Racket-Ebuild

Version 8.5

Maciej Barć <xgqt@riseup.net>

May 9, 2022

Library to ease automatic ebuild creation and maintenance.

Version: 14.0.0, commit hash: 3347176

Contents

1 About 6

1.1 Project Aim 6

1.2 Development 6

1.2.1 Tools 6

1.2.2 Weirdness 6

1.3 Upstream 6

1.4 License 7

2 Ebuild - Classes 8

2.1 Ebuild Class 8

2.2 Metadata Class 10

2.3 Package Class 12

2.4 Repository Class 14

3 Ebuild - Exported Functions 17

3.1 Ebuild Functions 17

3.1.1 Constraint Flag 17

3.1.2 Source URL 18

3.1.3 Shell Variables 19

3.1.4 Object manipulation 21

3.2 Shell Script Functions 22

3.2.1 POSIX shell script function creation 22

3.3 Metadata Functions 23

3.3.1 XML tags 23

3.3.2 localized 23

3.3.3 longdescription 24
3.3.4 maintainer 24
3.3.5 slots 25
3.3.6 upstream 26
3.3.7 use 27
3.4 Package Functions 28
3.4.1 Package Version 28
3.5 Repository Functions 30
3.5.1 Layout 30
3.6 Manifest Functions 31
3.6.1 Checksum 32
3.6.2 Dist 32
3.6.3 Manifest 33

4 Ebuild Transformers 35

4.1 GitHost 35
4.2 SRC_URI Transformers 36
4.3 CFlag Transformers 37
4.4 Metadata Transformers 37
4.4.1 longdescription transformation 37
4.4.2 maintainer transformation 38
4.4.3 slots transformation 38
4.4.4 stabilize-allarches transformation 38
4.4.5 upstream transformation 39
4.4.6 use transformation 39
4.4.7 metadata transformation 39

4.5 UnCache 40

5 Ebuild Templates 42

5.1 Rust's Cargo 42

5.2 Git Hosting snapshots 43

5.3 Git repositories 44

6 Ebuild Tools 47

6.1 Dispatcher (racket-ebuild) 47

6.1.1 About 47

6.1.2 Console usage 47

6.2 GitHost2Metadata 47

6.2.1 About 47

6.2.2 Console usage 47

6.3 UL2PKG 48

6.3.1 About 48

6.3.2 Console usage 48

6.4 Interactive 48

6.4.1 About 49

6.4.2 Console usage 49

6.5 Clean Versions 49

6.5.1 About 49

6.5.2 Console usage 49

6.6 Commit 49

6.6.1 About 50

6.6.2 Console usage 50

6.7 Fix Head 51

6.7.1 About 51

6.7.2 Console usage 51

6.8 PKGName 51

6.8.1 About 51

6.8.2 Console usage 51

7 External Resources **52**

Index **53**

Index **53**

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 strings 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-version`s 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-version`s as `string`s 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-versions` 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)
CATEGORY/PN: app-misc/unknown
PV: 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"
METADATA:
<?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
Repository packages:
CATEGORY/PN: sys-devel/asd
PV: 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"
METADATA:
<?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
```

3.6 Manifest Functions

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

This library does not use classes but rather structs. It is implemented in Typed Racket.

This library allows reading and creating Manifests.

Manifest files hold digests and size data for every file used by the package. More info about Manifests can be found in the Gentoo Developer manual.

3.6.1 Checksum

```
(struct checksum (type hash)
    #:extra-constructor-name make-checksum)
  type : string?
  hash : string?
```

Example:

```
> (checksum "BLAKE2B" "aa81a1a")
(checksum "BLAKE2B" "aa81a1a")
```

```
(list->checksums lst) → (listof checksum?)
  lst : (listof string?)
```

Example:

```
> (list->checksums '("BLAKE2B" "aa81a1a" "SHA521" "b6095d4"))
(list (checksum "BLAKE2B" "aa81a1a") (checksum "SHA521"
"b6095d4"))
```

```
(checksum->string cs) → string?
  cs : checksum?
```

Example:

```
> (checksum->string (checksum "BLAKE2B" "aa81a1a"))
"BLAKE2B aa81a1a"
```

3.6.2 Dist

```
(struct dist (type file size checksums)
    #:extra-constructor-name make-dist)
  type : (or/c 'dist 'ebuild 'misc)
  file : string?
  size : integer?
  checksums : (listof checksum?)
```

Example:

```
> (dist 'dist "package-source.tar.gz" 100 (list (checksum "BLAKE2B" "aa81a1a")))
(dist 'dist "package-source.tar.gz" 100 (list (checksum "BLAKE2B"
"aa81a1a")))
```

```
(string->dist-type str) → (or/c 'dist 'ebuild 'misc)
  str : string?
```

Example:

```
> (string->dist-type "EBUILD")
'ebuild
```

```
(dist-type->string dist-type) → string?
  dist-type : (or/c 'dist 'ebuild 'misc)
```

Example:

```
> (dist-type->string 'misc)
"MISC"
```

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

Example:

```
> (string->dist "DIST gcc-9.4.0.tar.xz 72411232 BLAKE2B 4bb000d
SHA512 dfd3500")
(dist
 'dist
 "gcc-9.4.0.tar.xz"
 72411232
 (list (checksum "BLAKE2B" "4bb000d") (checksum "SHA512"
"dfd3500")))
```

```
(dist->string dst) → string?
  dst : dist?
```

3.6.3 Manifest

```
(struct manifest (dists)
    #:extra-constructor-name make-manifest)
  dists : (listof dist?)
```

```
(read-manifest path) → manifest?
  path : path-string?
```

```
(write-manifest mnfst [out]) → void?
  mnfst : manifest?
  out : output-port? = (current-output-port)
```

```
(save-manifest mnfst path) → void?
  mnfst : manifest?
  path : path-string?
```

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 "gitlab.com" "asd/asd")
```

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

Converts a `ur` `url` to `githost`.

Example:

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

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

Converts a `str` `string` to `githost`.

Example:

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

```
(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 `cflag`s.

Example:

```
> (string->cflag "X? ( x11-libs/libX11 ) qt5? ( dev-qt/qtsvg dev-
qt/qtwidgets )")
(list
 (cflag "X?" '("x11-libs/libX11"))
 (cflag "qt5?" '("dev-qt/qtsvg" "dev-qt/qtwidgets")))
```

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

About, 6
About, 49
About, 50
About, 47
About, 51
About, 47
About, 49
About, 51
About, 48
append!, 9
as-array-variable, 21
as-variable, 20
cflag, 17
CFlag Transformers, 37
cflag->string, 18
cflag-content, 17
cflag-predicate, 17
cflag?, 17
Checksum, 32
checksum, 32
checksum->string, 32
checksum-hash, 32
checksum-type, 32
checksum?, 32
Clean Versions, 49
Commit, 49
concat!, 10
Console usage, 49
Console usage, 50
Console usage, 47
Console usage, 51
Console usage, 47
Console usage, 49
Console usage, 51
Console usage, 48
Constraint Flag, 17
create, 9
create, 11
create-list, 9
default-layout, 30
Development, 6
Dispatcher (racket-ebuild), 47
Dist, 32
dist, 32
dist->string, 33
dist-checksums, 32
dist-file, 32
dist-size, 32
dist-type, 32
dist-type->string, 33
dist?, 32
doc, 26
doc?, 26
docs->xexpr, 26
ebuild, 8
Ebuild - Classes, 8
Ebuild - Exported Functions, 17
Ebuild Class, 8
Ebuild Functions, 17
Ebuild Templates, 42
Ebuild Tools, 47
Ebuild Transformers, 35
new, 8
ebuild%, 8
ebuild-append!, 21
new, 42
ebuild-cargo%, 42
ebuild-cargo-mixin, 42
ebuild-concat!, 21
new, 43
ebuild-gh%, 43
ebuild-gh-mixin, 43
new, 45
ebuild-git%, 44
ebuild-git-mixin, 44
ebuild/ebuild, 8
ebuild/manifest, 31
ebuild/metadata, 10
ebuild/package, 12
ebuild/repository, 14
ebuild/sh-function, 22
ebuild/templates/cargo, 42

ebuild/templates/gh, 43
ebuild/templates/git, 44
ebuild/tools/clean-versions, 49
ebuild/tools/commit, 49
ebuild/tools/dispatcher, 47
ebuild/tools/fix-head, 51
ebuild/tools/githost2metadata, 47
ebuild/tools/interactive, 48
ebuild/tools/pkgname, 51
ebuild/tools/url2pkg, 48
ebuild/transformers/cflag, 37
ebuild/transformers/githost, 35
ebuild/transformers/metadata, 37
ebuild/transformers/src-uri, 36
ebuild/transformers/uncache, 40
External Resources, 52
Fix Head, 51
get-CATEGORY/PN, 12
get-Ps, 13
get-PVs, 12
get-versions, 12
Git Hosting snapshots, 43
Git repositories, 44
GitHost, 35
githost, 35
githost->string, 35
githost->url, 36
githost-domain, 35
githost-repository, 35
GitHost2Metadata, 47
githost?, 35
Interactive, 48
Layout, 30
layout, 30
layout->string, 31
layout-cache-formats, 30
layout-eapis-banned, 30
layout-eapis-deprecated, 30
layout-manifest-hashes, 30
layout-manifest-required-hashes, 30
layout-masters, 30
layout-sign-commits, 30
layout-sign-manifests, 30
layout-string, 14
layout-thin-manifests, 30
layout-update-changelog, 30
layout?, 30
License, 7
list->checksums, 32
list-as-array-variable, 20
list-as-variable, 19
live-version, 29
localized, 23
localized, 23
localized->xexpr, 23
localized-data, 23
localized-lang, 23
localized?, 23
longdescription, 24
longdescription, 24
longdescription transformation, 37
longdescription->xexpr, 24
longdescription?, 24
maintainer, 24
maintainer, 24
maintainer transformation, 38
maintainer->xexpr, 24
maintainer-description, 24
maintainer-email, 24
maintainer-name, 24
maintainer-proxied, 24
maintainer-type, 24
maintainer?, 24
make-array-variable, 21
make-cflag, 17
make-checksum, 32
make-dist, 32
make-doc, 26
make-githost, 35
make-layout, 30
make-localized, 23
make-longdescription, 24
make-maintainer, 24
make-manifest, 33

make-package-version, 29
make-remote-id, 26
make-script, 22
make-sh-function, 22
make-slot, 25
make-slots, 25
make-src-uri, 18
make-uflag, 27
make-upstream, 26
make-upstreammaintainer, 26
make-use, 28
make-variable, 20
Manifest, 33
manifest, 33
Manifest Functions, 31
manifest-dists, 33
manifest?, 33
Metadata Class, 10
Metadata Functions, 23
metadata transformation, 39
Metadata Transformers, 37
new, 11
metadata%, 10
metadata-empty-tags, 23
Object manipulation, 21
Package Class, 12
Package Functions, 28
Package Version, 28
new, 12
package%, 12
package-version, 29
package-version->string, 29
package-version-patch, 29
package-version-phase, 29
package-version-pre, 29
package-version-rc, 29
package-version-release, 29
package-version-revision, 29
package-version?, 29
PKGName, 51
POSIX shell script function creation, 22
Project Aim, 6
Racket-Ebuild, 1
read-manifest, 33
read-metadata, 40
read-metadata-file, 40
release?, 28
remote-id, 26
remote-id-tracker, 26
remote-id-type, 26
remote-id?, 26
Repository Class, 14
Repository Functions, 30
new, 14
repository%, 14
Rust's Cargo, 42
save, 15
save, 14
save, 9
save, 11
save-layout, 14
save-manifest, 34
save-name, 15
save-packages, 15
sh-function, 22
sh-function->string, 22
sh-function-body, 22
sh-function-name, 22
sh-function?, 22
Shell Script Functions, 22
Shell Variables, 19
show, 13
show, 14
simple-version, 29
slot, 25
slot-data, 25
slot-name, 25
slot?, 25
slots, 25
slots, 25
slots transformation, 38
slots->xexpr, 25
slots-lang, 25
slots-slotlist, 25

slots-subslots, 25
slots?, 25
Source URL, 18
src-uri, 18
src-uri->string, 19
src-uri-flag, 18
src-uri-name, 18
src-uri-src, 19
src-uri-url, 18
src-uri?, 18
SRC_URI Transformers, 36
stabilize-allarches transformation, 38
string->cflag, 37
string->dist, 33
string->dist-type, 33
string->githost, 35
struct:cflag, 17
struct:checksum, 32
struct:dist, 32
struct:doc, 26
struct:githost, 35
struct:layout, 30
struct:localized, 23
struct:longdescription, 24
struct:maintainer, 24
struct:manifest, 33
struct:package-version, 29
struct:remote-id, 26
struct:sh-function, 22
struct:slot, 25
struct:slots, 25
struct:src-uri, 18
struct:uflag, 27
struct:upstream, 26
struct:upstreammaintainer, 26
struct:use, 28
Tools, 6
uflag, 27
uflag-data, 27
uflag-name, 27
uflag?, 27
UL2PKG, 48
UnCache, 40
uncache, 40
uncache-file, 41
unroll-sh-functions, 22
Upstream, 6
upstream, 26
upstream, 26
upstream transformation, 39
upstream->xexpr, 26
upstream-bugs-to, 26
upstream-changelog, 26
upstream-doc, 26
upstream-maintainers, 26
upstream-remote-ids, 26
upstream?, 26
upstreammaintainer, 26
upstreammaintainer-email, 26
upstreammaintainer-name, 26
upstreammaintainer-status, 26
upstreammaintainer?, 26
url->githost, 35
url->src-uri, 36
use, 27
use, 28
use transformation, 39
use->xexpr, 28
use-flags, 28
use-lang, 28
use?, 28
Weirdness, 6
write-manifest, 34
xexpr->longdescriptions, 37
xexpr->maintainers, 38
xexpr->metadata, 39
xexpr->slots, 38
xexpr->stabilize-allarches, 38
xexpr->upstream, 39
xexpr->uses, 39
XML tags, 23