Version: PLT Version Checking

Version 4.1.4

January 20, 2009

The version collection contains several version-related pieces that are used by PLT Scheme.
See also `version` from `scheme/base`.

1 Installed Patch Level

```
(require version/patchlevel)
```

patchlevel : exact-nonnegative-integer?

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

2 Checking Available Versions

```
(require version/check)
```

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

Checks the currently available version on the PLT website (`http://download.plt-scheme.org`) and returns a value that indicates the current state of the curent 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.

3 DrScheme Version Tool

```
(require version/tool)
```

The version/tool library implements a DrScheme tool that

- makes the patchlevel display as a version p$\langle N \rangle$ suffix in DrScheme (though the base verion reported by (version) is not changed);

- if enabled by the user, periodically checks whether a new PLT Scheme distribution is available for download.

4 Version Utilities

```
(require version/utils)
```

The `version/utils` library provides a few of convenient utilities for dealing with version strings. Unless explicitly noted, these functions do not handle legacy versions of PLT Scheme.

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

Returns `#t` if `str` is a valid PLT Scheme version string, `#f` otherwise.

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

Returns a list of four numbers that the given version string represent. `str` is assumed to be a valid version.

```
(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. `str1` and `str2` are assumed to be valid versions.

```
(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. `str1` and `str2` are assumed to be valid versions.

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

Returns `#t` if the version that `str` represents is an alpha version. `str` is assumed to be a valid version.

```
(version->integer str) → (or/c integer? false/c)
```

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 PLT Scheme 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 the resulting value is `#f`.

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