NAMED READTABLES MANUAL

Contents

[in package EDITOR-HINTS.NAMED-READTABLES with nicknames NAMED-READTABLES]

- **[system]** **"named-readtables"**

```
- _Version:_ 0.9
- _Description:_ Library that creates a namespace for readtables akin
    to the namespace of packages.
- _Licence:_ BSD, see LICENSE
- _Author:_ Tobias C. Rittweiler <trittweiler@common-lisp.net>
- _Maintainer:_ Gábor Melis <mega@retes.hu>
- _Mailto:_ [mega@retes.hu](mailto:mega@retes.hu)
- _Homepage:_ <http://melisgl.github.io/named-readtables>
- _Bug tracker:_ <https://github.com/melisgl/named-readtables/issues>
- _Source control:_ [GIT](https://github.com/melisgl/named-readtables.git)
- *Depends on:* mgl-pax-bootstrap
```

1 Introduction

Named-Readtables is a library that provides a namespace for readtables akin to the already-existing namespace of packages. In particular:

- you can associate readtables with names, and retrieve readtables by names;
- you can associate source files with readtable names, and be sure that the right readtable is active when compiling/loading the file;
- similiarly, your development environment now has a chance to automatically determine what readtable should be active while processing source forms on interactive commands. (E.g. think of `C-c C-c` in Slime (yet to be done))

It follows that Named-Readtables is a facility for using readtables in a localized way.

Additionally, it also attempts to become a facility for using readtables in a *modular* way. In particular:

- it provides a macro to specify the content of a readtable at a glance;
- it makes it possible to use multiple inheritance between readtables.

1.1 Links

Here is the official repository and the HTML documentation for the latest version.

1.2 Acknowledgements

Thanks to Robert Goldman for making me want to write this library.

Thanks to Stephen Compall, Ariel Badichi, David Lichteblau, Bart Botta, David Crawford, and Pascal Costanza for being early adopters, providing comments and bugfixes.

2 Overview

2.1 Notes on the API

The API heavily imitates the API of packages. This has the nice property that any experienced Common Lisper will take it up without effort.

```
DEFREADTABLE               -   DEFPACKAGE

IN-READTABLE               -   IN-PACKAGE

MERGE-READTABLES-INTO      -   USE-PACKAGE

MAKE-READTABLE             -   MAKE-PACKAGE

UNREGISTER-READTABLE       -   DELETE-PACKAGE

RENAME-READTABLE           -   RENAME-PACKAGE

FIND-READTABLE             -   FIND-PACKAGE

READTABLE-NAME             -   PACKAGE-NAME

LIST-ALL-NAMED-READTABLES  -   LIST-ALL-PACKAGES
```

2.2 Important API idiosyncrasies

There are three major differences between the API of Named-Readtables, and the API of packages.

1. Readtable names are symbols not strings.

   Time has shown that the fact that packages are named by strings causes severe headache because of the potential of package names colliding with each other.

   Hence, readtables are named by symbols lest to make the situation worse than it already is. Consequently, readtables named `cl-oracle:sql-syntax` and `cl-mysql:sql-syntax` can happily coexist next to each other. Or, taken to an extreme, `scheme:syntax` and `elisp:syntax`.

   If, for example to duly signify the importance of your cool readtable hack, you really think it deserves a global name, you can always resort to keywords.

2. The inheritance is resolved statically, not dynamically.

   A package that uses another package will have access to all the other package's exported symbols, even to those that will be added after its definition. I.e. the inheritance is resolved at run-time, that is dynamically.

   Unfortunately, we cannot do the same for readtables in a portable manner.

   Therefore, we do not talk about "using" another readtable but about "merging" the other readtable's definition into the readtable we are going to define. I.e. the inheritance is resolved once at definition time, that is statically.

   (Such merging can more or less be implemented portably albeit at a certain cost. Most of the time, this cost manifests itself at the time a readtable is defined, i.e. once at compile-time, so it may not bother you. Nonetheless, we provide extra support for Sbcl, ClozureCL, and AllegroCL at the moment. Patches for your implementation of choice are welcome, of course.)

3. `defreadtable` does not have compile-time effects.

   If you define a package via `defpackage`, you can make that package the currently active package for the subsequent compilation of the same file via `in-package`. The same is, however, not true for `defreadtable` and `in-readtable` for the following reason:

   It's unlikely that the need for special reader-macros arises for a problem which can be solved in just one file. Most often, you're going to define the reader macro functions, and set up the corresponding readtable in an extra file.

   If `defreadtable` had compile-time effects, you'd have to wrap each definition of a reader-macro function in an `eval-when` to make its definition available at compile-time. Because that's simply not the common case, `defreadtable` does not have a compile-time effect.

   If you want to use a readtable within the same file as its definition, wrap the `defreadtable` and the reader-macro function definitions in an explicit `eval-when`.

2.3 Preregistered Readtables

- `nil`, `:standard`, and `:common-lisp` designate the *standard readtable*.
- `:modern` designates a *case-preserving standard-readtable*.
- `:current` designates the *current readtable*.

2.4 Examples

```
(defreadtable elisp:syntax
   (:merge :standard)
   (:macro-char #\? #'elisp::read-character-literal t)
   (:macro-char #\[ #'elisp::read-vector-literal t)
   ...
   (:case :preserve))

(defreadtable scheme:syntax
   (:merge :standard)
   (:macro-char #\[ #'(lambda (stream char)
                        (read-delimited-list #\] stream)))
   (:macro-char #\# :dispatch)
   (:dispatch-macro-char #\# #\t #'scheme::read-#t)
   (:dispatch-macro-char #\# #\f #'scheme::read-#f)
   ...
   (:case :preserve))

(in-readtable elisp:syntax)

...

(in-readtable scheme:syntax)

...
```

3 Reference

- **[macro]** **defreadtable** *name &body options*

  Define a new named readtable, whose name is given by the symbol `name`. Or, if a readtable is already registered under that name, redefine that one.

  The readtable can be populated using the following `options`:

  - If the first element of `options` is a string then it is associated with the readtable as in `(setf (documentation name 'readtable) docstring)`.
  - `(:merge readtable-designators+)`

    Merge the macro character definitions from the readtables designated into the new readtable being defined as per `merge-readtables-into`. The copied options

are `:dispatch-macro-char`, `:macro-char` and `:syntax-from`, but not `readtable-case`.

If no `:merge` clause is given, an empty readtable is used. See `make-readtable`.

- `(:fuse readtable-designators+)`

  Like `:merge` except:

  Error conditions of type `reader-macro-conflict` that are signaled during the merge operation will be silently *continued*. It follows that reader macros in earlier entries will be overwritten by later ones. For backward compatibility, `:fuze` is accepted as an alias of `:fuse`.

- `(:dispatch-macro-char macro-char sub-char function)`

  Define a new sub character `sub-char` for the dispatching macro character `macro-char`, per `set-dispatch-macro-character`. You probably have to define `macro-char` as a dispatching macro character by the following option first.

- `(:macro-char macro-char function [non-terminating-p])`

  Define a new macro character in the readtable, per `set-macro-character`. If `function` is the keyword `:dispatch`, `macro-char` is made a dispatching macro character, per `make-dispatch-macro-character`.

- `(:syntax-from from-readtable-designator from-char to-char)`

  Set the character syntax of `to-char` in the readtable being defined to the same syntax as `from-char` as per `set-syntax-from-char`.

- `(:case case-mode)`

  Defines the *case sensitivity mode* of the resulting readtable.

Any number of option clauses may appear. The options are grouped by their type, but in each group the order the options appeared textually is preserved. The following groups exist and are executed in the following order: `:merge` and `:fuse` (one group), `:case`, `:macro-char` and `:dispatch-macro-char` (one group), finally `:syntax-from`.

Notes:

The readtable is defined at load-time. If you want to have it available at compilation time -- say to use its reader-macros in the same file as its definition -- you have to wrap the `defreadtable` form in an explicit `eval-when`.

On redefinition, the target readtable is made empty first before it's refilled according to the clauses.

`nil`, `:standard`, `:common-lisp`, `:modern`, and `:current` are preregistered readtable names.

- **[macro]** **in-readtable** *name*

  Set `*readtable*` to the readtable referred to by the symbol `name` and return the readtable. This may signal `readtable-does-not-exist`.

- Everything `in-readtable` does is also performed at compile time if the call appears as a top level form.
- The effects of `in-readtable` are file-local since both `compile-file` and `load` rebind `*readtable*`.

- **[function]** **make-readtable** *&optional (name nil name-supplied-p) &key merge*

  Creates and returns a new readtable under the specified `name`.

  `merge` takes a list of `named-readtable-designator`s and specifies the readtables the new readtable is created from. (See the `:merge` clause of `defreadtable` for details.)

  If `merge` is `nil`, an empty readtable is used instead.

  If `name` is not given, an anonymous empty readtable is returned.

  Notes:

  An empty readtable is a readtable where each character's syntax is the same as in the *standard readtable* except that each macro character has been made a constituent. Basically: whitespace stays whitespace, everything else is constituent.

- **[function]** **merge-readtables-into** *result-readtable &rest named-readtables*

  Copy macro character definitions of each readtable in `named-readtables` into `result-readtable`.

  If a macro character appears in more than one of the readtables, i.e. if a conflict is discovered during the merge, an error of type `reader-macro-conflict` is signaled.

  The copied options are `:dispatch-macro-char`, `:macro-char` and `:syntax-from`, but not `readtable-case`.

- **[function]** **find-readtable** *name*

  Looks for the readtable specified by `name` and returns it if it is found. Returns `nil` otherwise.

- **[function]** **ensure-readtable** *name &optional (default nil default-p)*

  Looks up the readtable specified by `name` and returns it if it's found. If it is not found, it registers the readtable designated by `default` under the name represented by `name`; or if no default argument is given, it signals an error of type `readtable-does-not-exist` instead.

- **[function]** **rename-readtable** *old-name new-name*

  Replaces the associated name of the readtable designated by `old-name` with `new-name`. If a readtable is already registered under `new-name`, an error of type `readtable-does-already-exist` is signaled.

- **[function]** **readtable-name** *named-readtable*

  Returns the name of the readtable designated by `named-readtable`, or `nil`.

- **[function]** **register-readtable** *name readtable*

Associate `readtable` with `name`. Returns the readtable.

- **[function]** **unregister-readtable** *named-readtable*

  Remove the association of `named-readtable`. Returns `t` if successfull, `nil` otherwise.

- **[function]** **copy-named-readtable** *named-readtable*

  Like `copy-readtable` but takes a `named-readtable-designator` as argument.

- **[function]** **list-all-named-readtables**

  Returns a list of all registered readtables. The returned list is guaranteed to be fresh, but may contain duplicates.

- **[type]** **named-readtable-designator**

  Either a symbol or a readtable itself.

- **[condition]** **readtable-error** *error*

- **[condition]** **reader-macro-conflict** *readtable-error*

  Continuable.

  This condition is signaled during the merge process if a reader macro (be it a macro character or the sub character of a dispatch macro character) is present in the both source and the target readtable and the two respective reader macro functions differ.

- **[condition]** **readtable-does-already-exist** *readtable-error*

  Continuable.

- **[condition]** **readtable-does-not-exist** *readtable-error*

4 Indices

Referrer definition type abbreviations:

- *f*: for definitions in the function namespace (macros, compiler macros and also methods)
- *t*: `DEFTYPE`s, classes, conditions, structs
- *d*: documentation sections and glossary terms
- *l*: definitions of definition types
- *s*: ASDF systems
- *p*: packages
- *n*: named readtables
- *v*: special variables and constants
- *r*: restarts
- *?*: other

4.1 Function and Macro Index

4.2 Type Index

4.3 Misc Index