PLT DrScheme: Programming Environment Manual

PLT (scheme@plt-scheme.org)

Version 201
July 2002

Copyright notice

Send us your Web links

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

Thanks

Thanks to Tim Hanson, Philippe Meunier, Francisco Solsona, and Reini Urban for their help translating DrScheme's GUI to other languages.

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

This manual was typeset on August 1, 2002.

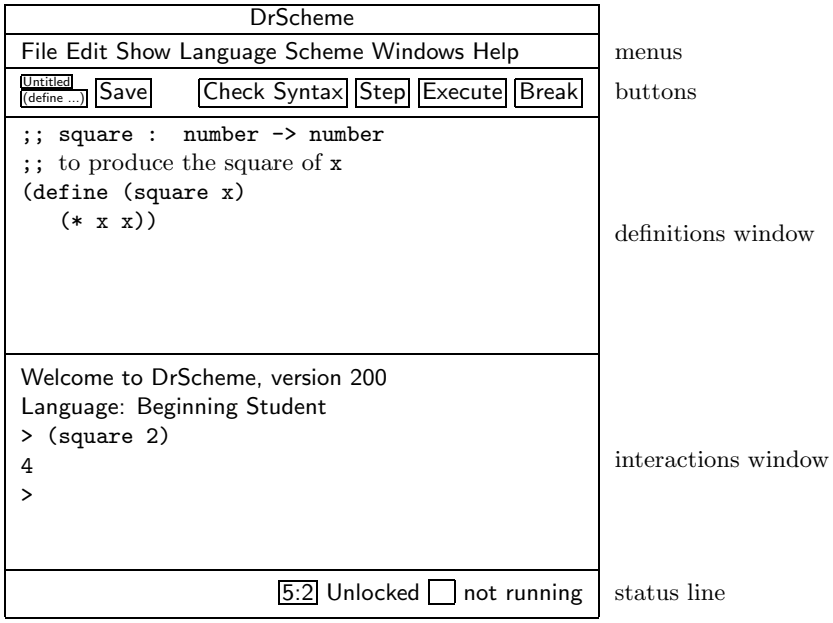
Contents

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

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

2.1 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 opens file starting in the corresponding directory.

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

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

- **Syntactic Highlighting** Keywords, bound variables, unbound variables and constants are all highlighted with font and 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 occurrance of the variable.

  Additionally, control or right-button mouse clicking on a variable activates a popup menu that lets you jump from binding location to bound location and vice versa, rename the variable, or tack the arrows so they do not disappear.
- **Require Annotations** Control-clicking or right-button clicking (depending on the platform DrScheme runs on) on the argument to **require** activates a popup menu that lets you open the file that contains the **require**d 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.

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

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

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

2.2 The Editor

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

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

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

2.3 The Interactions Window

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

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

If the expression following the current prompt is incomplete, then DrScheme will not try to evaluate it. In that case, hitting Enter or Return produces a new, auto-indented line.

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

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

2.4 Errors

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

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

2.5 Languages

DrScheme supports multiple dialects of Scheme. The name of the current evaluation language always appears in in the top of the interactions window. To choose a different language, select the Language|Choose Language... menu item. After changing the language, click Execute 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.
- `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. (Each option corresponds to one of the lines in the language table, but only a few of the lines in the figure have an option in the dialog.) Whenever the selected options do not match the default language specification, a Custom indicator appears next to the language-selection control at the top of the dialog.

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 familar 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`.

[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**.

- *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.

- *Top-level required at function call position* — In the Beginning Student languages, procedure calls must be of the form `(`*top-level-identifier* `...)`. This restriction helps detect confusing misuses of parentheses, such as `(`x`)` where x is a function argument. DrScheme can detect such mistakes syntactically because 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.

- **quote** *works only on symbols,* **quasiquote** *diallowed* — 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 diallowed* — In the Beginning Student through Advanced Student languages, a delimitted 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.

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 Section 2.7.1.
- *Quasiquote-style output* — See Section 2.7.2.
- *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 gives you 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 only controls the initial display of a number. Right-clicking or control-clicking 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)).
  ```

2.6 Executables

DrScheme's Create Executable... menu 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 Execute, just to make sure that the program is working as you expect. Then, choose the Create Executable... menu item from the Scheme menu. Choose a place to save the executable. You will be able to start the saved executable in the same way that you start any other program on your computer.

An executable created by Create Executable... is either a *launcher executable* or a *stand-alone executable*, and it uses either a *graphical (MrEd)* or *textual (MzScheme)* 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* tends to be small, and it 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* tends to be large, because it embeds a copy of your program at the time that it is created, as well as any library that your code uses. When the executable is started, it uses the embedded copies and does not need your original source file or your DrScheme installation. It may, however, require DLLs or framework libraries installed on your machine, depending on your operating system:
  - Windows — The executable requires the following DLLs: **libmzsch[vers].dll**, **libmzgc[vers].dll**, and (for executables using the MrEd engine) **libmred[vers].dll**, where **[vers]** is based on the current version number. These DLLs are normally installed in the system directory.
  - Mac OS X — The executable requires the **PLT_MzScheme** framework, which is normally installed in **/Library/Frameworks**. When using the MrEd engine, the executable also requires the **PLT_MrEd** framework from the same location.

  To move the "stand-alone" executable to another machine, the DLLs or frameworks that it uses must also be copied to the other machine.

DrScheme may also ask you to choose a base executable. The choices are MrEd and MzScheme. MzScheme exectables are smaller, but have no graphical libraries. Also, under Mac OS X, executables created with MzScheme as the base can only be run from the commandline. Using the MrEd base executable means that your executable can also be launched from the finder.

Tip: Disable debugging in the language dialog before creating your launcher. 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 click the No debugging or profiling check box, if it is available.

2.7 Printed Results

2.7.1 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`.

2.7.2 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

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`

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))))

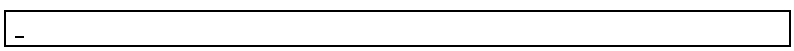
2.8 Input and Output

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

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

```
> (read)
```

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



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

as follows:

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

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

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

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

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

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

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

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

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

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

2.9 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 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 on the top of a Scheme box opens a menu to toggle the box between a Scheme box and a Scheme splice box.

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.
- Open URL... opens a dialog for choosing a Uniform Resource Locator (URL) to open in a new Help Desk window.
- Revert re-loads the file that is currently in the definitions window. All changes since the file was last saved will be lost.
- Save Definitions saves the program in the definitions window. If the program has never been saved before, a save-file dialog appears.
- Save Definitions As... opens a save-file dialog for choosing a destination file to save the program in the definitions window. Subsequent saves write to the newly-selected file.
- Save Other contains these sub-items
  - Save Definitions As Text... is like Save Definitions As..., but the file is saved in plain-text format (see Section 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... is like Save Interactions As..., but the file is saved in plain-text format (see Section 3.4.1). Subsequent saves are write in plain-text format.
- Print Definitions... opens a dialog for printing the current program in the definitions window.
- Print Interactions... opens a dialog for printing the contents of the interactions window.
- Close closes this DrScheme window. If this window is the only open DrScheme window, DrScheme quits.
- Quit or Exit exits DrScheme.

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 maintains a history of Undo actions, so multiple Redo operations can reverse multiple Undo actions.
- Cut copies the selected text to the clipboard and deletes it from the window.
- Copy copies the selected text to the clipboard.
- Paste pastes the current clipboard contents into the window.
- Delete 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.
- 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.
- Keybindings

  Shows all of the keybindings available in the current window.
- Preferences... opens the preferences dialog. See section 3.2.

3.1.3 Show

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

- Show Interactions shows interactions window.
- Hide Interactions hides interactions window.
- Show Definitions shows the definitions window.
- Hide Definitions hides the definitions window.
- Show 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 Contour hides the contour window.
- Show Profile shows the current profiling report. This menu is only useful if you have enabled profiling in the Language dialog. Profiling does not apply to all languages. When it does, the checkbox that enables it is in the details portion of the dialog.
- Hide Profile hides any profiling information currently displayed in the DrScheme window.
- 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 executed, the interactions window is made visible if it is invisible.

3.1.4 Language

- Choose Language opens a dialog for selecting the current evaluation language. Click Execute to make the language active in the interactions window. See section 2.5 for more information about the languages.
- Add Teachpack... opens a find-file dialog for choosing a teachpack to extend the current language. Click Execute to make the teachpack available in the interactions windows. See Section 4 for information on creating teachpacks.
- Clear All Teachpacks clears all of the current teachpacks. Click Execute to clear the teachpack from the interactions window.

In addition to the above menus, there is a menu item for each teachpack that clears only it.

3.1.5 Scheme

- Execute resets the interactions window and executes the program in the definitions window.
- Break breaks the current evaluation.
- Kill terminates the current evaluation.
- Create Executable creates a separate launcher for running your program. See section 2.6 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 on the box for that module (control-click under Mac OS).
- 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 puts ";" characters at each of the the beginning of each selected line of text.
- Uncomment removes all ";" characters at the start of each selected line of text. Uncommenting only removes a ";" if it appears at the start of a line and it only removes the first ";" on each line.

3.1.6 Special

- Insert Text Box inserts a box into the window. The text box may contain arbitrary text, and it is treated as a value, like a number or symbol.
- Insert Pasteboard Box inserts a graphics box. The box is treated as a value.
- Insert Image... opens a find-file dialog for selecting an image file in GIF, BMP, XBM, or XPM format. The image is treated as a value.
- Insert Fraction... opens a dialog prompting for a mixed-notation fraction and inserts that into the current editor.
- Insert Large Letters... Opens a dialog asking for a line of text and inserts a large version of that (made with semicolons and spaces) into the current editor.
- Insert Scheme Box

  Inserts a Scheme box. These boxes contain Scheme code and are meant to be inside XML boxes. See also Section 2.9.
- Insert Scheme Splice Box

  Inserts a Scheme splice box. These boxes are intended to be inside XML boxes. The Scheme expression inside must evaluate to a list and that list is spliced into the containing XML box. See also Section 2.9.
- Insert XML Box

  This menu item inserts an XML box. XML boxes must contain XML's concrete syntax and they evaluate to x-expressions, which are s-expressions representing the XML expression. See also Section 2.9.

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.
- Next Window

  Brings the next window to the front.
- Previous Window

  Brings the previous window to the front.

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.
- Check for Updates...

  Opens an internet connection to check if there have been any updates to various installed DrScheme packages.

- Related Web Sites

  This submenu has a list of related web sites. Choosing one of them sends a message to an external browser to visit the web site.

- Is English your mother tongue?

  This and the related menu items at the end of the Help menu switch DrScheme's interface (natural) language.

3.2 Preferences

The preferences dialog comprises several panels:

- Font

  This panel controls the main font used by DrScheme.

- Indenting

  This panel controls which keywords DrScheme recognizes for indenting, and how each keyword is treated.

- Editing
  - Auto-save files — If checked, the editor generates autosave files (see Section 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 line and column numbers from one — If checked, the status line's line: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.
  - 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.

- Warnings
  - Ask before changing save format — If checked, DrScheme consults the user before saving a file in non-text format (see Section 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 Execute. Otherwise, the warning appears on every interaction.

- 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 flashing caret is next to a parenthesis.
  - Correct parens — If checked, the editor automatically converts a typed ")" to "]" to match "[", or it converts a typed "]" to ")" to match "(".
  - Flash paren match — If checked, typing a closing parenthesis, square bracket, or quotation mark flashes the matching open parenthesis/bracket/quote.

- HTTP Proxy

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

- 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.

- Check Syntax

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

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, 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 down into an embedded editor to the left
- M-C-right move down into an embedded editor to the right
- M-C-up move up from an embedded editor

3.3.2 Editing Operations

- C-d delete forward one character
- C-h delete backward one character
- M-d delete forward one word
- M-DEL delete backward one word
- C-k delete forward to end of line
- M-C-k delete forward one S-expression
- M-w copy selection to clipboard
- C-w delete selection to clipboard (cut)
- C-y paste from clipboard (yank)

- C-t transpose characters
- M-t transpose words
- M-C-t transpose sexpressions
- M-C-m toggle dark green marking of matching parenthesis
- M-C-k cut complete sexpression
- C-_ undo
- C-+ redo
- C-x u undo
- 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 Execute

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

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 saves a program file in two different formats:

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

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

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

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

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

  Windows NT, XP, 2000: When DrScheme is launched under Windows NT, XP, or 2000 and HOMEDRIVE and HOMEPATH are not set, Windows automatically sets the variables to indicate the root directory of the main disk. Therefore, when HOMEDRIVE and HOMEPATH are not set, the preferences file **plt-prefs.ss** is saved in the root directory of the main disk.
- Under Mac OS X, preferences are stored in **.plt-prefs.ss** in the user's preferences folder. Under Mac OS Classic, preferences are stored in **plt-prefs.ss** in the system preferences folder.

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

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 have no preferences will get the preference settings you chose.

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, and the Analyze button starts the MrSpidey tool.

4.1 Teachpacks

Teachpacks are designed to supplement student programs with code that is beyond the teaching languages (Beginning Student, Intermediate Student, Advanced Student). 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 primtive operation to the user, respectively.

As an example 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 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*.

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 Library General Public License in the distribution for details.

What operating systems are supported for DrScheme?

Windows (95 and up), Mac OS (9 and 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 32 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 execute DrScheme). 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/`

Does DrScheme run under Mac OS versions earlier than 9?

No.

Does DrScheme run under DOS or Windows 3.1?

No.

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

No.

How do I install DrScheme?

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

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

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

More details about this problem are available from the Wise Solutions KnowledgeBase at

`http://www.wise.com/displayArticle.asp?articleno=1034`

How large is the distribution archive?

An average archive is around 4 MB. The Linux RPM distribution is somewhat over 5.0 MB.

How much disk space does DrScheme consume?

Around 20 MB in its normal configuration, not including the optional documentation. Some disk space (about 5MB) can be saved by deleting all files with the suffix **.zo**. DrScheme does not need the **.zo** files, but it starts up more slowly without them (so keep them unless you really need the disk space).

5.2 Using DrScheme

How do I find general help for DrScheme?

Select Help Desk in DrScheme's Help menu.

How do I run 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.

Can I change the key bindings in DrScheme?

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

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

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

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

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

Is there a DrScheme compiler?

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 and then executes the program or expression.

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

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

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

Can I produce stand-alone executables from Scheme code?

To produce a standalone executable that only works for your particular installation of DrScheme, see section 2.6.

In addition, the **mzc** compiler can be used to produce stand-alone executables. See the **mzc** documentation for more information.

Can files saved in DrScheme be transferred between platforms?

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

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

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

5.3 Memory and Performance

Does DrScheme really require at least 32 MB of memory?

Yes.

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 most languages, these checks can be disabled by un-checking Debugging in the detail part 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.

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

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

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

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

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

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

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

First, read this section to make sure your problem does not have a standard answer. If 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/`

If you do not have access to a web browser, as a last resort send mail to

`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`

Index