

# **Scheme: Compatibility Libraries and Executables**

Version 5.0.2

November 6, 2010

Racket was once called “PLT Scheme,” and a number of libraries with names starting [scheme](#) provide compatibility with the old name. A few old executables are also provided.

## Contents

<b>1</b>	<code>scheme</code>	<b>6</b>
<b>2</b>	<code>scheme/base</code>	<b>7</b>
<b>3</b>	<code>scheme/async-channel</code>	<b>8</b>
<b>4</b>	<code>scheme/bool</code>	<b>9</b>
<b>5</b>	<code>scheme/class</code>	<b>10</b>
<b>6</b>	<code>scheme/cmdline</code>	<b>11</b>
<b>7</b>	<code>scheme/contract</code>	<b>12</b>
<b>8</b>	<code>scheme/control</code>	<b>13</b>
<b>9</b>	<code>scheme/date</code>	<b>14</b>
<b>10</b>	<code>scheme/dict</code>	<b>15</b>
<b>11</b>	<code>scheme/file</code>	<b>16</b>
<b>12</b>	<code>scheme/fixnum</code>	<b>17</b>
<b>13</b>	<code>scheme/flonum</code>	<b>18</b>
<b>14</b>	<code>scheme/foreign</code>	<b>19</b>
<b>15</b>	<code>scheme/function</code>	<b>20</b>
<b>16</b>	<code>scheme/future</code>	<b>21</b>

<b>17</b>	<code>scheme/generator</code>	<b>22</b>
<b>18</b>	<code>scheme/gui</code>	<b>23</b>
<b>19</b>	<code>scheme/gui/base</code>	<b>24</b>
<b>20</b>	<code>scheme/gui/dynamic</code>	<b>25</b>
<b>21</b>	<code>scheme/help</code>	<b>26</b>
<b>22</b>	<code>scheme/include</code>	<b>27</b>
<b>23</b>	<code>scheme/init</code>	<b>28</b>
<b>24</b>	<code>scheme/language-info</code>	<b>29</b>
<b>25</b>	<code>scheme/list</code>	<b>30</b>
<b>26</b>	<code>scheme/load</code>	<b>31</b>
<b>27</b>	<code>scheme/local</code>	<b>32</b>
<b>28</b>	<code>scheme/match</code>	<b>33</b>
<b>29</b>	<code>scheme/math</code>	<b>34</b>
<b>30</b>	<code>scheme/mpair</code>	<b>35</b>
<b>31</b>	<code>scheme/nest</code>	<b>36</b>
<b>32</b>	<code>scheme/package</code>	<b>37</b>
<b>33</b>	<code>scheme/path</code>	<b>38</b>

34	<code>scheme/port</code>	39
35	<code>scheme/pretty</code>	40
36	<code>scheme/promise</code>	41
37	<code>scheme/provide</code>	42
38	<code>scheme/provide-syntax</code>	43
39	<code>scheme/provide-transform</code>	44
40	<code>scheme/require</code>	45
41	<code>scheme/require-syntax</code>	46
42	<code>scheme/require-transform</code>	47
43	<code>scheme/runtime-config</code>	48
44	<code>scheme/runtime-path</code>	49
45	<code>scheme/sandbox</code>	50
46	<code>scheme/serialize</code>	51
47	<code>scheme/set</code>	52
48	<code>scheme/signature</code>	53
49	<code>scheme/shared</code>	54
50	<code>scheme/splicing</code>	55

51	<code>scheme/string</code>	56
52	<code>scheme/struct-info</code>	57
53	<code>scheme/stxparam</code>	58
54	<code>scheme/stxparam-exptime</code>	59
55	<code>scheme/surrogate</code>	60
56	<code>scheme/system</code>	61
57	<code>scheme/tcp</code>	62
58	<code>scheme/trait</code>	63
59	<code>scheme/udp</code>	64
60	<code>scheme/unit</code>	65
61	<code>scheme/unit-exptime</code>	66
62	<code>scheme/unsafe/ops</code>	67
63	<code>scheme/vector</code>	68
64	<code>mred</code>	69
65	<b>Compatibility Executables</b>	70

## 1 `scheme`

```
(require scheme)
```

The `scheme` library re-exports `racket`, except based on `scheme/base` instead of `racket/base`, the `struct` from `scheme/unit` is exported, `scheme/set` is not re-exported, `scheme/system` is not re-exported, `pretty-print` is re-directed in as `scheme/pretty`, and `scheme/nest` is re-exported.

## 2 scheme/base

```
(require scheme/base)
```

The `scheme/base` library re-exports `racket/base`, except that `racket`'s `struct`, `hash`, `hasheq`, `hasheqv`, `in-directory`, and `local-require` are not exported, and `make-base-namespace` and `make-base-empty-namespace` are different.

---

```
(make-base-empty-namespace) → namespace?
```

Like `make-base-empty-namespace` from `racket/base`, but with `scheme/base` attached.

---

```
(make-base-namespace) → namespace?
```

Like `make-base-namespace` from `racket/base`, but with `scheme/base` attached.

### 3 `scheme/async-channel`

```
(require scheme/async-channel)
```

The `scheme/async-channel` library re-exports `racket/async-channel`.

## 4 `scheme/bool`

```
(require scheme/bool)
```

The `scheme/bool` library re-exports `racket/bool`.

## 5 `scheme/class`

```
(require scheme/class)
```

The `scheme/class` library re-exports `racket/class`, except that `writable<%>` is exported under the name `printable<%>` (and `printable<%>` from `racket/class` is not exported).

---

```
printable<%> : interface?
```

An alias for `writable<%>`.

## 6 `scheme/cmdline`

```
(require scheme/cmdline)
```

The `scheme/cmdline` library re-exports `racket/cmdline`.

## 7 `scheme/contract`

```
(require scheme/contract)
```

The `scheme/contract` library re-exports `racket/contract`.

## 8 `scheme/control`

```
(require scheme/control)
```

The `scheme/control` library re-exports `racket/control`.

## 9 `scheme/date`

```
(require scheme/date)
```

The `scheme/date` library re-exports `racket/date`.

## 10 `scheme/dict`

```
(require scheme/dict)
```

The `scheme/dict` library re-exports `racket/dict`.

## 11 `scheme/file`

```
(require scheme/file)
```

The `scheme/file` library re-exports `racket/file`.

## 12 `scheme/fixnum`

```
(require scheme/fixnum)
```

The `scheme/fixnum` library re-exports `racket/fixnum`.

## 13 `scheme/flonum`

```
(require scheme/flonum)
```

The `scheme/flonum` library re-exports `racket/flonum`.

## 14 `scheme/foreign`

```
(require scheme/foreign)
```

The `scheme/foreign` library re-exports `ffi/unsafe`, `ffi/unsafe/cvector`, and `ffi/vector`, except that `unsafe!` must be used to import the unsafe bindings of `ffi/unsafe` and `ffi/unsafe/cvector`.

---

```
(unsafe!)
```

Makes unsafe bindings available.

---

```
(provide* provide-star-spec ...)
```

```
provide-star-spec = (unsafe id)  
                    | (unsafe (rename-out [id external-id]))  
                    | provide-spec
```

Like `provide`, but `ids` under `unsafe` are not actually provided. Instead, they are collected for introduction into an importing module via a macro created by `define-unsafier`.

---

```
(define-unsafier id)
```

Cooperates with `provide*` to define `id` as a `unsafe!`-like form that introduces definitions for each binding provided as `unsafe`. The `define-unsafier` form must occur after all the `provide*` forms to which it refers.

## 15 `scheme/function`

```
(require scheme/function)
```

The `scheme/function` library re-exports `racket/function`.

## 16 `scheme/future`

```
(require scheme/future)
```

The `scheme/future` library re-exports `racket/future`.

## 17 `scheme/generator`

```
(require scheme/generator)
```

The `scheme/generator` library re-exports `racket/generator`.

## 18 `scheme/gui`

```
(require scheme/gui)
```

The `scheme/gui` library re-exports `racket/gui`, except that it builds on `scheme/gui/base` instead of `racket/gui/base`.

## 19 `scheme/gui/base`

```
(require scheme/gui/base)
```

The `scheme/gui/base` library re-exports `racket/gui/base`, except that it builds on `scheme` instead of `racket`.

---

```
(make-gui-empty-namespace) → namespace?
```

Like `make-base-empty-namespace`, but with `scheme/class` and `scheme/gui/base` also attached to the result namespace.

---

```
(make-gui-namespace) → namespace?
```

Like `make-base-namespace`, but with `scheme/class` and `scheme/gui/base` also required into the top-level environment of the result namespace.

## 20 `scheme/gui/dynamic`

```
(require scheme/gui/dynamic)
```

The `scheme/gui/dynamic` library re-exports `racket/gui/dynamic`.

## 21 `scheme/help`

```
(require scheme/help)
```

The `scheme/help` library re-exports `racket/help`.

## 22 `scheme/include`

```
(require scheme/include)
```

The `scheme/include` library re-exports `racket/include`.

## 23 `scheme/init`

```
(require scheme/init)
```

The `scheme/init` library re-exports `racket/init`, except that it builds on `scheme` instead of `racket`.

## 24 `scheme/language-info`

```
(require scheme/language-info)
```

The `scheme/language-info` library is like `racket/language-info`, except that it produces `'(#(scheme/runtime-config configure #f))` for the `'configure-runtime` information key.

See also `scheme/runtime-config`.

## 25 `scheme/list`

```
(require scheme/list)
```

The `scheme/list` library re-exports `racket/list`.

## 26 `scheme/load`

```
(require scheme/load)
```

The `scheme/load` library re-exports `racket/load`.

## 27 `scheme/local`

```
(require scheme/local)
```

The `scheme/local` library re-exports `racket/local`.

## 28 `scheme/match`

```
(require scheme/match)
```

The `scheme/match` library re-exports `racket/match`.

## 29 `scheme/math`

```
(require scheme/math)
```

The `scheme/math` library re-exports `racket/math`.

## 30 `scheme/mpair`

```
(require scheme/mpair)
```

The `scheme/mpair` library re-exports `racket/mpair`.

## 31 scheme/nest

```
(require scheme/nest)
```

---

```
(nest ([datum ...+] ...) body ...+)
```

Combines nested expressions that syntactically drift to the right into a more linear textual format, much in the same way that `let*` linearizes a sequence of nested `let` expressions.

For example,

```
(nest ([let ([x 10]
            [y 6])]
      [with-handlers ([exn:fail? (lambda (x) 15)]]
      [parameterize ([current-output-port (current-error-port)])]
      [let-values (([d r] (quotient/remainder x y)))]
      (display (+ d r))])
```

is equivalent to

```
(let ([x 10]
      [y 6])
  (with-handlers ([exn:fail? (lambda (x) 15)]]
    (parameterize ([current-output-port (current-error-port)])
      (let-values (([d r] (quotient/remainder x y)))
        (display (+ d r))))))
```

The `nest` form is unusual in that it has no semantics apart from its expansion, and its implementation is easier to understand than a precise prose description:

```
(define-syntax nest
  (syntax-rules ()
    [(nest () body0 body ...)
     (let () body0 body ...)]
    [(nest ([form forms ...]) body0 body ...)
     (form forms ... (let () body0 body ...))]
    [(nest ([form forms ...] . more) body0 body ...)
     (form forms ... (nest more body0 body ...))]))
```

## 32 `scheme/package`

```
(require scheme/package)
```

The `scheme/package` library re-exports `racket/package`.

### 33 `scheme/path`

```
(require scheme/path)
```

The `scheme/path` library re-exports `racket/path`.

## 34 `scheme/port`

```
(require scheme/port)
```

The `scheme/port` library re-exports `racket/port`.

## 35 `scheme/pretty`

```
(require scheme/pretty)
```

The `scheme/pretty` library re-exports `racket/pretty`, except that `pretty-write` is exported under the name `pretty-print` (and `pretty-print` from `racket/pretty` is not exported).

---

```
(pretty-print v [port]) → void?  
  v : any/c  
  port : output-port? = (current-output-port)
```

An alias for `pretty-write`.

## 36 `scheme/promise`

```
(require scheme/promise)
```

The `scheme/promise` library re-exports `racket/promise`.

## 37 `scheme/provide`

`(require scheme/provide)`

The `scheme/provide` library re-exports `racket/provide`.

## 38 `scheme/provide-syntax`

```
(require scheme/provide-syntax)
```

The `scheme/provide-syntax` library re-exports `racket/provide-syntax`.

## 39 `scheme/provide-transform`

```
(require scheme/provide-transform)
```

The `scheme/provide-transform` library re-exports `racket/provide-transform`.

## 40 `scheme/require`

```
(require scheme/require)
```

The `scheme/require` library re-exports `racket/require`.

## 41 `scheme/require-syntax`

```
(require scheme/require-syntax)
```

The `scheme/require-syntax` library re-exports `racket/require-syntax`.

## 42 `scheme/require-transform`

```
(require scheme/require-transform)
```

The `scheme/require-transform` library re-exports `racket/require-transform`.

## 43 `scheme/runtime-config`

```
(require scheme/runtime-config)
```

The `scheme/runtime-config` library is like `racket/runtime-config`, except that the result of its `configure` function is a procedure that sets `print-as-expression` to `#f`.

## 44 `scheme/runtime-path`

```
(require scheme/runtime-path)
```

The `scheme/runtime-path` library re-exports `racket/runtime-path`.

## 45 scheme/sandbox

```
(require scheme/sandbox)
```

The `scheme/sandbox` library re-exports `racket/sandbox`, except that `sandbox-namespace-specs`, `make-evaluator`, and `make-module-evaluator` are replaced.

---

```
(sandbox-namespace-specs) → (cons/c (-> namespace?)
                                     (listof module-path?))
(sandbox-namespace-specs spec) → void?
  spec : (cons/c (-> namespace?)
                (listof module-path?))
```

Like `sandbox-namespace-specs` from `racket/sandbox`, but the default is `(list make-base-namespace)` if `gui?` is `#f`, `(list make-gui-namespace)` if `gui?` is `#t`.

---

```
(make-evaluator language
                input-program ...
                #:requires requires
                #:allow-read allow) → (any/c . -> . any)
  language : (or/c module-path?
              (list/c 'special symbol?)
              (cons/c 'begin list?))
  input-program : any/c
  requires : (listof (or/c module-path? path?))
  allow : (listof (or/c module-path? path?))
(make-module-evaluator module-decl
                       #:language lang
                       #:allow-read allow) → (any/c . -> . any)
  module-decl : (or/c syntax? pair?)
  lang : (or/c #f module-path?)
  allow : (listof (or/c module-path? path?))
```

Like `make-evaluator` and `make-module-evaluator` from `racket/sandbox`, but the value of the `sandbox-namespace-specs` parameter is installed as the value of `sandbox-namespace-specs` from `racket/sandbox` before chaining to `make-evaluator` and `make-module-evaluator` from `racket/sandbox`.

## 46 `scheme/serialize`

```
(require scheme/serialize)
```

The `scheme/serialize` library re-exports `racket/serialize`.

## 47 `scheme/set`

```
(require scheme/set)
```

The `scheme/set` library re-exports `racket/set`.

## 48 `scheme/signature`

```
(require scheme/signature)
```

The `scheme/signature` library re-exports `racket/signature`.

## 49 `scheme/shared`

```
(require scheme/shared)
```

The `scheme/shared` library re-exports `racket/shared`.

## 50 `scheme/splicing`

```
(require scheme/splicing)
```

The `scheme/splicing` library re-exports `racket/splicing`.

## 51 `scheme/string`

```
(require scheme/string)
```

The `scheme/string` library re-exports `racket/string`.

## 52 `scheme/struct-info`

```
(require scheme/struct-info)
```

The `scheme/struct-info` library re-exports `racket/struct-info`.

## 53 `scheme/stxparam`

```
(require scheme/stxparam)
```

The `scheme/stxparam` library re-exports `racket/stxparam`.

## 54 `scheme/stxparam-exptime`

```
(require scheme/stxparam-exptime)
```

The `scheme/stxparam-exptime` library re-exports `racket/stxparam-exptime`.

## 55 `scheme/surrogate`

`(require scheme/surrogate)`

The `scheme/surrogate` library re-exports `racket/surrogate`.

## 56 `scheme/system`

`(require scheme/system)`

The `scheme/system` library re-exports `racket/system`.

## 57 `scheme/tcp`

```
(require scheme/tcp)
```

The `scheme/tcp` library re-exports `racket/tcp`.

## 58 `scheme/trait`

```
(require scheme/trait)
```

The `scheme/trait` library re-exports `racket/trait`.

## 59 `scheme/udp`

```
(require scheme/udp)
```

The `scheme/udp` library re-exports `racket/udp`.

## 60 `scheme/unit`

```
(require scheme/unit)
```

The `scheme/unit` library re-exports `racket/unit`.

## 61 `scheme/unit-exptime`

```
(require scheme/unit-exptime)
```

The `scheme/unit-exptime` library re-exports `racket/unit-exptime`.

## 62 `scheme/unsafe/ops`

```
(require scheme/unsafe/ops)
```

The `scheme/unsafe/ops` library re-exports `racket/unsafe/ops`.

## 63 `scheme/vector`

```
(require scheme/vector)
```

The `scheme/vector` library re-exports `racket/vector`.

## 64 mred

```
(require mred)
```

The `mred` library is like `scheme/gui/base`, except that it provides variants of `make-gui-namespace` and `make-gui-empty-namespace` that attach `mred` instead of `scheme/gui/base`.

Both `scheme/gui/base` and `racket/gui/base` depend on `mred`, so it is attached by all variants of `make-gui-empty-namespace`.

## 65 Compatibility Executables

The following executables are included in the Racket distribution for compatibility with older versions of Racket:

- `mzscheme` — the same as `racket -I scheme/init`
- `mred` — the same as `gracket -I scheme/gui/init`
- `drscheme` — the same as `drracket`
- `mzc` — an old interface to some of the tools provided by `raco`, including `raco make` and `raco ctool`; use `mzc --help` for more information
- `plt-help` — the same as `raco docs`