Racket-Ebuild

Version 8.5

Maciej Barć <xgqt@riseup.net>

May 5, 2022

Library to ease automatic ebuild creation and maintenance.

Version: 13.1.6, commit hash: 8880f74

Contents

Index **49**

1 About

1.1 Project Aim

Racket-Ebuild is primarily made with collector2 in mind.

This package is meant to help Gentoo (and it's forks) developers in creating and maintaining ebuilds.

1.2 Development

1.2.1 Tools

- Gentoo GNU+Linux system (for testing generated ebuilds) with following tools:
  - repoman — for verifying ebuild QA correctness
  - ebuild-mode — if using GNU Emacs for viewing or manually editing ebuilds
- IDE/editor with editorconfig support (or just follow rules in `".editorconfig"` file)
- DrRacket for formatting Scribble code (and writing Racket code in general if it is your editor of choice)
- Web browser for reading documentation (generated from Scribble code)

1.2.2 Weirdness

- Besides non-lisp style and sometimes weird indentation...
- "ebuild" and "metadata" are implemented as classes
- This is one repository that has many packages (so-called monorepo)

1.3 Upstream

The upstream repository can be found on GitLab.

GitLab allows to run CI pipelines and to generate static web pages for documentation.

Configuration for GitLab CI/CD pipelines can be found in `.gitlab-ci.yml`.

1.4 License

Racket-Ebuild is released under GNU GPL, version 3 (only) license.

Read the license text here.

2 Ebuild - Classes

```
(require ebuild)
```
package: ebuild-lib

ebuild module reexports functions from the class module and modules included in this section, that is: ebuild/ebuild, ebuild/metadata, ebuild/package and ebuild/repository

2.1 Ebuild Class

```
(require ebuild/ebuild)
```
package: ebuild-lib

```
ebuild% : class?
  superclass: object%
  extends: printable<%>
```

Ebuild class.

For creating package ebuild files ("package.ebuild").

```
(new ebuild%
   [[year year]
    [EAPI EAPI]
    [custom custom]
    [inherits inherits]
    [DESCRIPTION DESCRIPTION]
    [HOMEPAGE HOMEPAGE]
    [SRC_URI SRC_URI]
    [S S]
    [LICENSE LICENSE]
    [SLOT SLOT]
    [KEYWORDS KEYWORDS]
    [IUSE IUSE]
    [REQUIRED_USE REQUIRED_USE]
    [RESTRICT RESTRICT]
    [COMMON_DEPEND COMMON_DEPEND]
    [RDEPEND RDEPEND]
    [DEPEND DEPEND]
    [BDEPEND BDEPEND]
    [PDEPEND PDEPEND]
    [body body]])
→ (is-a?/c ebuild%)
 year : integer? = (date-year (current-date))
 EAPI : integer? = 7
```

```
  custom : (listof (or/c (-> any) string?)) = '()
  inherits : (listof string?) = '()
  DESCRIPTION : string? = "package"
  HOMEPAGE : string?
           = "https://wiki.gentoo.org/wiki/No_homepage"
  SRC_URI : (listof src-uri?) = '()
  S : (or/c #f string?) = #f
  LICENSE : string? = "all-rights-reserved"
  SLOT : string? = "0"
  KEYWORDS : (listof string?) = '("~amd64")
  IUSE : (listof string?) = '()
  REQUIRED_USE : (listof cflag?) = '()
  RESTRICT : (listof (or/c cflag? string?)) = '()
  COMMON_DEPEND : (listof (or/c cflag? string?)) = '()
  RDEPEND : (listof (or/c cflag? string?)) = '()
  DEPEND : (listof (or/c cflag? string?)) = '()
  BDEPEND : (listof (or/c cflag? string?)) = '()
  PDEPEND : (listof (or/c cflag? string?)) = '()
  body : (listof (or/c (-> any) string?)) = '()
```

```
(send an-ebuild create-list) → list?
```

Creates a `list` that contains `string`s or `false` as elements. The elaments of the list are created using the "unroll" functions which are not exposed to the user.

Also, concatenates values that `custom` and `body` arguments return.

```
(send an-ebuild create) → string?
```

Takes non-`false` elemets of the `list` created by `create-list` method and turns them into one `string` ready to be written into a file (by `save` for example).

```
(send an-ebuild save name [pth]) → void
  name : string?
  pth : path-string? = (current-directory)
```

Creates a file named *name* in the given location *pth* (or current directory). Internally uses the interfece implemented by this object's `printable<%>` to dispaly this to file.

```
(send an-ebuild append! sym lst) → void
  sym : symbol?
  lst : list?
```

Uses `dynamic-get-field` and `dynamic-set-field!`. Sets the field-name represented by *sym* symbol of this object to that field appended with *lst* `list`.

```
(send an-ebuild concat! sym v) → void
  sym : symbol?
  v : any/c
```

Like `append!` but a list is automatically generated from `v`.

Examples:

```
> (define my-ebuild
    (new ebuild%
         [IUSE (list "debug" "test")]
         [RESTRICT (list (cflag "test?" (list "debug")))]))
> (display my-ebuild)
# Copyright 1999-2022 Gentoo Authors
# Distributed under the terms of the GNU General Public
License v2

EAPI=8

DESCRIPTION="package"
HOMEPAGE="https://wiki.gentoo.org/wiki/No_homepage"

LICENSE="all-rights-reserved"
SLOT="0"
KEYWORDS="~amd64"
IUSE="debug test"
RESTRICT="test? ( debug )"
```

2.2 Metadata Class

`(require ebuild/metadata)` package: `ebuild-lib`

```
metadata% : class?
  superclass: object%
  extends: printable<%>
```

Metadata class.

For crating package metadata files (`"metadata.xml"`).

```
(new metadata%
   [[maintainers maintainers]
    [longdescriptions longdescriptions]
    [slots slots]
    [stabilize-allarches stabilize-allarches]
    [uses uses]
    [upstream upstream]])
→ (is-a?/c metadata%)
 maintainers : (listof maintainer?) = '()
 longdescriptions : (listof longdescription?) = '()
 slots : (listof slots?) = '()
 stabilize-allarches : boolean? = #f
 uses : (listof use?) = '()
 upstream : (or/c #f upstream?) = #f
```

```
(send a-metadata create) → document?
```

Creates a XML `document` ready to be written into a file.

```
(send a-metadata save [pth]) → void
 pth : path-string? = (current-directory)
```

Creates a file named `"metadata.xml"` in the given location (or current directory). Internally uses the interfece implemented by this object's `printable<%>` to dispaly object to file.

Examples:

```
> (define my-metadata
    (new metadata%
         [maintainers
          (list (maintainer 'person #f "me@me.com" "Me" #f))]
         [upstream
          (upstream (list) #f #f #f (list (remote-
id 'gitlab "me/myproject")))]))
> (display my-metadata)
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "https://www.gentoo.org/dtd/metadata.dtd">

<pkgmetadata>
  <maintainer type="person">
    <email>me@me.com</email>
    <name>Me</name>
  </maintainer>
  <upstream>
    <remote-id type="gitlab">me/myproject</remote-id>
  </upstream>
</pkgmetadata>
```

2.3 Package Class

```
(require ebuild/package)
```
package: ebuild-lib

```
package% : class?
  superclass: object%
```

Package class.

For creating packages.

```
(new package%
   [[CATEGORY CATEGORY]
    [PN PN]
    [ebuilds ebuilds]
    [metadata metadata]]) → (is-a?/c package%)
 CATEGORY : string? = "app-misc"
 PN : string? = "unknown"
 ebuilds : (hash/c package-version? (is-a?/c ebuild%))
         = (hash (live-version) (new ebuild%))
 metadata : (is-a?/c metadata%) = (new metadata%)
```

By default if ebuilds are not given the default ebuild% object (with PN set to "unknown" is used) and if metadata is not given default "empty" metadata% object is used.

```
(send a-package get-versions) → (listof package-version?)
```

Return a list of package-versions extracted from ebuilds.

Example:

```
> (send (new package%) get-versions)
(list (package-version "9999" #f #f #f #f #f))
```

```
(send a-package get-PVs) → (listof string?)
```

Return a list of package-versions as strings extracted from ebuilds.

Example:

```
> (send (new package%) get-PVs)
'("9999")
```

```
(send a-package get-CATEGORY/PN) → string?
```

Return a `string` composed of: `CATEGORY`, `"/"` and `PN`.

Example:

```
> (send (new package%) get-CATEGORY/PN)
"app-misc/unknown"
```

`(send a-package get-Ps) → (listof string?)`

Return a `list` of `package-version`s joined with `get-CATEGORY/PN`.

Example:

```
> (send
   (new package%
        [CATEGORY  "dev-scheme"]
        [PN  "bytes"]
        [ebuilds
         (hash (simple-version "1") (new ebuild%)
               (simple-version "2") (new ebuild%))])
   get-Ps)
'("dev-scheme/bytes-1" "dev-scheme/bytes-2")
```

`(send a-package show) → void`

Display `ebuilds` and `metadata`.

Example:

```
> (send (new package%) show)
app-misc/unknown
9999
# Copyright 1999-2022 Gentoo Authors
# Distributed under the terms of the GNU General Public
License v2

EAPI=8

DESCRIPTION="package"
HOMEPAGE="https://wiki.gentoo.org/wiki/No_homepage"

LICENSE="all-rights-reserved"
SLOT="0"
KEYWORDS="~amd64"
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "https://www.gentoo.org/dtd/metadata.dtd">

<pkgmetadata>
</pkgmetadata>
```

```
(send a-package save [pth]) → void
  pth : path-string? = (current-directory)
```

Creates (uses their save methods) `ebuilds` and `metadata` of this package. The names of ebuilds are equal to so-called P ebuild variable which is composed of PN (taken from this `package%` object) and PV (version, taken from the ebuilds hash). So ebuilds will be saved as `"P.ebuild"`.

2.4 Repository Class

```
(require ebuild/repository)
```
package: `ebuild-lib`

```
repository% : class?
  superclass: object%
```

Repository class.

For creating ebuild repository structures.

```
(new repository%
     [name name]
    [[layout layout]]
     [packages packages]) → (is-a?/c repository%)
  name : string?
  layout : layout? = (default-layout)
  packages : (listof (is-a?/c package%))
```

By default if *layout* is not given `default-layout` (parameter variable) is used.

```
(send a-repository show) → void
```

Display repository `name`, `layout` and `packages` it contains.

```
(send a-repository layout-string) → string?
```

Wrapper for `layout->string`.

```
(send a-repository save-layout [pth]) → void
  pth : path-string? = (current-directory)
```

Creates directory `"metadata"` with a file `"layout.conf"` in the given location (or current directory).

```
(send a-repository save-name [pth]) → void
  pth : path-string? = (current-directory)
```

Creates directory "profiles" with a file "repo_name" in the given location (or current directory).

```
(send a-repository save-packages [pth]) → void
  pth : path-string? = (current-directory)
```

Creates directory "CATEGORY/PN" with a "metadata.xml" file and "P.ebuild" ebuilds in the given location (or current directory).

```
(send a-repository save [pth]) → void
  pth : path-string? = (current-directory)
```

Creates a full ebuild repository directory structure. (uses save-layout, save-name and save-packages).

Examples:

```
> (define r
    (new repository%
         [name "test"]
         [packages
          (list
           (new package%
                [CATEGORY "sys-devel"]
                [PN "asd"]
                [ebuilds (hash (simple-version "1.1.1") (new ebuild%))]
                [metadata
                 (new metadata%
                      [upstream
                       (upstream
                        (list) #f #f #f (list (remote-
id 'gitlab "asd/asd")))])]))]))
> (display (send r layout-string))
cache-formats = md5-dict
eapis-banned = 0 1 2 3 4 5 6
eapis-deprecated =
manifest-hashes = BLAKE2B SHA512
manifest-required-hashes = BLAKE2B
masters = gentoo
sign-commits = true
sign-manifests = false
thin-manifests = true
update-changelog = false
```

```
> (send r show)
Repository name:
test
Repository layout:
cache-formats = md5-dict
eapis-banned = 0 1 2 3 4 5 6
eapis-deprecated =
manifest-hashes = BLAKE2B SHA512
manifest-required-hashes = BLAKE2B
masters = gentoo
sign-commits = true
sign-manifests = false
thin-manifests = true
update-changelog = false
Packages:
sys-devel/asd
1.1.1
# Copyright 1999-2022 Gentoo Authors
# Distributed under the terms of the GNU General Public License v2

EAPI=8

DESCRIPTION="package"
HOMEPAGE="https://wiki.gentoo.org/wiki/No_homepage"

LICENSE="all-rights-reserved"
SLOT="0"
KEYWORDS="~amd64"
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "https://www.gentoo.org/dtd/metadata.dtd">

<pkgmetadata>
  <upstream>
    <remote-id type="gitlab">asd/asd</remote-id>
  </upstream>
</pkgmetadata>
```

3 Ebuild - Exported Functions

Functions used to be prefixed with "ebuild:" and "metadata:". If you still wish to keep this naming scheme, use:

```
(require (prefix-in ebuild: ebuild/ebuild)
         (prefix-in metadata: ebuild/metadata))
```

3.1 Ebuild Functions

3.1.1 Constraint Flag

```
(struct cflag (predicate content)
    #:extra-constructor-name make-cflag)
  predicate : (or/c #f string?)
  content : (listof (or/c cflag? string?))
```

No restrictive contracts are enforced upon neither `predicate` nor `content` for now.

`cflag`s can be nested, for recursively transforming them, see `cflag->string`.

`predicate` symbolizes the logical ebuild condition to satisfy.

Examples:

```
> (define restrict
    (cflag "||" (list "qt5" "gtk")))
> restrict
(cflag "||" '("qt5" "gtk"))
> (cflag->string restrict)
"|| ( qt5\n\tgtk )"
> (display (cflag->string restrict))
|| ( qt5
 gtk )
> (define iuse
    (cflag "X?" (list (cflag "gtk?" (list "x11-libs/gtk+:3"))
                      (cflag "qt5?" (list "dev-qt/qtcore:5")))))
> iuse
(cflag
 "X?"
 (list
  (cflag "gtk?" '("x11-libs/gtk+:3"))
  (cflag "qt5?" '("dev-qt/qtcore:5"))))
```

```
> (for ([i '(0 1 2)])
    {define s (cflag->string iuse i)}
    (printf "~v\n" s)
    (displayln s))
"X? ( gtk? ( x11-libs/gtk+:3 )\nqt5? ( dev-qt/qtcore:5 ) )"
X? ( gtk? ( x11-libs/gtk+:3 )
qt5? ( dev-qt/qtcore:5 ) )
"X? ( gtk? ( x11-libs/gtk+:3 )\n\tqt5? ( dev-qt/qtcore:5 ) )"
X? ( gtk? ( x11-libs/gtk+:3 )
 qt5? ( dev-qt/qtcore:5 ) )
"X? ( gtk? ( x11-libs/gtk+:3 )\n\t\tqt5? ( dev-qt/qtcore:5 ) )"
X? ( gtk? ( x11-libs/gtk+:3 )
 qt5? ( dev-qt/qtcore:5 ) )
```

```
(cflag->string fl [tabs #:flat? flat?]) → string?
  fl : cflag?
  tabs : exact-nonnegative-integer? = 1
  flat? : boolean? = #f
```

Extracts `cflag` contents to a `string`.

`cflag->string` calls itself recursively if a `cflag` is encountered inside currently given *`fl`* variable.

Each internal recursive call increments *`tabs`* by 1. If *`tabs`* is given (grater than 0), then that tab step is applied to the result.

If *`flat?`* is `true`, then the produced `string` does not contain neither tabs nor newlines.

Example:

```
> (cflag->string (cflag "||" (list "qt5" "gtk")) #:flat? #t)
"|| ( qt5 gtk )"
```

3.1.2 Source URL

Functions and structs used to create SRC_URI ebuild variable.

```
(struct src-uri (flag url name)
    #:extra-constructor-name make-src-uri)
  flag : (or/c #f string?)
  url : string?
  name : (or/c #f string?)
```

Struct to hold URLs of SRC_URI.

Fields represent inside a ebuild:

- flag — condition to grab given URL
- url — any URL, maybe with shell variables inside
- name — optional file name to which source downloaded from `url` will be renamd to, ie.: `"${P}.tag.gz"`

```
(src-uri-src su) → string?
  su : src-uri
```

Creates a source string from given `src-uri`.

Example:

```
> (src-uri-src
   (src-uri
    "doc" "https://mywebsite.com/${PN}-docs-${PV}.zip" "${P}-
docs.zip"))
"https://mywebsite.com/${PN}-docs-${PV}.zip -> ${P}-docs.zip"
```

```
(src-uri->string su) → string?
  su : src-uri
```

Creates a source string, with optional flag, from given `src-uri`.

Example:

```
> (src-uri->string
   (src-uri
    "doc" "https://mywebsite.com/${PN}-docs-${PV}.zip" "${P}-
docs.zip"))
"doc ( https://mywebsite.com/${PN}-docs-${PV}.zip -> ${P}-docs.zip
)"
```

3.1.3 Shell Variables

```
(list-as-variable name lst) → (or/c #f string?)
  name : string?
  lst : (or/c #f (listof (or/c number? string? symbol?)))
```

Returns a string that can be used in creating POSIX-like shell scripts. `"name=\"value\""` if only value in `list lst` is a string or 2 or more of any type of values or `"name=value"`

if given `list` *`lst`* with one value that is a number or symbol. If `#f` is given as *`lst`*, then `#f` is returned.

Examples:

```
> (list-as-variable "TURN_ON_FEATURE" '(YES))
"TURN_ON_FEATURE=YES"
> (list-as-variable "TURN_ON_FEATURES" '(THIS THAT))
"TURN_ON_FEATURES=\"THIS THAT\""
```

```
(as-variable name value ...) → (or/c #f string?)
  name : string?
  value : (or/c #f (or/c number? string? symbol?))
```

Wrapper for `list-as-variable`.

Example:

```
> (as-variable "TURN_ON_FEATURE" 'YES)
"TURN_ON_FEATURE=YES"
```

```
(make-variable name)

  name : (or/c identifier? string? symbol?)
```

Uses `as-variable`. If *`name`* is an identifier, then it is the name of variable and the value is what that *`name`* resolves to. If *`name`* is a string or a symbol then it is passed to both name and value of `as-variable` (symbol is converted to a string).

Examples:

```
> (make-variable "this_variable_will_probably_change")
"this_variable_will_probably_change=\"this_variable_will_probably_change\""
> (define Z "Zzz...")
> (make-variable Z)
"Z=\"Zzz...\""
```

```
(list-as-array-variable name lst) → (or/c #f string?)
  name : string?
  lst : (or/c #f (listof (or/c number? string? symbol?)))
```

Simialr to `list-as-variable`, except for arrays.

Example:

```
> (list-as-array-variable "FEATURES" '("mirror" "test"))
"FEATURES=(\"mirror\" \"test\")"
```

```
(as-array-variable name value ...) → (or/c #f string?)
  name : string?
  value : (or/c #f (or/c number? string? symbol?))
```

Wrapper for `list-as-variable`.

Example:

```
> (as-array-variable "FEATURES" "mirror" "test")
"FEATURES=(\"mirror\" \"test\")"
```

```
(make-array-variable name)

  name : (or/c identifier? string? symbol?)
```

Uses `list-as-array-variable` and `as-array-variable`.

Examples:

```
> (make-array-variable "this_variable_will_probably_change")
"this_variable_will_probably_change=(\"this_variable_will_probably_change\")"
> (define FEATURES '("mirror" "test"))
> (make-array-variable FEATURES)
"FEATURES=(\"mirror\" \"test\")"
```

3.1.4 Object manipulation

```
(ebuild-append! id obj lst)

  id : identifier?
  obj : (is-a? ebuild%)
  lst : list?
```

Calls `append!` method of a given *obj* `ebuild%` object.

```
(ebuild-concat! id obj v)

  id : identifier?
  obj : (is-a? ebuild%)
  v : any/c
```

Calls `concat!` method of a given *obj* `ebuild%` object.

3.2 Shell Script Functions

```
(require ebuild/sh-function)
```
package: `ebuild-lib`

3.2.1 POSIX shell script function creation

Ease the creation of ebuild scripts' "body".

```
(struct sh-function (name body)
    #:extra-constructor-name make-sh-function)
  name : string?
  body : string?
```

Example:

```
> (sh-function "my_function" "do_something || die")
#<sh-function>
```

```
(sh-function->string sf) → string?
  sf : sh-function?
```

Creates a string that looks like a function from as POSIX shell script.

Example:

```
> (display (sh-function->string
            (sh-function "my_function" "do_something || die")))
my_function() {
do_something || die
}
```

```
(unroll-sh-functions lst) → string?
  lst : (listof sh-function?)
```

Uses `sh-function->string` to create a shell script body string. Basically it is some POSIX shell script functions.

```
(make-script #:indent indent strs ...) → string?
  indent : exact-integer?
  strs : string?
```

Produces a shell script body from any amount of given strings, with the indentation *indent* represented by tab characters (this its to comply with the ebuild format where we indent with tabs).

This function exists to ease writing custom ebuild (shell script) functions.

Example:

```
> (display (sh-function->string
            (sh-function "my_function" (make-script "do_something
|| die"))))
my_function() {
 do_something || die
}
```

3.3 Metadata Functions

The `structname->xexpr` procedures are primarily used internally but are also exported because sometimes they can be handy.

3.3.1 XML tags

```
(metadata-empty-tags) → list?
(metadata-empty-tags list) → void?
  list : list?
 = (list 'stabilize-allarches)
```

Parameter that determines shorthanded tags *list* passed to `empty-tag-shorthand`.

3.3.2 localized

```
(struct localized (lang data)
    #:extra-constructor-name make-localized)
  lang : (or/c #f string? symbol?)
  data : string?
```

"Intermediate" strcture inherited by in `longdescription` and `doc`.

```
(localized->xexpr lo element-name) → xexpr?
  lo : localized?
  element-name : symbol?
```

Converts *lo* `localized` struct to a x-expression, where the x-expression tag is *element-name*.

3.3.3 longdescription

```
(struct longdescription localized ()
    #:extra-constructor-name make-longdescription)
```

```
(longdescription->xexpr ld) → xexpr?
  ld : longdescription?
```

Converts *ld* longdescription struct to a x-expression.

Examples:

```
> (define my-longdescription
    (longdescription
     "en"
     "This is some package providing some very important func-
tion,\n  everybody probably needs it, of course it is written in
the best\n  programming language starting with letter R ;D"))
> (display-xml/content (xexpr->xml (longdescription->xexpr my-
longdescription)))

<longdescription lang="en">
  This is some package providing some very important function,
  everybody probably needs it, of course it is written in the best
  programming language starting with letter R ;D
</longdescription>
```

3.3.4 maintainer

```
(struct maintainer (type proxied email name description)
    #:extra-constructor-name make-maintainer)
  type : (or/c 'person 'project 'unknown)
  proxied : (or/c #f 'yes 'no 'proxy)
  email : string?
  name : (or/c #f string?)
  description : (or/c #f string?)
```

```
(maintainer->xexpr maint) → xexpr?
  maint : maintainer?
```

Converts *maint* maintainer struct to a x-expression.

Examples:

```
> (define my-maintainer
    (maintainer 'person #f "asd@asd.asd" "A.S.D." #f))
> (display-xml/content (xexpr->xml (maintainer->xexpr my-
maintainer)))

<maintainer type="person">
  <email>
    asd@asd.asd
  </email>
  <name>
    A.S.D.
  </name>
</maintainer>
```

3.3.5 slots

```
(struct slot (name data)
    #:extra-constructor-name make-slot)
  name : string?
  data : string?
```

```
(struct slots (lang slotlist subslots)
    #:extra-constructor-name make-slots)
  lang : (or/c #f string? symbol?)
  slotlist : (or/c #f (listof slot?))
  subslots : (or/c #f string?)
```

```
(slots->xexpr ss) → xexpr?
  ss : slots?
```

Converts *ss* slots struct to a x-expression.

Examples:

```
> (define my-slots
    (slots #f (list (slot "1.9" "Provides libmypkg19.so.0")) #f))
> (display-xml/content (xexpr->xml (slots->xexpr my-slots)))

<slots>
  <slot name="1.9">
    Provides libmypkg19.so.0
  </slot>
</slots>
```

3.3.6 upstream

```
(struct upstreammaintainer (status email name)
    #:extra-constructor-name make-upstreammaintainer)
  status : (or/c 'active 'inactive 'unknown)
  email : string?
  name : string?
```

```
(struct remote-id (type tracker)
    #:extra-constructor-name make-remote-id)
  type : (or/c
          'bitbucket 'cpan 'cpan-module 'cpe 'cran 'ctan
          'freecode 'freshmeat
          'gentoo 'github 'gitlab 'gitorious 'google-code
          'heptapod
          'launchpad
          'pear 'pecl 'pypi
          'rubyforge 'rubygems 'sourceforge 'sourceforge-jp
          'vim)
  tracker : string?
```

```
(struct doc localized ()
    #:extra-constructor-name make-doc)
```

```
(docs->xexpr v) → xexpr?
  v : (or/c #f string? (listof doc?))
```

Given a `string` or `list` of `doc` produces produces `list` of x-expressions. Given `false` produces a empty list.

```
(struct upstream (maintainers changelog doc bugs-to remote-ids)
    #:extra-constructor-name make-upstream)
  maintainers : (listof upstreammaintainer?)
  changelog : (or/c #f string?)
  doc : (or/c #f string? (listof doc?))
  bugs-to : (or/c #f string?)
  remote-ids : (listof remote-id?)
```

```
(upstream->xexpr up) → xexpr?
  up : upstream?
```

Converts *up* `upstream` struct to a x-expression.

Examples:

```
> (define my-upstream
    (upstream (list) #f #f #f (list (remote-
id 'gitlab "asd/asd"))))
> (display-xml/content (xexpr->xml (upstream->xexpr my-upstream)))

<upstream>
  <remote-id type="gitlab">
    asd/asd
  </remote-id>
</upstream>
> (set-upstream-doc! my-upstream "https://asd.asd/docs.html")
> (display-xml/content (xexpr->xml (upstream->xexpr my-upstream)))

<upstream>
  <doc>
    https://asd.asd/docs.html
  </doc>
  <remote-id type="gitlab">
    asd/asd
  </remote-id>
</upstream>
> (set-upstream-doc! my-upstream (list (doc "en" "https://asd.asd/doc.html")
                                       (doc "pl" "https://asd.asd/dok.html")))
> (display-xml/content (xexpr->xml (upstream->xexpr my-upstream)))

<upstream>
  <doc lang="en">
    https://asd.asd/doc.html
  </doc>
  <doc lang="pl">
    https://asd.asd/dok.html
  </doc>
  <remote-id type="gitlab">
    asd/asd
  </remote-id>
</upstream>
```

3.3.7 use

```
(struct uflag (name data)
    #:extra-constructor-name make-uflag)
  name : string?
  data : (or/c #f string?)
```

```
(struct use (lang flags)
    #:extra-constructor-name make-use)
  lang : (or/c #f string? symbol?)
  flags : (listof uflag?)
```

```
(use->xexpr us) → xexpr?
  us : use?
```

Converts *us* `use` struct to a x-expression.

Examples:

```
> (define my-use
    (use #f (list (uflag "racket" "Build using dev-
scheme/racket"))))
> (display-xml/content (xexpr->xml (use->xexpr my-use)))

<use>
  <flag name="racket">
    Build using dev-scheme/racket
  </flag>
</use>
```

3.4 Package Functions

3.4.1 Package Version

```
(release? v) → boolean?
  v : any/c
```

Checks if given argement passes package-version validation. True if is a string a and has only digits and dots (".") and no more than one letter at the end.

Examples:

```
> (release? "")
#f
> (release? "0")
#t
> (release? "1q")
#t
> (release? "1.2.3a")
#t
> (release? "1.2.3abc")
#f
```

```
(struct package-version (release phase pre rc patch revision)
    #:extra-constructor-name make-package-version)
  release : release?
  phase : (or/c #f 'a 'alpha 'b 'beta)
  pre : (or/c boolean? exact-nonnegative-integer?)
  rc : (or/c boolean? exact-nonnegative-integer?)
  patch : (or/c #f exact-positive-integer?)
  revision : (or/c #f exact-positive-integer?)
```

Example:

```
> (package-version "0a" 'beta 1 2 3 4)
(package-version "0a" 'beta 1 2 3 4)
```

```
(live-version [nines]) → package-version
  nines : (and/c positive? exact-integer?) = 4
```

Generates a live `package-version`, that is: the one that has only `9`s in "release" (all other fields are `false`). Optional number determines athe number of `9`s.

Example:

```
> (live-version 8)
(package-version "99999999" #f #f #f #f #f)
```

```
(simple-version rel) → package-version
  rel : release?
```

Generates a `package-version` with only a given *rel* (other fields are #f).

Example:

```
> (simple-version "8.0")
(package-version "8.0" #f #f #f #f #f)
```

```
(package-version->string ver) → string?
  ver : package-version?
```

Converts `package-version` struct to a string.

Examples:

```
> (package-version->string (package-version "0a" 'beta 1 2 3 4))
"0a_beta_pre1_rc2_p3-r4"
> (package-version->string (live-version))
"9999"
> (package-version->string (simple-version "0a"))
"0a"
```

3.5 Repository Functions

3.5.1 Layout

```
(struct layout (masters
                cache-formats
                sign-commits
                update-changelog
                eapis-banned
                eapis-deprecated
                manifest-hashes
                manifest-required-hashes
                sign-manifests
                thin-manifests)
  #:extra-constructor-name make-layout)
  masters : string?
  cache-formats : (listof string?)
  sign-commits : boolean?
  update-changelog : boolean?
  eapis-banned : (listof exact-integer?)
  eapis-deprecated : (listof exact-integer?)
  manifest-hashes : (listof string?)
  manifest-required-hashes : (listof string?)
  sign-manifests : boolean?
  thin-manifests : boolean?
```

```
(default-layout) → layout?
(default-layout layout) → void?
  layout : layout?
```

```
= (layout
   "gentoo"
   '("md5-dict")
   #t
   #f
   '(0 1 2 3 4 5 6)
   '()
   '("BLAKE2B" "SHA512")
   '("BLAKE2B")
   #f
   #t)
```

Parameter that determines default *layout* used when creating a repository% object.

```
(layout->string lo) → string?
  lo : layout?
```

Converts *lo* layout struct to a string.

Example:

```
> (display (layout->string (default-layout)))
cache-formats = md5-dict
eapis-banned = 0 1 2 3 4 5 6
eapis-deprecated =
manifest-hashes = BLAKE2B SHA512
manifest-required-hashes = BLAKE2B
masters = gentoo
sign-commits = true
sign-manifests = false
thin-manifests = true
update-changelog = false
```

4 Ebuild Transformers

4.1 GitHost

```
(require ebuild/transformers/githost)
```
package: ebuild-transformers

```
(struct githost (domain repository)
    #:extra-constructor-name make-githost)
  domain : string?
  repository : string?
```

Example:

```
> (githost "gitlab.com" "asd/asd")
#<githost>
```

```
(url->githost ur) → githost?
  ur : url?
```

Converts a *ur* url to githost.

```
(string->githost str) → githost?
  str : string?
```

Converts a *str* string to githost.

Example:

```
> (string->githost "https://gitlab.com/asd/asd.git")
#<githost>
```

```
(githost->string gh) → string?
  gh : githost?
```

Converts a *gh* githost to string.

Example:

```
> (githost->string (githost "gitlab.com" "asd/asd"))
"https://gitlab.com/asd/asd"
```

```
(githost->url gh) → url?
  gh : githost?
```

Converts a *gh* `githost` to `url`.

Example:

```
> (githost->url (githost "gitlab.com" "asd/asd"))
(url
 "https"
 #f
 "gitlab.com"
 #f
 #t
 (list (path/param "asd" '()) (path/param "asd" '()))
 '()
 #f)
```

4.2 SRC_URI Transformers

```
(require ebuild/transformers/src-uri)
```
package: `ebuild-transformers`

```
(url->src-uri ur pn) → src-uri
  ur : path-string?
  pn : string?
```

Converts a URL *ur* path-`string` to a `src-uri` struct.

If *pn* is supplied also a PN (package name) detection is carried out.

Examples:

```
> (define racket-url "https://mirror.racket-
lang.org/installers/8.1/racket-8.1-src-builtpkgs.tgz")
> (define racket-src-uri (url->src-uri racket-url "racket"))
> racket-src-uri
(src-uri
 #f
 "https://mirror.${PN}-lang.org/installers/8.1/${PN}-8.1-src-
builtpkgs.tgz"
 #f)
> (src-uri->string racket-src-uri)
"https://mirror.${PN}-lang.org/installers/8.1/${PN}-8.1-src-
builtpkgs.tgz"
```

4.3 CFlag Transformers

```
(require ebuild/transformers/cflag)
```
package: ebuild-transformers

```
(string->cflag str) → (listof (or/c cflag? string?))
  str : string?
```

Converts a string back to cflags.

4.4 Metadata Transformers

```
(require ebuild/transformers/metadata)
```
package: ebuild-transformers

4.4.1 longdescription transformation

```
(xexpr->longdescriptions xexpr) → (listof longdescription?)
  xexpr : xexpr?
```

Converts a *xexpr* to list of longdescriptions.

Example:

```
> (xexpr->longdescriptions
   '(pkgmetadata (longdescription ((lang "en")) "Description")
                 (longdescription ((lang "pl")) "Opis")))
(list (longdescription "en" "Description") (longdescription "pl"
"Opis"))
```

4.4.2 maintainer transformation

```
(xexpr->maintainers xexpr) → (listof maintainer?)
  xexpr : xexpr?
```

Converts a *xexpr* to list of maintainers.

Example:

```
> (xexpr->maintainers
   '(pkgmetadata (maintainer ((type "person"))
                             (email "asd@asd.asd") (name "A.S.D."))))
(list (maintainer 'person #f "asd@asd.asd" "A.S.D." #f))
```

4.4.3 slots transformation

```
(xexpr->slots xexpr) → (listof slots?)
  xexpr : xexpr?
```

Converts a *xexpr* to `list` of `slots`.

Example:

```
> (xexpr->slots '(pkgmetadata (slots (slot ((name "1.9"))
                                          "Provides
libmypkg19.so.0"))))
(list (slots #f (list (slot "1.9" "Provides libmypkg19.so.0"))
#f))
```

4.4.4 stabilize-allarches transformation

```
(xexpr->stabilize-allarches xexpr) → boolean?
  xexpr : xexpr?
```

Converts a *xexpr* to `boolean`.

Example:

```
> (xexpr->stabilize-allarches '(pkgmetadata (stabilize-
allarches ())))
#t
```

4.4.5 upstream transformation

```
(xexpr->upstream xexpr) → upstream?
  xexpr : xexpr?
```

Converts a *xexpr* to `upstream`.

Example:

```
> (xexpr->upstream '(pkgmetadata (upstream
                                  (remote-
id ((type "gitlab")) "asd/asd")
                                  (remote-
id ((type "github")) "asd-org/asd"))))
```

```
(upstream
 '()
 #f
 #f
 #f
 (list (remote-id 'gitlab "asd/asd") (remote-id 'github "asd-
org/asd")))
```

4.4.6 use transformation

```
(xexpr->uses xexpr) → (listof use?)
  xexpr : xexpr?
```

Converts a *xexpr* to `list` of `use`s.

Example:

```
> (xexpr->uses '(pkgmetadata (use (flag ((name "racket"))
                                          "Build using dev-
scheme/racket"))))
(list (use #f (list (uflag "racket" "Build using dev-
scheme/racket"))))
```

4.4.7 metadata transformation

```
(xexpr->metadata xexpr) → metadata%
  xexpr : xexpr?
```

Converts a *xexpr* to `metadata%` object.

Example:

```
> (xexpr->metadata '(pkgmetadata))
(document
 (prolog
  '(#(struct:p-i
      #(struct:location 0 0 0)
      #(struct:location 0 0 0)
      xml
      "version=\"1.0\" encoding=\"UTF-8\""))
  (document-type
   'pkgmetadata
   (external-dtd/system "https://www.gentoo.org/dtd/metadata.dtd")
```

```
   #f)
  '())
 (element 'racket 'racket 'pkgmetadata '() '())
 '())
```

```
(read-metadata [port]) → metadata%
  port : input-port? = current-input-port
```

Converts input from *port* to metadata% object.

Examples:

```
> {define str "<?xml version=\"1.0\" encoding=\"UTF-
8\"?>\n<!DOCTYPE pkgmetadata SYSTEM \"https://www.gentoo.org/dtd/metadata.dtd\">\n<pkgmetada
> (display (read-metadata (open-input-string str)))
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "https://www.gentoo.org/dtd/metadata.dtd">

<pkgmetadata>
</pkgmetadata>
```

```
(read-metadata-file pth) → metadata%
  pth : path-string?
```

Converts contents of `file` from *pth* to metadata% object.

4.5 UnCache

```
(require ebuild/transformers/uncache)
              package: ebuild-transformers
```

```
(uncache port) → ebuild%
  port : input-port?
```

Consumes *port* and attempts to convert it to a ebuild% (racket object).

Certain to be missed are any additional shell script functions because cached ebuild files contain only special ebuild variables.

```
(uncache-file pth) → ebuild%
  pth : path-string?
```

Attempts to convert a cached ebuild file at *pth* to a ebuild% (racket object).

5 Ebuild Templates

5.1 Rust's Cargo

```
(require ebuild/templates/cargo)
```
package: ebuild-templates

```
ebuild-cargo-mixin : (class? . -> . class?)
```
argument extends/implements: ebuild%

```
ebuild-cargo% : class?
```
superclass: ebuild%

Pre-made class extending ebuild% for writing ebuilds using the cargo.eclass.

When creating a ebuild-cargo% object following values are automatically added to fields:

- "cargo" to "inherit"ed eclasses
- "$(cargo_crate_uris ${CRATES})" to SRC_URI
- CRATES to a CRATES variable

```
(new ebuild-cargo% [[CRATES CRATES]]) → (is-a?/c ebuild-
cargo%)
  CRATES : (listof string?) = '()
```

Examples:

```
> (define my-ebuild
    (new ebuild-cargo% [CRATES '("crate1" "crate2" "crate3")]))
> (display my-ebuild)
# Copyright 1999-2022 Gentoo Authors
# Distributed under the terms of the GNU General Public
License v2

EAPI=8

CRATES="
 crate1
 crate2
```

```
 crate3"

inherit cargo

DESCRIPTION="package"
HOMEPAGE="https://wiki.gentoo.org/wiki/No_homepage"
SRC_URI="$(cargo_crate_uris ${CRATES})"

LICENSE="all-rights-reserved"
SLOT="0"
KEYWORDS="~amd64"
```

5.2 Git Hosting snapshots

`(require ebuild/templates/gh)` package: `ebuild-templates`

`ebuild-gh-mixin : (class? . -> . class?)`
argument extends/implements: `ebuild%`

`ebuild-gh% : class?`
superclass: `ebuild%`

Pre-made class extending `ebuild%` for writing ebuilds using the gh.eclass.

```
(new ebuild-gh%
   [GH_REPO GH_REPO]
  [[GH_DOM GH_DOM]
   [GH_COMMIT GH_COMMIT]]) → (is-a?/c ebuild-gh%)
 GH_REPO : string?
 GH_DOM : string? = "gitlab.com"
 GH_COMMIT : (or/c #f string?) = #f
```

Examples:

```
> (define my-ebuild
    (new ebuild-gh%
         [GH_DOM "gitlab.com"]
         [GH_REPO "asd/asd"]
         [GH_COMMIT "b46c957f0ad7490bc7b0f01da0e80380f34cac2d"]))
> (display my-ebuild)
```

```
# Copyright 1999-2022 Gentoo Authors
# Distributed under the terms of the GNU General Public
License v2

EAPI=8

GH_DOM="gitlab.com"
GH_REPO="asd/asd"
GH_COMMIT="b46c957f0ad7490bc7b0f01da0e80380f34cac2d"

inherit gh

DESCRIPTION="package"
HOMEPAGE="https://wiki.gentoo.org/wiki/No_homepage"

LICENSE="all-rights-reserved"
SLOT="0"
KEYWORDS="~amd64"
```

5.3 Git repositories

```
(require ebuild/templates/git)
          package: ebuild-templates
```

```
ebuild-git-mixin : (class? . -> . class?)
  argument extends/implements: ebuild%
```

```
ebuild-git% : class?
  superclass: ebuild%
```

Pre-made class extending `ebuild%` for writing ebuilds using the git-r3.eclass.

```
(new ebuild-git%
    [EGIT_REPO_URI EGIT_REPO_URI]
   [[EGIT_BRANCH EGIT_BRANCH]
    [EGIT_CHECKOUT_DIR EGIT_CHECKOUT_DIR]
    [EGIT_COMMIT EGIT_COMMIT]
    [EGIT_COMMIT_DATE EGIT_COMMIT_DATE]
    [EGIT_MIN_CLONE_TYPE EGIT_MIN_CLONE_TYPE]
    [EGIT_MIRROR_URI EGIT_MIRROR_URI]
    [EGIT_SUBMODULES EGIT_SUBMODULES]
    [EVCS_OFFLINE EVCS_OFFLINE]])
→ (is-a?/c ebuild-git%)
 EGIT_REPO_URI : string?
 EGIT_BRANCH : (or/c #f string?) = #f
 EGIT_CHECKOUT_DIR : (or/c #f string?) = #f
 EGIT_COMMIT : (or/c #f string?) = #f
 EGIT_COMMIT_DATE : (or/c #f string?) = #f
 EGIT_MIN_CLONE_TYPE : (or/c #f string?) = #f
 EGIT_MIRROR_URI : (or/c #f string?) = #f
 EGIT_SUBMODULES : (or/c #f (listof string?)) = #f
 EVCS_OFFLINE : (or/c #f string?) = #f
```

Examples:

```
> (define my-ebuild
    (new ebuild-git%
         [EGIT_REPO_URI "https://gitlab.com/asd/asd.git"]
         [EGIT_BRANCH "trunk"]
         [EGIT_SUBMODULES '()]
         [KEYWORDS '()]))
> (display my-ebuild)
# Copyright 1999-2022 Gentoo Authors
# Distributed under the terms of the GNU General Public
License v2

EAPI=8

EGIT_REPO_URI="https://gitlab.com/asd/asd.git"
EGIT_BRANCH="trunk"
EGIT_SUBMODULES=()

inherit git-r3

DESCRIPTION="package"
HOMEPAGE="https://wiki.gentoo.org/wiki/No_homepage"

LICENSE="all-rights-reserved"
```

```
SLOT="0"
```

6 Ebuild Tools

6.1 Dispatcher (racket-ebuild)

`(require ebuild/tools/dispatcher)` package: `ebuild-tools`

6.1.1 About

Invoked from command-line as `racket-ebuild`.

Dispatcher calls a Racket-Ebuild sub-command given as 1st argument with rest leftover arguments, similar to how `git` calls it's sub-commands.

For example: when calling `racket-ebuild commit -s` the dispatcher will execute `racket-ebuild-commit -s`.

6.1.2 Console usage

- `-h` or `--help` — show help information with usage options
- `-V` or `--verbose` — show the version of this program

6.2 GitHost2Metadata

`(require ebuild/tools/githost2metadata)`
package: `ebuild-tools`

6.2.1 About

Create a PMS package metadata file from Git hosting service repository.

6.2.2 Console usage

- `--c` or `--create` — create (save) the metadata
- `--s` or `--show` — show (display) the metadata
- `--d` ⟨*directory-path*⟩ or `--dir` ⟨*directory-path*⟩ — directory where the created metadata file should be saved

- `-v` or `--verbose` — be verbose (detailed console output)
- `--q` ⟨*quiet*⟩ or `--package-version` ⟨*package-version*⟩ — be quiet (minimal/no console output)
- `-h` or `--help` — show help information with usage options
- `-V` or `--verbose` — show the version of this program

githost2metadata also expects two arguments given last in the command invocation. First is the git hosting domain (e.g.: github.com, gitlab.com, ...). Second is the repository path (e.g.: xgqt/racket-ebuild, gentoo/gentoo, ...).

6.3 UL2PKG

```
(require ebuild/tools/url2pkg)
```
package: `ebuild-tools`

6.3.1 About

Creates a package from given URL based on different heuristic tactics.

Quality: very experimental.

6.3.2 Console usage

- `--pn` ⟨*package-name*⟩ or `--package-name` ⟨*package-name*⟩ — package name
- `--pv` ⟨*package-version*⟩ or `--package-version` ⟨*package-version*⟩ — package version
- `--save` — save generated package
- `--show` — show generated package
- `-h` or `--help` — show help information with usage options
- `-V` or `--verbose` — show the version of this program

6.4 Interactive

```
(require ebuild/tools/interactive)
```
package: `ebuild-tools`

6.4.1 About

Interactively creates packages.

6.4.2 Console usage

Does not accept any supplied arguments/options (for now).

6.5 Clean Versions

```
(require ebuild/tools/clean-versions)
```
package: `ebuild-tools`

6.5.1 About

Cleans up old versions of ebuilds from a repository.

6.5.2 Console usage

- `-v` or `--verbose` — be verbose (detailed console output)
- `-q` or `--quiet` — be quiet (minimal/no console output)
- `-m` ⟨*number*⟩ or `--max` ⟨*number*⟩ — maximum number of ebuilds to keep
- `-r` ⟨*path*⟩ or `--repository` ⟨*path*⟩ — directory path to ebuild repository
- `-s` or `--simulate` — simulate, show only what would be deleted
- `-n` or `--no-simulate` — create ebuilds in a directory (defaults to current unless a directory is specified by `--repository`)
- `-h` or `--help` — show help information with usage options
- `-V` or `--verbose` — show the version of this program

6.6 Commit

```
(require ebuild/tools/commit)
```
package: `ebuild-tools`

6.6.1 About

Create a commit.

This utility is inspired by `repoman commit` and new utility mean to take place of `repoman - pkgdev commit`.

6.6.2 Console usage

- `--a` or `---all` — stage all changed/new/removed files
- `--u` or `---update` — stage all changed files
- `--b` ⟨*gentoo-bug/url*⟩ or `---bug` ⟨*gentoo-bug/url*⟩ — add "Bug" tag for a given Gentoo or upstream bug/URL
- `--c` ⟨*gentoo-bug/url*⟩ or `---closes` ⟨*gentoo-bug/url*⟩ — add "Closes" tag for a given Gentoo or upstream bug/URL
- `--d` or `---dry-run` — do not make commits
- `--e` or `---edit` — force editing the commit even if message is not empty
- `--n` or `---scan-nonfatal` — do not fail when scanning with "pkgcheck"
- `--m` ⟨*commit-message*⟩ or `---message` ⟨*commit-message*⟩ — specify the commit message
- `--s` or `---scan` — scan using "pkgcheck"
- `--B` ⟨*package-version*⟩ or `---bump` ⟨*package-version*⟩ — set the commit message to "bump to ⟨*package-version*⟩" (use when updating ebuilds)
- `--D` ⟨*package-version*⟩ or `---drop` ⟨*package-version*⟩ — set the commit message to "drop old ⟨*package-version*⟩" (use when removing ebuilds)
- `--M` ⟨*commit-message*⟩ or `---aux-message` ⟨*commit-message*⟩ — auxiliary commit message, auto-line-wrapped
- `--U` or `---update-manifests` — update Manifest files using "pkgdev"
- `--S` or `---sign` — sign the created commit
- `-h` or `--help` — show help information with usage options
- `-V` or `--verbose` — show the version of this program

6.7 Fix Head

`(require ebuild/tools/fix-head)` package: `ebuild-tools`

6.7.1 About

Fix the header of given Ebuild files.

6.7.2 Console usage

- `-h` or `--help` — show help information with usage options
- `-V` or `--verbose` — show the version of this program

6.8 PKGName

`(require ebuild/tools/pkgname)` package: `ebuild-tools`

6.8.1 About

Shows package name for a given directory.

6.8.2 Console usage

- `-h` or `--help` — show help information with usage options
- `-V` or `--verbose` — show the version of this program

Also takes any number of of arguments not followed by flags that specify system paths to be passed to "pkgname".

7 External Resources

- General
  - Gentoo project page
  - Gentoo Wiki
  - Gentoo Development Manual
- Ebuilds
  - Ebuild documentation
  - Ebuild Functions
  - Ebuild Repository documentation
  - Ebuild Repository Format
  - Selected Ebuild Repositoriess
    - Official Gentoo ebuild repository (Gentoo's GitWeb)
    - Official Gentoo ebuild repository (GitHub)
    - Experimental Racket Gentoo Overlay
- Metadata
  - Package metadata documentation
  - Gentoo XML DTD of package metadata
  - Racket XML documentation
- Racket
  - Classes
  - XML

Index