PLT DrScheme: Programming Environment Manual

PLT (scheme@plt-scheme.org)

352
Released July 2006

Copyright notice

Copyright ©1996-2006 PLT

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Library General Public License, Version 2 published by the Free Software Foundation. A copy of the license is included in the appendix entitled "License."

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.

Thanks

The following individuals contributed to the implementation of DrScheme and associated tools: Ian Barland, Eli Barzilay, Gann Bierner, John Clements, Richard Cobbe, Greg Cooper, Ryan Culpepper, Carl Eastlund, Moy Easwaran, Matthias Felleisen, Robby Findler, Kathi Fisler, Cormac Flanagan, Matthew Flatt, Sebastian Good, Paul Graunke, Kathy Gray, Dan Grossman, Bruce Hauman, Dave Herman, Mark Krentel, Shriram Krishnamurthi, Mario Latendresse, Guillaume Marceau, Jacob Matthews, Jay McCarthy, Philippe Meunier, Scott Owens, Jamie Raymond, Paul Schlie, Dorai Sitaram, Mike Sperber, Paul Steckler, Jens Axel Søgaard, Francisco Solsona, Sam Tobin-Hochstadt, Neil W. Van Dyke, Anton van Straaten, Dale Vaillancourt, Stephanie Weirich, Noel Welsh, Adam Wick, and ChongKai Zhu.

Thanks to ChongKai Zhu, Ian Barland, Biep Durieux, Tim Hanson, Chihiro Kuraya, Philippe Meunier, Jens Axel Søgaard, Francisco Solsona, Mike Sperber, Reini Urban, and Paolo Zoppetti for their help translating DrScheme's GUI to other languages.

This manual was typeset 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 July 28, 2006.

Contents

1 About DrScheme **1**

2 Interface Essentials **2**

2.1 Buttons 2

2.2 The Editor 4

2.3 The Interactions Window 4

2.4 Tabbed Editing 5

2.5 Errors 5

2.6 Languages 5

2.7 Executables 8

2.8 Printed Results 9

2.8.1 Constructor-style Output 9

2.8.2 Quasiquote-style Output 9

2.9 Input and Output 10

2.10 XML 11

2.11 Test Cases 12

3 Interface Reference **13**

3.1 Menus 13

3.1.1 File 13

3.1.2 Edit 14

3.1.3 View 14

3.1.4 Language 15

3.1.5 Scheme 15

3.1.6 Special 16

3.1.7 Windows . . . 17
3.1.8 Help . . . 17
3.2 Preferences . . . 17
3.3 Keyboard Shortcuts . . . 19
3.3.1 Moving Around . . . 19
3.3.2 Editing Operations . . . 20
3.3.3 File Operations . . . 21
3.3.4 Searching . . . 21
3.3.5 Miscellaneous . . . 21
3.3.6 Interactions . . . 21
3.3.7 Defining Custom Shortcuts . . . 21
3.4 DrScheme Files . . . 22
3.4.1 Program Files . . . 22
3.4.2 Backup and Autosave Files . . . 22
3.4.3 Preference Files . . . 23

4 Extending DrScheme **24**
4.1 Teachpacks . . . 24
4.2 Tools . . . 25
4.3 Environment Variables . . . 25

5 Frequently Asked Questions **27**
5.1 Supported Operating Systems and Installation . . . 27
5.2 Using DrScheme . . . 28
5.3 Memory and Performance . . . 29
5.4 Troubleshooting . . . 30

License **31**

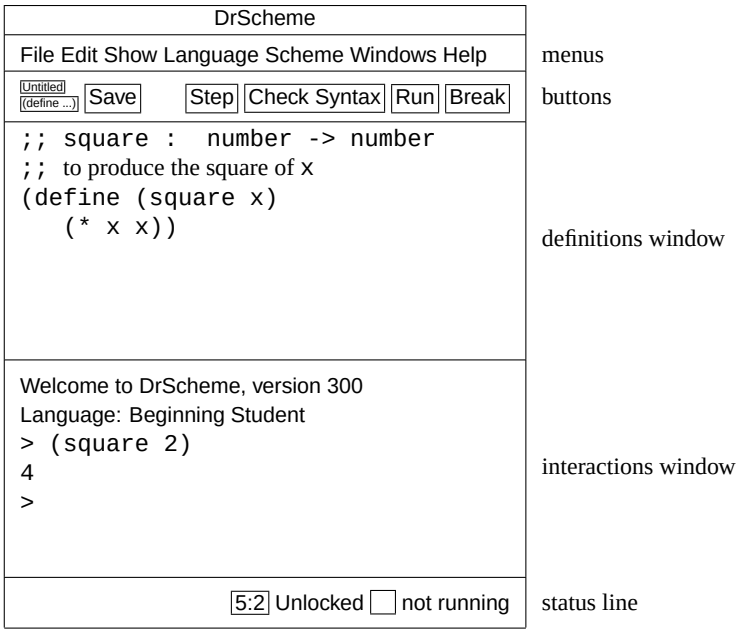
Index **35**

1. About DrScheme

DrScheme is a graphical environment for developing programs using the Scheme programming language. DrScheme runs under Windows (95 and up), Mac OS X, and Unix/X.

2. 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 Beginning Student, which is the default language.

Clicking the Run 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 Buttons

The left end of the row of buttons in DrScheme contains a miniature button with the current file's name. Clicking the button opens a menu that shows the file's full pathname. Selecting one of the menu entries produces an open-file dialog starting in the corresponding directory.

Below the filename button is a (define ...) button for a popup menu of names that are defined in the definitions window. Selecting an item from the menu moves the blinking caret to the corresponding definition.

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—which appears only for the *How to Design Programs* teaching languages "Beginning Student" through "Intermediate Student with Lambda"—starts the Stepper, 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 several new buttons: these buttons allow navigation of the evaluation as a series of steps.

The Debug button—which does *not* appear for the *How to Design Programs* teaching languages—starts a more conventional stepping debugger. It runs the program in the definitions window like the Run button, but also opens a debugging panel with Pause, Continue, and Step buttons. A newly started program is paused the program's first possible pause point, and the current pause location is shown with a green arrow. Click the Continue button to continue running the program, click Step to run until the next possible pause point, and right-click on an expression's open or close parenthesis to set or remove an explicit pause. (Valid pause points are highlighted with a pink dot as you mouse over the program text.) When the program is paused, move the mouse over a variable to display its value in the debugging panel to the right of the buttons. When pausing at an expression's result, the result is shown to the left of the debugging panel's buttons, and the result can be changed by right-clicking the pause point. Click the Stop button to stop debugging so that the program in the definitions window can be edited. Debugging also stops when all expressions in the definition window have been evaluated.

Clicking the Check Syntax button annotates the program text in the definitions window. It add these annotations:

- **Syntactic Highlighting** Imported variables and locally defined variables are highlighted with color changes.
- **Lexical Structure** The lexical structure is shown with arrows overlaid on the program text. When the mouse cursor passes over a variable, DrScheme draws an arrow from the binding location to the variable, or from the binding location to every bound occurrence of the variable.

  Additionally, right-button clicking (or control-clicking under Mac OS X) on a variable activates a popup menu that lets you jump from binding location to bound location and vice versa, α-rename the variable, or tack the arrows so they do not disappear.
- **Tail Calls** Any subexpression that is (syntactically) in tail-position with respect to its enclosing context is annotated by drawing a light purple arrow from the tail expression to its surrounding expression.
- **Require Annotations** Right-button clicking (or control-clicking under Mac OS X) on the argument to `require` activates a popup menu that lets you open the file that contains the `required` module.

  Passing the mouse cursor over a `require` expression inside a module shows all of the variables that are used from that `require` expression. Additionally, if no variables are used from that require expression, it is colored like an unbound variable.

  Also, passing the mouse cursor over a variable that is imported from a module shows the module that it is imported from in a status line at the bottom of the frame.

The Run 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 Run 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 kill the evaluation immediately.

2.2 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.

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 blinking 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.

DrScheme also rewrites parenthesis as you type them, in order to make them match better. If you type a closing parenthesis ")", a closing square bracket "[", or a closing curley brace "}", and DrScheme can match it back to some earlier opening parenthesis, bracket, or brace, DrScheme will change what you type to match. DrScheme also rewrites open square brackets, usually to an open parenthesis. There are some exceptions, where opening square brackets are not automatically changed to parentheses:

- if the square bracket is after the `cond` or `provide/contract` keywords,
- if the square bracket is after a parenthesis that is after a `let`, `let*`, `letrec` (or similar) keyword,
- if the square bracket is in a `new` expression beginning the keyword arguments,
- if the square bracket is in a sequence and the s-expression before in the sequence is a compound expression, DrScheme uses the same kind parenthesis, brace, or bracket as before, or
- if the square bracket is in the middle of string, comment, character, or symbol.

The upshot of DrScheme's trickery: always use the (presumably unshifted) square brackets on your keyboard to type parenthesis. For example, when typing this:

```
(define (length l)
  (cond
   [(empty? l) 0]
   [else (+ 1 (length (rest l)))]))
```

If you always type "[" and "]" where any of the square brackets or parentheses appear, DrScheme will change the square brackets to match the code above.

Of course, these features can be disabled in the preferences dialog (see section 3.2). Also, in case DrScheme is only being too clever occasionally, holding down the control key while typing a parenthesis prevents DrScheme from changing it.

2.3 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 blinking 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. You can force DrScheme to evaluate the expression by typing Alt-Return or Command-Return (depending on your platform).

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.

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

2.4 Tabbed Editing

DrScheme's allows you to edit multiple files in a single window, via tabs. The File | New Tab creates a new tab to show a new file. Each tab has its own interactions window.

In the General sub-pane of the Editing pane in the preferences window, a checkbox labelled Open files in separate tabs causes DrScheme to open files in new tabs in the frontmost window, rather than opening a new window for the file.

The key bindings Control-Pageup and Control-Pagedown move between tabs. Under Mac OS X, Command-Shift-Left and Command-Shift-Right also move between tabs.

2.5 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.

2.6 Languages

DrScheme supports multiple dialects of Scheme. The name of the current evaluation language always appears in the top of the interactions window. To choose a different language, select the Language | Choose Language... menu item. After changing the language, click Run to reset the language in the interactions window.

Five of DrScheme's languages are specifically designed for teaching:

- *Beginning Student* is a small version of Scheme that is tailored for beginning computer science students.
- *Beginning Student with List Abbreviations* is an extension to Beginning Student that prints lists with `list` instead of `cons`, and accepts quasiquoted input.
- *Intermediate Student* adds local bindings and higher-order functions.
- *Intermediate Student with Lambda* adds anonymous functions.
- *Advanced Student* adds mutable state.

The teaching languages different from conventional Scheme in a number of ways, described below.

DrScheme also supports several languages for experienced programmers:

- *Standard (R^{5}RS)* contains those primitives and syntax defined in the R^{5}RS Scheme standard. See the *Revised^5 Report on the Algorithmic Language Scheme* for details.

- *PLT Textual (MzScheme)* extends R^{5}RS with exceptions, threads, objects, modules, components, regular expressions, TCP support, filesystem utilities, and process control operations. See *PLT MzScheme: Language Manual* for details.

- *PLT Graphical (MrEd)* extends MzScheme with a graphical toolbox for creating GUI applications (with special support for editor applications, hence the "Ed" in "MrEd"). See also *PLT MrEd: Graphical Toolbox Manual*.

- *PLT Pretty Big* extends MrEd with the forms of the Advanced Student teaching language and more.[1] It is useful as a step past Advanced Student, or for implementing MrEd programs with a richer base syntax and set of primitives.

  Note that there some forms (for example, `define-struct`) that appear in both Advanced and MrEd, but with slightly different semantics. In all such cases, the PLT Pretty Big language uses the forms from the MrEd language (In the case of `define-struct`, Advanced makes the structure transparent, whereas MrEd does not).

- `module` requires that the definitions window contain only a single module declaration, as defined in *PLT MzScheme: Language Manual*. The module explicitly declares the language for the module's body.

The Language | Choose Language... dialog contains a Show Details button for configuring certain details of the language specification. 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.

The teaching languages differ from conventional Scheme in a number of ways:

- *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. The teaching languages are case-sensitive by default, and other languages are usually case-insensitive. Case-sensitivity can be adjusted through the detail section of the language-selection dialog.

- *All numbers are exact unless `#i` is specified* — In the Beginning Student through Intermediate Student with Lambda languages, numbers containing a decimal point are interpreted as exact numbers. This interpretation allows students to use familiar decimal notation without inadvertently triggering inexact arithmetic. Exact numbers with decimal representations are also printed in decimal. Inexact inputs and results are explicitly marked with `#i`.

- *Procedures must take at least one argument* — In the Beginning Student through Intermediate Student 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 Beginning Student through Intermediate Student 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.

[1] More precisely, Pretty Big is MrEd extended with the following MzLib libraries (see *PLT MzLib: Libraries Manual*): **etc.ss**, **file.ss**, **list.ss**, **class.ss**, **unit.ss**, **unitsig.ss**, **include.ss**, **defmacro.ss**, **pretty.ss**, **string.ss**, **thread.ss**, **math.ss**, **match.ss**, and **shared.ss**.

- *Top-level required at function call position* — In the Beginning Student languages, procedure calls must be of the form `(top-level-identifier ...)`, and the number of actual arguments must match the number of formal arguments if `top-level-identifier` is `define`d. This restriction helps detect confusing misuses of parentheses, such as `(x)` where `x` is a function argument. DrScheme can detect such mistakes syntactically because Beginning Student does not support higher-order procedures.

- *Primitive and defined functions allowed only in function call position* — In Beginning Student languages, the name of a primitive operator or of a defined function can be used only after the open-parenthesis of a function call (except where teachpack extensions allow otherwise, as in the `convert-gui` extension). Incorrect uses of primitives trigger a syntax error. Incorrect uses of defined names trigger a run-time error. DrScheme can detect such mistakes because Beginning Student does not support higher-order procedures.

- `lambda` *allowed only in definitions* — In the Beginning Student through Intermediate Student languages, `lambda` (or `case-lambda`) may appear only in a definition, and only as the value of the defined variable.

- *Free variables are not allowed* — In the Beginning Student through Advanced Student languages, every variable referenced in the definitions window must be defined, pre-defined, or the name of a local function argument.

- `quote` *works only on symbols,* `quasiquote` *disallowed* — In the Beginning Student language, `quote` and `'` can specify only symbols. This restriction avoids the need to explain to beginners why `1` and `'1` are equivalent in standard Scheme. In addition, `quasiquote`, `` ` ``, `unquote`, `,`, `unquote-splicing`, and `,@` are disallowed.

- *Unmatched* `cond`/`case` *is an error* — In the Beginning Student through Advanced Student 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* `true` *or* `false` — In the Beginning Student through Advanced Student languages, an expression whose value is treated as a boolean must return an actual boolean, `true` or `false`. 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 Beginning Student through Advanced Student 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 Beginning Student through Advanced Student 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.

- `set!` *disallowed on arguments* — In the Advanced Student language, `set!` cannot be used to mutate variables bound by `lambda`. This restriction ensures that the substitution model of function application is consistent with DrScheme's evaluation.

- *Improper lists disallowed* — A *proper list* is either an empty list or a list created by `cons`ing onto a proper list. In the Beginning Student through Advanced Student languages, `cons` constructs only *proper lists*, signaling an error if the second argument is not a proper list. Since beginning students do not need improper lists, this restriction help detect logical errors in recursive functions.

- *Dot is disallowed* — In the Beginning Student through Advanced Student languages, a delimited period is disallowed, (e.g., as an improper-list constructor in a quoted form, or for defining multi-arity procedures).

- *Keywords disallowed as variable names* — In the Beginning Student through Advanced Student languages, all syntactic form names are keywords that cannot be used as variable names.

- *Re-definitions are disallowed* — In the Beginning Student through Advanced Student languages, top-level names can never be re-defined.

- *Function definitions are allowed only in the definitions window* — In the Beginning Student languages, function definitions are not allowed in the interactions window.

The teaching languages also deviate from traditional Scheme in printing values. Different printing formats can be selected for any language through the detail section of language-selection dialog.

- *Constructor-style output* — See Constructor-style Output (§2.8.1).
- *Quasiquote-style output* — See Quasiquote-style Output (§2.8.2).
- *Rational number printing* – In the teaching languages, all numbers that have a finite decimal expansion are printed in decimal form. For those numbers that do not have a finite decimal expansion (such as 4/3) DrScheme offers a choice. It either prints them as mixed fractions or as repeating decimals, where the repeating portion of the decimal expansion is shown with an overbar. In addition, DrScheme only shows the first 25 digits of the number's decimal expansion. If there are more digits, the number appears with an ellipses at the end. Click the ellipses to see the next 25 digits of the expansion.

  This setting controls only the initial display of a number. Right-clicking or control-clicking (Mac OS X) on the number lets you change from the fraction representation to the decimal representation.
- `write` *output* — Prints values with `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 ([lt (list 0)]) (list lt lt))` prints as

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

  instead of

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

2.7 Executables

DrScheme's Create Executable... menu item lets you create an executable for your program that you can start without first starting DrScheme. To create an executable, first save your program to a file and set the language and teachpacks. Click Run, just to make sure that the program is working as you expect. The executable you create will not have a read-eval-print-loop, so be sure to have an expression that starts your program running in the definitions window before creating the executable.

Once you are satisfied with your program, choose the Create Executable... menu item from the Scheme menu. You will be asked to choose an executable file name or an archive file name. In the latter case, unpack the generated archive (on this machine or another one) to access the executable. In either case, you will be able to start the executable in the same way that you start any other program on your computer.

The result of Create Executable... is either a *launcher executable*, a *stand-alone executable*, or a *distribution archive*, and it uses either a *MzScheme* (textual) or *MrEd* (graphical) engine. For programs implemented with certain languages, Create Executable... will prompt you to choose the executable type and engine, while other languages support only one type or engine.

Each type has advantages and disadvantages:

- A *launcher executable* uses the latest version of your program source file when it starts. It also accesses library files from your DrScheme installation when it runs. Since a launcher executable contains specific paths to access those files, launchers usually cannot be moved from one machine to another.
- A *stand-alone executable* embeds a compiled copy of your program and any Scheme libraries that your program uses. When the executable is started, it uses the embedded copies and does not need your original source file. It may, however, access your DrScheme installation for DLLs, frameworks, shared libraries, or helper executables. Consequently, a stand-alone executable usually cannot be moved from one machine to another.

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 2.1: Comparison of constructor-style output to `write`

- A *distribution archive* packages a stand-alone executable together with any needed DLLs, frameworks, shared libraries, and helper executables. A distribution archive can be unpacked and run on any machine with the same operating system as yours.

TIP: Disable debugging in the language dialog before creating your executable. With debugging enabled, you will see a stack trace with error messages, but your program will run more slowly. To disable debugging, open the language dialog, click the Show Details button, and select No debugging or profiling, if it is available.

2.8 Printed Results

This section describes the different formats that DrScheme uses for printing results in the interactions window. Each of the different settings here also apply to the `print` primitive. That is, printing in the interactions window is identical to output produced by the `print` primitive.

2.8.1 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 2.1 shows the differences between values printed in constructor style and values printed with `write`.

2.8.2 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)))
```

This example:

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

in quasiquote-style printing prints as:

```
`(lambda (x) ,(box `(lambda (y) (+ x y))))
```

2.9 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 blinking caret.) Type a number into the box and hit Enter, and that number becomes the result of the `(read)` expression. Once text is submitted for an input box, it is moved outside the input box, and when DrScheme shows a new prompt, it hides the interaction box. Thus, if you type `5` in the above input box and hit Return, the result appears as follows:

```
> (read)
5
5
> _
```

In this case, the first `5` is the input, and the second `5` is the result of the `(read)` expression. The second `5` is colored blue, as usual for a result printed by DrScheme.

Output goes into the interaction window directly. If you run 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, and the final result is printed last. The displayed output is drawn in purple. (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))
> _
```

Depending on the input operation, you may enter more text into an input box than is consumed. In that case, the leftover text remains in the input stream for later reads. For example, in the following interaction, two values are provided in response to the first `(read)`, so the second value is returned immediately for the second `(read)`:

```
> (read)
5 6
5
> (read)
6
> _
```

The following example illustrates that submitting input with Return inserts a newline character into the input stream:

```
> (read)
5
5
> (read-char)
#\newline
> _
```

2.10 XML

DrScheme has special support for XML concrete syntax. The Special menu's Insert XML Box menu inserts an embedded editor into your program. In that embedded editor, you type XML's concrete syntax. When a program containing an XML box is evaluated, the XML box is translated into an x-expression (or xexpr). Xexprs are s-expression representation for XML expressions. Each xexpr is a list whose first element is a symbol naming the tag, second element is an association list representing attributes and remaining elements are the nested XML expressions.

XML boxes have two modes for handling whitespace. In one mode, all whitespace is left intact in the resulting xexpr. In the other mode, any tag that only contains nested XML expressions and whitespace has the whitespace removed. You can toggle between these modes by right-clicking or control-clicking (Mac OS X) on the top portion of the XML box.

In addition to containing XML text, XML boxes can also contain Scheme boxes. Scheme boxes contain Scheme

expressions. These expressions are evaluated and their contents are placed into the containing XML box's xexpr. There are two varieties of Scheme box: the standard Scheme box and the splicing Scheme box. The standard Scheme box inserts its value into the containing xexpr. The contents of the splice box must evaluate to a list and the elements of the list are "flattened" into the containing xexpr. Right-clicking or control-clicking (Mac OS X) on the top of a Scheme box opens a menu to toggle the box between a Scheme box and a Scheme splice box.

2.11 Test Cases

DrScheme also includes support for creating test cases as part of the program text. This support is designed as an aid for students building tests as part of the How to Design Programs design recipes.

Test cases in DrScheme are written in special boxes. To create one, choose Insert Test Case from the Scheme menu in the DrScheme window. The test cases consists of three editable areas. From the top, the first is a comment that names the test case. The second is an expression to test. The third is the expected result. Below the expected result is a light-blue box that will contain the actual result of the expression to be tested. Clicking on the triangle in the top-right hides the expression to test and the expected and actual results.

To run test cases, simply click Run. The top-right corner of the test case will change to either be a check box or a red X, indicating success or failure of the test case.

To disable all of the test cases in the definitions window, choose the Disable All Test Cases menu item in the Scheme menu.

3. Interface Reference

3.1 Menus

3.1.1 File

- New — Creates a new DrScheme window.
- Open... — Opens a find-file dialog for choosing a file to load into a definitions window.
- Open Recent — Lists recently opened files. Choosing one of them opens that file for editing.
- Install PLT File... — Opens a dialog asking for the location of the PLT file (either on the local disk or on the web) and installs the contents of the PLT file.
- 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 Other — Contains these sub-items
  - Save Definitions As Text... — Like Save Definitions As..., but the file is saved in plain-text format (see DrScheme Files (§3.4.1)). Subsequent saves also write in plain-text format.
  - Save Interactions — Saves the contents of the interactions window to a file. If the interaction constants 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... — Like Save Interactions As..., but the file is saved in plain-text format (see DrScheme Files (§3.4.1)). Subsequent saves are write in plain-text format.
- Log Definitions and Interactions... — Starts a running of log of the text in the interactions and definitions windows, organized by executions. In a directory of your choosing, DrScheme saves files with the names 01-definitions, 01-interactions, 02-definitions, 02-interactions, etc as you interact with various programs.
- 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.
- Search in Files... — Opens a dialog where you can specify the parameters of a multi-file search. The results of the search are displayed in a separate window.
- Close — Closes this DrScheme window. If this window is the only open DrScheme window, DrScheme quits.
- Quit or Exit Exits DrScheme. (Under Mac OS X this menu item is in the apple menu.)

3.1.2 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 (and boxed-subwindow) maintains its own 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 — or Clear — Deletes the selected text.
- Select All — Highlights the entire text of the buffer.
- Wrap Text — Toggles between wrapped text and unwrapped text in the window.
- Find... — Opens a search dialog or, depending on the preferences, an interactive searching window attached to the frame.
- Find Again — Finds the next occurrence of the text that was last searched for.
- Replace & Find Again — Replaces the selection with the replace string (if it matches the find string) and finds the next occurrence of the text that was last searched for.
- Keybindings —
  - Show Active Keybindings — Shows all of the keybindings available in the current window.
  - Add User-defined Keybindings... — Choosing this menu item opens a file dialog where you can select a file containing Scheme-definitions of keybindings. See Defining Custom Shortcuts (§3.3.7) for more information.
- Preferences... — Opens the preferences dialog. See section 3.2. (In Mac OS X, this menu item is under the apple menu.)

3.1.3 View

One each of the following show/hide pairs of menu items appears at any time.

- Show Definitions — Shows the definitions window.
- Hide Definitions — Hides the definitions window.
- Show Interactions — Shows interactions window.
- Hide Interactions — Hides interactions window.
- Show Program Contour — Shows a “20,000 foot” overview window along the edge of the DrScheme window. Each pixel in this window corresponds to a letter in the program text.
- Hide Program Contour — Hides the contour window.
- Show Module Browser — Shows the module DAG rooted at the currently opened file in DrScheme.

- Hide Module Browser — Hides the module browser.
- Show Toolbar — Makes the toolbar (along the top of DrScheme's window) and the status line (along the bottom) visible.
- Hide Toolbar — Hides the toolbar (along the top of DrScheme's window) and the status line (along the bottom).
- Show Profile — Shows the current profiling report. This menu is useful only if you have enabled profiling in the Choose Language... dialog's Details section. Profiling does not apply to all languages.
- Hide Profile — Hides any profiling information currently displayed in the DrScheme window.
- Show Tracing — Shows a trace of functions called since the last time Run was clicked. This menu is useful only if you have enabled tracing in the Choose Language... dialog's Details section. Profiling does not apply to all languages.
- Hide Tracing — Hides the tracing display.
- Split — Splits the current window in half to allow for two different portions of the current window to be visible simultaneously.
- Collapse — If the window has been split before, this menu item becomes enabled, allowing you to collapse the split window.

Note: whenever a program is run, the interactions window is made visible if it is hidden.

3.1.4 Language

- Choose Language... — Opens a dialog for selecting the current evaluation language. Click Run to make the language active in the interactions window. See section 2.6 for more information about the languages.
- Add Teachpack... — Opens a find-file dialog for choosing a teachpack to extend the current language. Click Run to make the teachpack available in the interactions windows. See Extending DrScheme (§4) for information on creating teachpacks.
- Clear All Teachpacks — Clears all of the current teachpacks. Click Run to clear the teachpack from the interactions window.

In addition to the above items, a menu item for each teachpack that clears only the corresponding teachpack.

3.1.5 Scheme

- Run — Resets the interactions window and runs the program in the definitions window.
- Break — Breaks the current evaluation.
- Kill — Terminates the current evaluation.
- Clear Error Highlight — Removes the red background that signals the source location of an error.
- Create Executable... — Creates a separate launcher for running your program. See Executables (§2.7) for more info.
- Module Browser... — Prompts for a file and then opens a window showing the module import structure for the module import DAG starting at the selected module.

  The module browser window contains a square for each module. The squares are colored based on the number of lines of code in the module. If a module has more lines of code, it gets a darker color.

In addition, for each normal import, a blue line drawn is from the module to the importing module. Similarly, purple lines are drawn for each for-syntax import. In the initial module layout, modules to the left import modules to the right, but since modules can be moved around interactively, that property might not be preserved.

To open the file corresponding to the module, right-click or control-click (Mac OS X) on the box for that module.

- Reindent — Indents the selected text according to the standard Scheme formatting conventions. (Pressing the Tab key has the same effect.)
- Reindent 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 with Semicolons — Puts ";" characters at each of the beginning of each selected line of text.
- Comment Out with a Box — Boxes the selected text with a comment box.
- Uncomment — Removes all ";" characters at the start of each selected line of text or removes a comment box around the text. Uncommenting only removes a ";" if it appears at the start of a line and it only removes the first ";" on each line.
- Disable All Test Cases — Disables all test case boxes (see section 2.11)

3.1.6 Special

- Insert Comment Box — Inserts a box that is ignored by DrScheme; use it to write comments for people who read your program.
- Insert Image... — Opens a find-file dialog for selecting an image file in GIF, BMP, XBM, XPM, PNG, or JPG format. The image is treated as a value.
- Insert Fraction... — Opens a dialog for a mixed-notation fraction, and inserts the given fraction into the current editor.
- Insert Large Letters... — Opens a dialog for a line of text, and inserts a large version of the text (using semicolons and spaces).
- Insert λ — Inserts the symbol λ (as a Unicode character) into the program. The λ symbol is normally bound the same as `lambda`.
- Insert Java Comment Box — Inserts a box that is ignored by DrScheme. Unlike the Insert Comment Box menu item, this is designed for the ProfessorJ language levels. See the *ProfessorJ Beginner Language*, *ProfessorJ Intermediate Language*, and *ProfessorJ Advanced Language* manuals for details.
- Insert Java Interactions Box — Inserts a box that will allows Java expressions and statements within Scheme programs. The result of the box is a Scheme value corresponding to the result(s) of the Java expressions. At this time, Scheme values cannot enter the box. The box will accept one Java statement or expression per line.
- Insert Java Examples — Inserts a box that allows field-like definitions in the definitions window. The names created by this box can be used in any definition or test case box that follows and in the Interactions window. This box is intended only for use in the ProfessorJ language levels and should only appear at the top level. See the *ProfessorJ Beginner Language*, *ProfessorJ Intermediate Language*, and *ProfessorJ Advanced Language* manuals for details.
- Insert XML Box — Inserts an XML; see XML in DrScheme (§2.10) for more information.
- Insert Scheme Box — Inserts a box to contain Scheme code, typically used inside an XML box; see also XML in DrScheme (§2.10).
- Insert Scheme Splice Box — Inserts a box to contain Scheme code, typically used inside an XML box; see also XML in DrScheme (§2.10).

- Insert Test Case — Creates a new test case box; see section 2.11.
- Insert Pict Box — Creates a box for generating a Slideshow picture. Inside the pict box, insert and arrange Scheme boxes that produce picture values.

3.1.7 Windows

- Bring Frame to Front... — Opens a window that lists all of the opened DrScheme frames. Selecting one of them brings the window to the front.
- Most Recent Window — Toggles between the currently focused window and the one that most recently had the focus.

Additionally, after the above menu items, this menu contains an entry for each window in DrScheme. Selecting a menu item brings the corresponding window to the front.

3.1.8 Help

- Help Desk — Opens the Help Desk. This is the clearing house for all documentation about DrScheme and its language.
- About DrScheme... — Shows the credits for DrScheme.
- Related Web Sites — Provides links to related web sites.
- Tool Web Sites — Provides links to web sites for installed tools.
- Interact with DrScheme in English — Changes DrScheme's interface to use English; the menu item appears only when the current language is not English. Additional menu items switch DrScheme to other languages.

3.2 Preferences

The preferences dialog comprises several panels:

- Font

  This panel controls the main font used by DrScheme.
- Colors

  The Coloring panel has several subpanels that enable you to configure the colors that DrScheme uses for the editor background, for highlighting matching parentheses, for online coloring, and for Check Syntax.
- Background Color

  This panel configures the background color for the editors in DrScheme.
- Editing

  The Editing panel consists of several sub-panels:
  - Indenting
    This panel controls which keywords DrScheme recognizes for indenting, and how each keyword is treated.
  - General
    * Number of recent items — controls the length of the Open Recent menu (in the File menu).
    * Auto-save files — If checked, the editor generates autosave files (see DrScheme Files (§3.4.2)) for files that have not been saved after five minutes.

    * Backup files — If checked, when saving a file for the first time in each editing session, the original copy of the file is copied to a backup file in the same directory. The backup files have the same name as the original, except that they end in either **.bak** or ∼.
    * Map delete to backspace — If checked, the editor treats the Delete key like the Backspace key.
    * Show status-line — If checked, DrScheme shows a status line at the bottom of each window.
    * Count column numbers from one — If checked, the status line's column counter counts from one. Otherwise, it counts from zero.
    * 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.
    * Wrap words in editor buffers — If checked, DrScheme editors auto-wrap text lines by default. Changing this preference affects new windows only.
    * 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.
    * Reuse existing frames when opening new files — If checked, new files are opened in the same DrScheme window, rather than creating a new DrScheme window for each new file.
    * Enable keybindings in menus — If checked, some DrScheme menu items have keybindings. Otherwise, no menu items have key bindings. This preference is designed for people who are comfortable editing in Emacs and find the standard menu keybindings interfere with the Emacs keybindings.
    * Color syntax interactively — If checked, DrScheme colors your syntax as you type.
    * Automatically print to PostScript file — If checked, printing will automatically save PostScript files. If not, printing will use the standard printing mechanisms for your computer.
    * Open files in separate tabs (not separate windows) – If checked, DrScheme will use tabs in the front-most window to open new files, rather than creating new windows for new files.
    * Automatically open interactions window when running a program – If checked, DrScheme shows the interactions window (if it is hidden) when a program is run.
  - Scheme
    * 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 blinking 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 "(". Also, the editor changes type "[" to match the context (as explained in section 2.2).
    * Flash paren match — If checked, typing a closing parenthesis, square bracket, or quotation mark flashes the matching open parenthesis/bracket/quote.

- Warnings
  - Ask before changing save format — If checked, DrScheme consults the user before saving a file in non-text format (see DrScheme Files (§3.4.1)).
  - Verify exit — If checked, DrScheme consults the user before exiting.
  - 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 teachpack is changed without a corresponding click on Run. Otherwise, the warning appears on every interaction.
  - Ask about clearing test coverage — If checked, when test coverage annotations are displayed DrScheme prompts about removing them. This setting only applies to the PLT languages. DrScheme never asks in the teaching languages.
  - Check for newer PLT Scheme versions — If checked, DrScheme will periodically poll a server to determine whether a newer version of DrScheme is available.

- Profiling

  This preference panel configures the profiling report. The band of color shows the range of colors that profiled functions take on. Colors near the right are used for code that is not invoked often and colors on the right are used for code that is invoked often.

If you are interested in more detail at the low end, choose the "Square root" check box. If you are interested in more detail at the upper end, choose the "Square" check box.

- Browser

  The Use Help Desk browser for external URLs check box determines whether Help Desk visits URLs (out of our documentation) in a platform-specific browser or within Help Desk itself.

  This preferences panel also allows you to configure your HTTP proxy. Contact your system administrator for details.

3.3 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 trigger 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 Mac OS X, 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.

If you are most familiar with Emacs-style key bindings, you should uncheck the Enable keybindings in menus preference. Many of the keybindings below are inspired by Emacs.

3.3.1 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
- M-C-left move backwards to the nearest editor box
- A-C-left move backwards to the nearest editor box
- M-C-right move forward to the nearest editor box
- A-C-right move forward to the nearest editor box
- M-C-up move up out of an embedded editor
- A-C-up move up out of an embedded editor
- M-C-down move down into an embedded editor
- A-C-down move down into an embedded editor

3.3.2 Editing Operations

- C-_ undo
- C-+ redo
- C-x u undo
- 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
- M-C-t transpose sexpressions
- M-C-m toggle dark green marking of matching parenthesis
- M-C-k cut complete sexpression
- M-(wrap selection in parentheses
- M-[wrap selection in square brackets
- M-{ wrap selection in curly brackets
- M-S-L wrap selection in `(lambda () ...)` and put the insertion point in the arglist of the lambda
- C-c C-o move-sexp-out: the sexpression following the insertion point is put in place of its containing sexpression
- C-c C-l insert-let-binding: wraps a let around the sexpression following the insertion point and puts a printf in at that point (useful for debugging).
- M-o toggle overwrite mode

3.3.3 File Operations

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

3.3.4 Searching

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

3.3.5 Miscellaneous

- F5 Run

3.3.6 Interactions

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

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

3.3.7 Defining Custom Shortcuts

The Add User-defined Keybindings... menu item in the Keybindings sub-menu of Edit selects a file containing Scheme definitions of keybindings. The file must contain a single module that uses a special keybindings language, `(lib "keybinding-lang.ss" "framework")`. For example, a file named **mykeys.ss** for keybindings might contain the following code:

```
(module mykeys (lib "keybinding-lang.ss" "framework")
  ...)
```

The keybindings language includes all of MzScheme, `(lib "mred.ss" "mred")`, `(lib "class.ss")`, and one addition, a `keybinding` form:

```
(keybinding keybinding-string-expr keybinding-proc-expr)
```

The *`keybinding-string-expr`* must produce a suitable first argument for `map-function` in `keymap%`, and the *`keybinding-proc-expr`* must produce a suitable second argument for `add-function` in `keymap%`.

3.4 DrScheme Files

3.4.1 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, and it uses an ASCII encoding (so that different ways of transferring the file are unlikely to corrupt the file).

3.4.2 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 Unix and Mac OS X, 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 Unix and Mac OS X, 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.

3.4.3 Preference Files

On start-up, DrScheme reads configuration information from a preferences file. The name and location of the preferences file depends on the platform and user:[1]

- Under Unix, preferences are stored in a **.plt-scheme** subdirectory in the user's home directory, in a file **plt-prefs.ss**.
- Under Windows, preferences are stored in a file **plt-prefs.ss** in a sub-directory **PLT Scheme** in the user's application-data folder as specified by the Windows registry; the application-data folder is usually **Application Data** in the user's profile directory.
- Under Mac OS X, preferences are stored in **org.plt-scheme.prefs.ss** in the user's preferences folder.

A lock file is used while modifying the preferences file, and it is created in the same directory as the preferences file. Under Windows, the lock file is named **_LOCKplt-prefs.ss**; under Unix, it is **.LOCK.plt-prefs.ss**; under Mac OS X, it is **.LOCK.org.plt-scheme.prefs.ss**.

If the user-specific preferences file does not exist, and the file **plt-prefs.ss** in the **defaults** collection does exist, then it is used for the initial preference settings. (See Library Collections and MzLib, §16 in *PLT MzScheme: Language Manual* for more information about collections.) This file thus allows site-specific configuration for preference defaults. To set up such a configuration, start DrScheme and configure the preferences to your liking. Then, exit DrScheme and copy your preferences file into the **defaults** collection as **plt-prefs.ss**. Afterward, users who have no preference settings already will get the preference settings that you chose.

[1] The MzScheme procedure `find-system-path` returns the platform-specific path when given the argument `'pref-file`.

4. Extending DrScheme

DrScheme supports two forms of extension to the programming environment:

- A *teachpack* extends the set of procedures that are built into a language in DrScheme. For example, a teachpack might extend the Beginning Student language with a procedure for playing sounds.

  Teachpacks are particularly useful in a classroom setting, where an instructor can provide a teachpack that is designed for a specific exercise. To use the teachpack, each student must download the teachpack file and select it through the Language | Add Teachpack... 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.

4.1 Teachpacks

Teachpacks are designed to supplement student programs with code that cannot be expressed in a teaching language. For example, to enable students to play hangman, we supply a teachpack that

- implements the random choosing of a word
- maintains the state variable of how many guesses have gone wrong
- manages the GUI.

All these tasks are beyond students in the third week and/or impose memorization of currently useless knowledge on students. The essence of the hangman game, however, is not. The use of teachpacks enables the students to implement the interesting part of this exercise and still be able to enjoy today's graphics without the useless memorization.

A single Scheme source file defines a teachpack (although the file may access other files via `require`). The file must contain a module, according to the naming convention laid out in the MzScheme manual (the name of the file must be the name of the module, with an additional `.scm` or `.ss` extension on the filename). Each exported syntax definition or value definition from the module is provided as a new primitive form or primitive operation to the user, respectively.

As an example, the following teachpack provides a lazy cons implementation. To test it, be sure to save it in a file named **lazycons.scm**.

```
(module lazycons mzscheme
  (provide (rename :lcons lcons) lcar lcdr)

  (define-struct lcons (hd tl))

  (define-syntax (:lcons stx)
    (syntax-case stx ()
      [(_ hd-exp tl-exp)
       (syntax (make-lcons
```

```
                (delay hd-exp)
                (delay tl-exp)))]))

  (define (lcar lcons) (force (lcons-hd lcons)))
  (define (lcdr lcons) (force (lcons-tl lcons))))
```

Then, in this program:

```
(define (lmap f l)
  (lcons
   (f (lcar l))
   (lmap f (lcdr l))))

(define all-nums (lcons 1 (lmap add1 all-nums)))
```

the list `all-nums` is bound to an infinite list of ascending numbers.

For more examples, see the **htdp** directory of the **teachpack** directory in the PLT installation.

4.2 Tools

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

4.3 Environment Variables

This section lists the environment variables that affect DrScheme's behavior. See the MzScheme manual, §15.4 in *PLT MzScheme: Language Manual* for general information about environment variables.

- **PLTNOTOOLS** When this environment variable is set, DrScheme doesn't load any tools.
- **PLTONLYTOOL** When this environment variable is set, DrScheme only loads the tools in the collection named by the value of the environment variable. If the variable is bound to a parenthesized list of collections, only the tools in those collections are loaded (The contents of the environment variable are `read` and expected to be a single symbol or a list of symbols).
- **PLTDRCM** When this environment variable is set, DrScheme installs the compilation manager before starting up, which means that the .zo files are automatically kept up to date, as DrScheme's (or a tools) source is modified.

  If the variable is set to **trace** then CM's output is traced, using the `manager-trace-handler` procedure from the CM library.
- **PLTHDCM** When this environment variable is set, Help Desk installs the compilation manager before starting up (but only in standalone mode), which means that the .zo files are automatically kept up to date, as Help Desk's source is modified.

  If the variable is set to **trace** then CM's output is traced, using the `manager-trace-handler` procedure from the CM library.
- **PLTDRDEBUG** When this environment variable is set, DrScheme starts up with errortrace enabled. If the variable is set to **profile**, DrScheme also records profiling information about itself.
- **PLTDRBREAK** When this environment variable is set, DrScheme creates a window with a break button, during startup. Clicking the button breaks DrScheme's eventspace's main thread. This works well in combination with PLTDRDEBUG since the source locations are reported for the breaks.

- **PLTDRTESTS** When this environment variable is set, DrScheme installs a special button in the button bar that starts the test suite. (This is only available in the source distribution)
- **PLTSTRINGCONSTANTS** When this environment variable is set, DrScheme prints out the string constants that have not yet been translated. If it is set to a particular language (corresponding to one of the files in plt/collects/string-constants) it only shows the unset string constants matching that language.

  This environment variable must be set when .zo files are made. To ensure that you see its output properly, run setup-plt with the -c option, set the environment variable, and then run setup-plt again.

5. Frequently Asked Questions

5.1 Supported Operating Systems and Installation

Where can I get DrScheme and/or documentation?

DrScheme is available for download at

`http://www.drscheme.org/`

Some documentation is provided with DrScheme, accessible through Help Desk. Other documentation is provided online in HTML format and is also available for download in Adobe PDF format at

`http://download.plt-scheme.org/doc/`

How much does DrScheme cost?

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

What operating systems are supported for DrScheme?

Windows (95 and up), Mac OS X, 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 128 MB of RAM.

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

MrEd is PLT's raw graphical Scheme implementation (used to run DrScheme itself). MrEd provides a minimal read-eval-print loop, but MrEd does not provide DrScheme's various 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. MzScheme distributions can be downloaded through

`http://www.plt-scheme.org/software/mzscheme/`

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 Mac OS X, the distribution is a disk image containing a folder to copy to your disk. For Unix/X, the distribution is a self-extracting shell program; running it will unpack and install the archive. In all cases, the final download page provides detailed, platform-specific installation instructions.

How large is the distribution archive?

Distribution archives vary in sizes. From the download page, select your platform and click the download button; the next screen will have download links as well as the file size at the top of the page.

How much disk space does DrScheme consume?

Around 70 MB in its normal configuration, not including the optional documentation.

5.2 Using DrScheme

How do I find general help for DrScheme?

Select Help Desk in DrScheme's Help menu.

How do I run MrFlow, DrScheme's program analyzer (and the successor to MrSpidey)?

MrFlow is distributed separately from the standard DrScheme distribution. Download MrFlow from

`http://www.plt-scheme.org/software/mrflow/`

What happened to the Analyze button?

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

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. See also the Keybindings menu item in the Edit menu.

Can I change the key bindings in DrScheme?

See "Defining Custom Shortcuts" in *PLT DrScheme: Development Environment Manual* for information on customizing keybindings.

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 Run button. To suppress all but the first warning, see the Warnings tab in the Preferences dialog.

Why can't I type in the interaction window before 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.

Why doesn't `let` work? (or `letrec` or `lambda` or `set!`, etc)

DrScheme supports many different variants of Scheme, and you may have selected a language such as the Beginning Student teaching language. To remedy this, select Choose Language... from the Language menu, and then choose a language such as Standard or Pretty Big.

Is there a DrScheme compiler?

Technically, DrScheme is a compiler as well as an interpreter. Each time the user loads a program or enters expression in the interactions window, DrScheme compiles (to native code, on popular platforms) and then runs the program or expression.

Can I produce stand-alone executables from Scheme code?

See "Executables" in *PLT DrScheme: Development Environment Manual*.

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 test 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, and DrScheme will read it correctly on the destination platform.

5.3 Memory and Performance

Does DrScheme really require at least 128 MB of memory?

Yes.

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. For many languages, these checks can be disabled by un-checking Debugging in the details portion of the language-selection dialog.

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.

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 you need to, submit a bug report using the form available from the home page of Help Desk. Alternatively, you may submit a bug report using the Web at

`http://bugs.plt-scheme.org/`

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@plt-scheme.org`

License

GNU Library General Public License

Version 2, June 1991

Copyright (C) 1991 Free Software Foundation, Inc.

675 Mass Ave, Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries

themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

GNU LIBRARY GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

   The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

   "Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) The modified work must itself be a software library.
   b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
   c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
   d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.
      (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

   In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

   If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

   Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
   b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
   c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
   d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

   For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
   b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

    If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

    It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

    This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Index

~, 18
(define ...) button, 2
.LOCK.org.plt-scheme.prefs.ss, 23
.LOCK.plt-prefs.ss, 23
.bak, 18, 22
.plt-scheme, 23
.sch, 22
.scm, 22
.ss, 22
_LOCKplt-prefs.ss, 23
> prompt, 4

About DrScheme... menu item, 17
Add Teachpack... menu item, 15
Add User-defined Keybindings... menu item, 14
Advanced Student language, 5
alpha renaming, 3
Ask about clearing test coverage preference, 18
Ask before changing save format preference, 18
Auto-save files preference, 17
automatic parenthesis, 4
Automatically open interactions window when running a program preference, 18
Automatically print to PostScript file preference, 18
autosave files, 22

backup files, 22
Backup files preference, 18
Beginning Student language, 5
Beginning Student language with List Abbreviations, 5
Break button, 3
Break menu item, 15
Bring Frame to Front... menu item, 17

changing keybindings, 28
changing parenthesis as you type, 4
Check for newer PLT Scheme versions preference, 18
Check Syntax, 3
Choose Language... menu item, 15
Clear All Teachpacks menu item, 15
Clear Error Highlight menu item, 15
Clear menu item, 14
Close menu item, 13
Collapse menu item, 15
Color syntax interactively preference, 18
Comment Out with a Box menu item, 16
Comment Out with Semicolons menu item, 16
compiler, 29
configuration files, 23
constructor-style output, 9
Copy menu item, 14
Correct parens preference, 18
Count column numbers from one preference, 18
Create Executable... menu item, 15
Cut menu item, 14

Debug button, 3
debugger, 3
defaults
    site-specific, 23
defaults, 23
definitions window, 2
Delete menu item, 14
Disable All Test Cases menu item, 16
disk requirements, 28
`display`, 10
Display line numbers in buffer; not character offsets preference, 18
distribution archives, 9
documentation
    downloading, 27
DrScheme
    Environment Variables, 25
DrScheme Teachpacks, 24

editors, 4
Emacs keybindings, 19
Enable keybindings in menus preference, 18
error highlighting, 5
evaluating expressions, 4
execution speed, 29

file extensions, 22
filename button, 2
Find Again menu item, 14
Find menu item, 18
Find... menu item, 14
Flash paren match preference, 18
flashing parenthesis matches, 4
font preference, 17
formatting Scheme code, 4
frequently asked questions, 27

graphical interface, 2
    details, 13
gray highlight regions, 4

Help Desk, 28

Help Desk menu item, 17
Hide Definitions menu item, 14
Hide Interactions menu item, 14
Hide Module Browser menu item, 15
Hide Profile menu item, 15
Hide Program Contour menu item, 14
Hide Toolbar menu item, 15
Hide Tracing menu item, 15
Highlight between matching parens preference, 18

I/O, 10
Indenting preferences, 17
indenting Scheme code, 4
Insert λ menu item, 16
Insert Comment Box menu item, 16
Insert Fraction... menu item, 16
Insert Image... menu item, 16
Insert Java Comment Box menu item, 16
Insert Java Examples menu item, 16
Insert Java Interactions Box menu item, 16
Insert Large Letters... menu item, 16
Insert Pict Box menu item, 17
Insert Scheme Box menu item, 16
Insert Scheme Splice Box menu item, 16
Insert Test Case menu item, 17
Insert XML Box menu item, 16
Install PLT File... menu item, 13
installation instructions, 27
Interact with DrScheme in English menu item, 17
interactions window, 2, 4
Intermediate Student language, 5
Intermediate Student with Lambda language, 5

keybindings, 19
    A-C-down, 20
    A-C-left, 20
    A-C-right, 20
    A-C-up, 20
    b, 21
    C-+, 20
    C-_, 20
    C-a, 19
    C-b, 19
    C-c C-l, 21
    C-c C-o, 21
    C-d, 20
    C-e, 20
    C-f, 19
    C-h, 20
    C-k, 20
    C-n, 20
    C-p, 20
    C-r, 21
    C-s, 21
    C-t, 20
    C-v, 19
    C-w, 20
    C-x C-s, 21
    C-x C-w, 21
    C-x u, 20
    C-y, 20
    copy selection to clipboard, 20
    cut complete sexpression, 21
    delete backward one character, 20
    delete backward one word, 20
    delete forward one character, 20
    delete forward one S-expression, 20
    delete forward one word, 20
    delete forward to end of line, 20
    delete selection to clipboard (cut), 20
    F5, 21
    insert-let-binding: wraps a let around the sexpression following the insertion point and puts a printf in at that point (useful for debugging)., 21
    M-<, 19
    M->, 19
    M-(, 21
    M-[, 21
    M-{, 21
    M-b, 19
    M-C-b, 20
    M-C-d, 20
    M-C-down, 20
    M-C-f, 20
    M-C-k, 20, 21
    M-C-left, 20
    M-C-m, 21
    M-C-p, 20
    M-C-right, 20
    M-C-SPACE, 20
    M-C-t, 21
    M-C-u, 20
    M-C-up, 20
    M-d, 20
    M-DEL, 20
    M-f, 19
    M-n, 21
    M-o, 21
    M-p, 21
    M-S-L, 21
    M-t, 20
    M-v, 19
    M-w, 20
    match parentheses backward, 20
    move backward one character, 19
    move backward one page, 19

move backward one S-expression, 20
move backward one word, 19
move backwards to the nearest editor box, 20
move down into a nested S-expression, 20
move down into an embedded editor, 20
move forward one character, 19
move forward one page, 19
move forward one S-expression, 20
move forward one word, 19
move forward to the nearest editor box, 20
move to beginning of file, 19
move to beginning of line (left), 19
move to end of file, 19
move to end of line (right), 20
move to next line (down), 20
move to previous line (up), 20
move up out of an embedded editor, 20
move up out of an S-expression, 20
move-sexp-out: the sexpression following the insertion point is put in place of its containing sexpression, 21
paste from clipboard (yank), 20
redo, 20
Run, 21
save file, 21
save file under new name, 21
search for string backward, 21
search for string forward, 21
select forward S-expression, 20
toggle dark green marking of matching parenthesis, 21
toggle overwrite mode, 21
transpose characters, 20
transpose sexpressions, 21
transpose words, 20
undo, 20
wrap selection in curly brackets, 21
wrap selection in parentheses, 21
wrap selection in square brackets, 21
wrap selection in `(lambda () ...)` and put the insertion point in the arglist of the lambda, 21
Keybindings menu item, 14
keyboard shortcuts, 19
Kill menu item, 15

language levels, *see* languages
languages, 5
changing, 5
extending, 24
launcher executables, 8
license, 27
Log Definitions and Interactions... menu item, 13

Map delete to backspace preference, 18
memory requirements, 27
menu items, 13
module browser, 14
Module Browser... menu item, 15
`module` language, 6
Most Recent Window menu item, 17
MrFlow, 28
MrSpidey, 28
multimedia file format, 22

New menu item, 13
Number of recent items preference, 17
numbers
printing, 8

Only warn once when executions and interactions are not synchronized preference, 18
Open files in separate tabs (not separate windows) preference, 18
Open Recent menu item, 13
Open... menu item, 13
org.plt-scheme.prefs.ss, 23
output format, 9
overwrite mode, 21

Paste menu item, 14
plain-text file format, 22
PLT Graphical (MrEd) language, 6
PLT Pretty Big language, 6
PLT Scheme, 23
PLT Textual (MzScheme) language, 6
plt-prefs.ss, 23
PLTDRBREAK, 25
PLTDRCM, 25
PLTDRDEBUG, 25
PLTDRTESTS, 26
PLTHDCM, 25
PLTNOTOOLS, 25
PLTONLYTOOL, 25
PLTSTRINGCONSTANTS, 26
preference files, 23
preferences, 17
site-specific, 23
Preferences... menu item, 14
Print Definitions... menu item, 13
Print Interactions... menu item, 13
printing format, 9
proxy, 19

quasiquote-style output, 9

R5RS, 6
`read`, 10

`read-char`, 10
read-eval-print loop, 9
recycling icon, 2
Redo menu item, 14
Reindent All menu item, 16
Reindent menu item, 16
Related Web Sites menu item, 17
Replace & Find Again menu item, 14
Reuse existing frames when opening new files preference, 18
Revert menu item, 13
Run button, 5
Run menu item, 15

Save button, 3
Save Definitions As Text... menu item, 13
Save Definitions As... menu item, 13
Save Definitions menu item, 13
Save Interactions As Text... menu item, 13
Save Interactions As... menu item, 13
Save Interactions menu item, 13
Save Other menu item, 13
Search in Files... menu item, 13
Select All menu item, 14
Show Active Keybindings menu item, 14
Show Definitions menu item, 14
Show Interactions menu item, 14
Show Module Browser menu item, 14
Show Profile menu item, 15
Show Program Contour menu item, 14
Show status-line preference, 18
Show Toolbar menu item, 15
Show Tracing menu item, 15
Split menu item, 15
stand-alone executables, 8
status line, 2
Step button, 3
stepper, 3
storage requirements, 28
supported platforms, 27

tabbed editing, 5
tabs, 5
tail calls, 3
Teachpacks
    implementing, 24
The Stepper, 3
Tool Web Sites menu item, 17
tools, 24

Uncomment menu item, 16
Undo menu item, 14
Use separate dialog for searching preference, 18

Verify exit preference, 18

Wrap Text menu item, 14
Wrap words in editor buffers preference, 18
`write`, 10

XML, 11

yellow and black messages, 28