, 18

- Map delete to backspace preference, 12
- memory requirements, 18
- menu items, 9
- MrEd Debug, 4
- MrEd language, 4
- MrEd Preferences**, 15
- mred.fnt**, 15
- mred.ini**, 15

- mred.pre**, 15
- MrSpidey, 19
- multimedia file format, 14
- MzScheme Debug, 4
- MzScheme language, 4
- New menu item, 9
- Only warn once when executions and interactions
are not synchronized preference, 12
- Open URL... menu item, 9
- Open... menu item, 9
- output format, 7
- Paste menu item, 10
- plain-text file format, 14
- preference files, 15
- preferences, 11
- Preferences menu item, 10
- Print Definitions... menu item, 10
- Print Interactions... menu item, 10
- printing format, 7
- quasiquote-style output, 8
- read**, 8
- read-char**, 8
- read-eval-print loop, 7
- recycling icon, 2
- Redo menu item, 10
- Replace menu item, 10
- restart**, 17
- Revert menu item, 9
- Save button, 2
- Save Definitions As Text... menu item, 9
- Save Definitions As... menu item, 9
- Save Definitions menu item, 9
- Save Interactions As Text... menu item, 9
- Save Interactions As... menu item, 9
- Save Interactions menu item, 9
- Select Library... menu item, 11
- Show Definitions menu item, 11
- Show Interactions History menu item, 10
- Show Interactions menu item, 11
- Show status-line preference, 12
- stand-alone executables, 21
- status line, 2
- Step button, 3
- storage requirements, 19
- supported platforms, 18
- Toggle Wrap Text menu item, 10
- tools, 16
- Uncomment menu item, 11
- Undo menu item, 10
- Use platform-specific file dialogs preference, 12
- Use separate dialog for searching preference, 12
- Verify exit preference, 12
- Wrap words in editor buffers preference, 12
- write**, 8
- yellow and black messages, 20