PLT Tools: DrScheme Extension Manual

Robert Bruce Findler (robby@plt-scheme.org)

371
Released August 2007

Copyright notice

Send us your Web links

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

Contents

1. This Manual

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

[build date: August 18, 2007]

1.1 Thanks

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

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 August 18, 2007.

2. Implementing DrScheme Tools

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

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

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

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

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

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

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

tool-urls (*listof* (*union* #f *string*[*url*]))

The *tools* field names a list of tools in this collection. Each tool is specified as a collection path, relative to the collection where the **info.ss** file resides. As an example, if there is only one tool named **tool.ss**, this suffices:

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

If the *tool-icons* or *tool-names* fields are present, they must be the same length as *tools*. The *tool-icons* specifies the path to an icon for each tool and the name of each tool. If it is `#f`, no tool is shown. If it is a relative pathname, it must refer to a bitmap and if it is a list of strings, it is treated the same as the arguments to *lib*, inside `require`.

This bitmap and the name show up in the about box, Help Desk's bug report form, and the splash screen as the tool is loaded at DrScheme's startup.

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

The `drscheme:tool-exports^` signature contains two names: *phase1* and *phase2*. These names must be bound to thunks. After all of the tools are loaded, all of the `phase1` functions are called and then all of the `phase2` functions are called. Certain primitives can only be called during the dynamic extent of those calls.