Version: Racket Version Checking

Version 9.0.0.4

November 16, 2025

The "version" collection contains several version-related pieces that are used by Racket. See also version from racket/base.

1 Installed Patch Level

```
(require version/patchlevel)
```
package: base

```
patchlevel : exact-nonnegative-integer?
```

Indicates the current installed patch level, which is normally zero, but may be updated by patches to DrRacket.

2 Checking Available Versions

```
(require version/check)
```
package: base

```
(check-version) → (or/c symbol? list?)
```

Checks the currently available version on the Racket website (`https://download.racket-lang.org`) and returns a value that indicates the state of the current installation:

- `` `ok `` — You're fine.
- `` `(ok-but ,version) `` — You have a fine stable version, but note that there is a newer alpha version available numbered *version*.
- `` `(newer ,version) `` — You have an old version. Please upgrade to *version*.
- `` `(newer ,version ,alpha) `` — You have an old-but-stable version, please upgrade to *version*; you may consider also the newer alpha version numbered *alpha*.
- `` `(error ,message) `` — An error occurred, and *message* is a string that indicates the error.
- `` `(error ,message ,additional-info) `` — An error occurred; *message* is a string that indicates the error, and *additional-info* is a string containing a system error. The *additional-info* content is always parenthesizes, so `message` is a short error and `(string-append message " " additional-info)` is a verbose one.

Note that, depending on network conditions, `check-version` may keep trying for a long time (currently 30 seconds) before returning `'(error "timeout")`. For testing purposes, when the environment variable `PLT_CHECK_VERSION_SIMULATE_TIMEOUT` is set, `check-version` will simulate such network conditions, logging a `'warning`-level message with the topic `'version/check` and then sleeping until the timeout.

3 Version Utilities

```
(require version/utils)
```
package: base

The version/utils library provides a few of convenient utilities for dealing with version strings.

```
(valid-version? v) → boolean?
  v : any/c
```

Returns #t if v is a valid Racket version string, #f otherwise.

A valid version has one of the following forms:

- ⟨*maj*⟩.⟨*min*⟩
- ⟨*maj*⟩.⟨*min*⟩.⟨*sub*⟩
- ⟨*maj*⟩.⟨*min*⟩.⟨*sub*⟩.⟨*rel*⟩

subject to the following constraints:

- ⟨*maj*⟩, ⟨*min*⟩, ⟨*sub*⟩, and ⟨*rel*⟩ are all canonical decimal representations of natural numbers (i.e., decimal digits with no leading 0 unless the number is exactly 0)
- ⟨*rel*⟩ is not 0
- ⟨*sub*⟩ is not 0 unless ⟨*rel*⟩ is included
- ⟨*min*⟩ has no more than two digits
- ⟨*sub*⟩ and ⟨*rel*⟩ have no more than three digits

The constraints force version numbers to be in a canonical form. For example, a would-be version string "4.3.0" must be written instead as "4.3", "4.3.1.0" must be written instead as "4.3.1", and "4" must be written as "4.0".

```
(version->list str)
 → (list/c integer? integer? integer? integer?)
  str : valid-version?
```

Returns a list of four numbers that the given version string represent.

```
(version<? str1 str2) → boolean?
  str1 : valid-version?
  str2 : valid-version?
```

Returns `#t` if *str1* represents a version that is strictly smaller than *str2*, `#f` otherwise.

```
(version<=? str1 str2) → boolean?
  str1 : valid-version?
  str2 : valid-version?
```

Returns `#t` if *str1* represents a version that is smaller than or equal to *str2*, `#f` otherwise.

```
(alpha-version? str) → boolean?
  str : valid-version?
```

Returns `#t` if the version that *str* represents is an alpha version.

A version number of the form ⟨*maj*⟩`.`⟨*min*⟩, ⟨*maj*⟩`.`⟨*min*⟩`.`⟨*sub*⟩, or ⟨*maj*⟩`.`⟨*min*⟩`.`⟨*sub*⟩`.`⟨*rel*⟩ is an alpha version if ⟨*min*⟩ is `90` or more, ⟨*sub*⟩ is `900` or more, or ⟨*rel*⟩ is `900` or more.

```
(version->integer str) → (or/c integer? #f)
  str : string?
```

Converts the version string into an integer. For version `"X.YY.ZZZ.WWW"`, the result will be `XYYZZZWWW`. This function works also for legacy Racket versions by translating `"XYY.ZZZ"` to `XYYZZZ000`. The resulting integer can thefore be used to conveniently compare any two (valid) version strings. If the version string is invalid as either a regular version string or a legacy version string, the resulting value is `#f`.

Note that this is the only function that deals with legacy version strings.