
enstaller Documentation

Release 4.6.3

Ilan Schnell

July 15, 2015

1	User Guide	3
1.1	Installation	3
1.2	Configuration	3
1.3	YAML configuration	4
1.4	Additional tools	5
2	Dev Guide	7
2.1	Examples	7
3	Reference	11
3.1	Dev guide for transitioning to 4.7.0 API	11
3.2	API documentation	13
3.3	File formats reference	16
	Python Module Index	21

Enstaller is Enthought's package manager.

This part of the documentations describes the usage of `enpkg` and `egginst`, the main tools available from `enstaller`.

1.1 Installation

1.1.1 Installing a released version

To install `enstaller`, simply executes the `bootstrap.py` script:

```
$ python bootstrap.py
enstaller-4.7.6-py2.7.egg           [installing egg]
  4.34 MB [.....]
```

1.1.2 Installing from sources

While you can install with the usual `setup.py install` dance, it is advised to build `enstaller` and install it through the `bootstrap` script:

```
$ python setup.py bdist_egg # build dist/enstaller-<version>.egg
$ python scripts/bootstrap.py dist/enstaller-<version>.egg
```

Note: you can safely reinstall various versions of `enstaller` by re-executing the `bootstrap` script, as it ensures the old `enstaller` is removed before installing the new one.

1.1.3 Testing the installation

The main CLI tool available is `enpkg`, which is used to install packages with their dependencies:

```
enpkg <requirement>
```

`enpkg` uses the file `~/.enstaller4rc` to store credentials, and various configuration settings.

1.2 Configuration

When used from the command line, `enpkg` looks for the file `.enstaller4rc` in the following locations:

- *sys.prefix*
- *sys.real_prefix* (if under a virtual environment)
- the user's home directory

The first found file is considered.

1.2.1 Examples

A simple example is as follows:

```
# To use an http proxy
proxy = 'http://john:doe@acme.com:3128'

# If auth has been set up by `--userpass`
EPD_auth = 'ZGF2aWRjQGVudGhvdWdodC5jb206Xl5kNHYxZGMhIw=='

# To disable SSL verification (insecure)
verify_ssl = False
```

To use custom legacy repositories:

```
# Disable canopy mode
use_webservice = False

# Set up local repositories
IndexedRepos = [
    'http://www.enthought.com/repo/GPL/eggs/{PLATFORM}/',
    'http://www.enthought.com/repo/free/{PLATFORM}/',
    'http://www.enthought.com/repo/commercial/{PLATFORM}/',
]
```

To use API token against a *brood* store:

```
store_url = "brood+https://brood.acme.com"
api_token = "<api_token>"
```

1.3 YAML configuration

Enstaller now supports a new configuration format, using the YAML syntax. Currently, you have to explicitly specify the configuration file path to use this format:

```
enpkg -c <...>/enstaller.yaml ...
```

The YAML-based configuration is more consistent, and simpler to read/write as a human than the legacy format.

Note: this format is still experimental, and may change without notice.

1.3.1 Basic example

This is a fairly complete example of available settings in the YAML format:


```
# Which server to use to connect to
store_url: "https://api.enthought.com"

# Whether to enforce SSL CA verification
verify_ssl: false

# List of <organization>/<repository>
repositories:
  - "enthought/commercial"
  - "enthought/free"
  # Local FS repository -- should be a directory, and an index.json is expected
  # in that directory
  - "file:///foo"

# Authentication
authentication:
  api_token: <your_api_token>

# Directory to use to cache eggs
files_cache: "~/.cached_eggs/{PLATFORM}"
```

1.3.2 Authentication

One can select plain-text authentication (insecure):

```
# Authentication
authentication:
  kind: simple
  username: <your_username>
  password: <your_password>
```

or token-based:

```
authentication:
  # kind is optional as api_token is the default
  kind: token
  api_token: <your_token>
```

Tokens should still be hold secret, but are more secure than password because you can revoke tokens.

1.4 Additional tools

Enstaller contains a set of tools beyond enpkg. We describe here how to use those.

Note: Those tools are not meant to be used outside Enthought. We make no stability or even existence guarantee.

Note: repack is a rewrite of endist' build_egg -r option. Endist is deprecated, please contact the build team @ Enthought if you miss some features of endist not in repack.

1.4.1 Repack

Repack is a tool to repack eggs built by setuptools into a format understood by enstaller. While enstaller can install any standard egg, converting them to the Enthought format allows you to specify dependencies (including non-egg dependencies), etc...

Simple usage:

```
# Repack the egg foo-1.0.0-py2.7.egg, with the build number 1
python -m enstaller.tools.repack -b 1 foo-1.0.0-py2.7.egg
```

By default, the tool will try to detect your platform and set the egg to this platform. If this fails, or if you want to set the platform manually, you should use the `-a` flag:

```
python -m enstaller.tools.repack -b 1 -a win-32 foo-1.0.0-py2.7.egg
```

Customizing metadata

The repack tool supports customization of egg metadata through the `endist.dat` file. The `endist.dat` is actually `exec-ed`, and the following variables are understood:

```
# endist.dat

# To override the package name
name = "foo"
# To override the version
version = "1.2.3"
# To override the build
build = 2

# To add runtime dependencies
packages = [
    "bar 1.2",
    "fubar 1.3",
]

# To add files not in the original egg
# The format is a list of triples (dir_path, regex, archive_dir),
# where regex is the regular expression to match for files in
# dir_path, to put in the "EGG-INFO/foo" directory inside the egg.
add_files = [("foo", "*.txt", "EGG-INFO/foo")]
```

This part of the documentation explains the main concepts of enstaller from a developer POV, that is people willing to use/extend enstaller in their application.

Enstaller is currently being rewritten as a library with enpkg being a CLI on top of it, so the API is still in flux.

2.1 Examples

This section highlights the main APIs through some simple examples.

2.1.1 Package search/listing

Most search, listing operations in enstaller are done through *Repository* instances, which are containers of package metadata. For example, to list every egg installed in sys.prefix:

```
from enstaller import Repository

repository = Repository._from_prefixes([sys.prefix])
for package in repository.iter_packages():
    print("{0}-{1}".format(package.name, package.version))
```

one can also list the most recent version for each package:

```
for package in repository.iter_most_recent_packages():
    print("{0}-{1}".format(package.name, package.version))
```

Repository instances are “dumb” containers, and don’t handle network connections, authentication, etc... A simple way to create a “real” repository is to start from a set of eggs:

```
from enstaller import Repository, RepositoryPackageMetadata

repository = Repository()
for path in glob.glob("*.egg"):
    package = RepositoryPackageMetadata.from_egg(path)
    repository.add_package(package)
```

We will later look into creating repositories from old-style repositories (as created by epd-repo) or brood repositories. The simplicity of repositories allows loose-coupling between operations on a repository and the package metadata origin.

2.1.2 Connecting and authenticating

Http connections are handled through `Session` objects. To start a session, one may simply do:

```
from enstaller import Configuration, Session

configuration = Configuration(auth=("username", "password"))

session = Session.authenticated_from_configuration(configuration)
```

`Session` are thin wrappers around requests' `Session`. Its main features over requests' `Session` are etag handling, file:// uri handling, pluggable authentication method as well as integration with `Configuration` instances for settings (proxy, etc...).

In addition to head/get/post methods, `Session` instances have a slightly higher-level download method, which enables streaming and raises an exception if an HTTP error occurs, and is robust against stalled/cancelled downloads:

```
# target is the path for the created file. Will not exist if download fails
# (including cancelled by e.g. `Ctrl+C`).
target = session.download(some_url)
```

Delayed authenticated sessions

If one needs to authenticate the session later than creation time, e.g. if the auth information is set up in the configuration, that's possible as follows:

```
from enstaller import Configuration, Session

configuration = Configuration()
session = Session.from_configuration(configuration)

# Prompt the user for authentication, etc...
...

configuration.update(auth=("username", "password"))
session.authenticate(configuration.auth)
```

2.1.3 Creating remote repositories

To create repositories from http indices:

```
from enstaller import Configuration, Repository, Session

config = Configuration._from_legacy_locations()
repository_info = config.repositories[0]

session = Session.from_configuration(config)
session.authenticate(config.auth)

remote_repository = Repository.from_repository_info(session,
                                                    repository_info)

# Same, with etag-based caching
with session.etag():
    remote_repository = Repository.from_repository_info(session,
                                                        repository_info)
```

Note: this works for both use_webservice enabled and disabled, and for brood repositories

2.1.4 Solving dependencies

The dependency solver has a simple API to resolve dependencies:

```
from enstaller.solver import Request, Requirement, Solver

# represents the set of packages available
remote_repository = Repository(...)
# represents the set of packages currently installed
installed_repository = Repository(...)

solver = Solver(remote_repository, installed_repository)

request = Request()
request.install(Requirement.from_anything("numpy"))
request.install(Requirement.from_anything("ipython"))

# actions are (opcode, egg) pairs
# WARNING: this is likely to change
actions = solver.resolve(request)
```

Note: actions returned by the solver are only of the install/remove type, fetching is handled outside the solver.

2.1.5 Executor

3.1 Dev guide for transitioning to 4.7.0 API

As the pre 4.7.0 API was fundamentally tied to module-level globals, we had to change the internal enstaller API in a fundamental way. This guide explains how to transition to the new API.

3.1.1 Configuration

The `enstaller.config` module has been completely revamped:

- The main API is now the `Configuration` class. Multiple instances of this class can coexist in the same process (though one may need to be careful about keyring-related interactions).
- Most individual methods of the module have been removed, and the information need to be accessed through the configuration object instead.

Creating configuration instances

The most common way to create a configuration is to start from an existing configuration file:

```
config = Configuration.from_file(filename)
print(config.use_webservice)
```

One can also create a configuration from the default location:

```
config = Configuration._from_legacy_locations()
```

This API may be revised as we change the location logic (the current logic made sense for an EPD-like setup, but does not anymore with `virtualenvs`).

Note: this method will fail if no `.enstaller4rc` is found. To create a default `enstaller4rc`, you should use the `write_default_config()` function first:

```
filename = ...
if not os.path.exists(filename):
    write_default_config(filename)

config = Configuration.from_file(filename)
```

Replacing get_auth

As the `get_auth` function was implicitly sharing global state, it has been removed. Instead of:

```
# Obsolete
from enstaller.config import get_auth
print(get_auth())
```

one should use:

```
config = Configuration._from_legacy_locations()
print(config.auth)
```

Note: the authentication may not be setup, in which case `config.auth` may return an invalid configuration. To check whether the authentication is valid, use the `is_auth_configured` property:

```
config = Configuration._from_legacy_locations()
if config.is_auth_configured:
    print(config.auth)
```

Changing authentication

Enstaller does not use keyring anymore to hold password securely. Instead, the password is now always stored insecurely in the `EPD_auth` setting inside `.enstaller4rc`.

For backward compatibility, `enpkg` will convert password stored in the keyring back into `.enstaller4rc` automatically. To avoid keyring issues, keyring is bundled inside `enstaller`.

We advise *not* to change authentication directly in `.enstaller4rc`, as changing configuration file is user-hostile. Instead, applications using `enstaller` library should store the authentication themselves, and set it up inside the `Configuration` object through the `set_auth()` method.

The private methods `Configuration._change_auth` and `Configuration._checked_change_auth` are there for convenience, but their usage is discouraged.

Note: while keeping the password in clear is insecure, this is actually more secure as `enstaller` would before implicitly change from clear to secure and vice et versa depending on whether keyring was available to `enstaller`. The midterm solution is to use token-based authentication and never store password, but this will need some support server-side before being deployed.

Removed config module functions

The following functions have been removed:

- `clear_auth`: obsolete with keyring removal
- `clear_cache`: there is no configuration state anymore, just use a new `Configuration` instance.
- `get_repository_cache`: use `Configuration.repository_cache` attribute
- `get`: use corresponding `Configuration` attributes instead
- `read`: use `Configuration` instance and its attributes
- `web_auth`: use `authenticate()` instead
- `write`: use the `write()` method instead

3.1.2 Repositories and package metadata

Most of the store-related functionalities are now available through the *Repository* class. See repository-guide-label for more information.

3.2 API documentation

3.2.1 Configuration

`class enstaller.config.Configuration (**kw)`

api_url

Url to hit to get user information on api.e.com.

auth

The auth object that may be passed to Session.authenticate

Returns the auth instance, or None is configured.

Return type IAuth or None

autoupdate

Whether enpkg should attempt updating itself.

filename

The filename this configuration was created from. May be None if the configuration was not created from a file.

classmethod from_file (filename)

Create a new Configuration instance from the given file.

filename: str or file-like object If a string, is understood as a filename to open. Understood as a file-like object otherwise.

indexed_repositories

List of (old-style) repositories. Only actually used when use_webservice is False.

indices

Returns a list of pair (index_url, store_location) for this given configuration.

Takes into account webservice/no webservice and pypi True/False

Deprecated since version 4.9.0: Use the repositories property instead

max_retries

Max attempts to retry an http connection or re-fetching data whose checksum failed.

noapp

Ignore appinst entries.

prefix

Prefix in which enpkg operates.

proxy

A ProxyInfo instance or None if no proxy is configured.

proxy_dict

A dictionary <scheme>:<proxy_string> that can be used as the proxies argument for requests.

repository_cache

Absolute path where eggs will be cached.

set_repositories_from_names (*names*)

Set repositories from their names alone, e.g. 'enthought/free'.

Parameters **names** (*list*) – List of repository names

store_kind

Store kind (brood, legacy canopy, old-repo style).

store_url

The store url to hit for indices and eggs.

update (***kw*)

Set configuration attributes given as keyword arguments.

use_pypi

Whether to load pypi repositories (in *webservice* mode).

use_webservice

Whether to use canopy legacy or not.

verify_ssl

Whether to verify SSL CA or not.

webservice_entry_point

Whether to fetch indices and data (in *webservice* mode).

write (*filename*)

Write this configuration to the given filename.

Parameters **filename** (*str*) – The path of the written file.

enstaller.config.write_default_config (*filename*)

Write a default configuration file at the given location.

Will raise an exception if a file already exists.

Parameters **filename** (*str*) – The location to write to.

3.2.2 Repository

class enstaller.repository.Repository (*packages=None*)

A Repository is a set of package, and knows about which package it contains.

delete_package (*package_metadata*)

Remove the given package.

Removing a non-existent package is an error.

Parameters **package_metadata** (*PackageMetadata*) – The package to remove

find_latest_package (*name*)

Returns the latest package with the given name.

Parameters **name** (*str*) – The package's name

Returns **package** (*PackageMetadata*)

find_package (*name, version*)

Search for the first match of a package with the given name and version.

Parameters

- **name** (*str*) – The package name to look for.
- **version** (*str*) – The full version string to look for (e.g. ‘1.8.0-1’).

Returns **package** (*RemotePackageMetadata*) – The corresponding metadata.

find_package_from_requirement (*requirement*)

Search for latest package matching the given requirement.

Parameters **requirement** (*Requirement*) – The requirement to match for.

Returns **package** (*RemotePackageMetadata*) – The corresponding metadata.

find_packages (*name, version=None*)

Returns a list of package metadata with the given name and version, sorted from lowest to highest version.

Parameters

- **name** (*str*) – The package’s name
- **version** (*str or None*) – If not None, the version to look for

Returns **packages** (*iterable*) – Iterable of RemotePackageMetadata-like (order is unspecified)

find_sorted_packages (*name*)

Returns a list of package metadata with the given name and version, sorted from lowest to highest version.

Parameters

- **name** (*str*) – The package’s name
- **version** (*str or None*) – If not None, the version to look for

Returns **packages** (*iterable*) – Iterable of RemotePackageMetadata-like (order is unspecified)

Deprecated since version 4.9: Use **:method:‘find_packages‘**.

classmethod from_repository_info (*session, repository_info*)

Create a repository from a remote index.

Parameters

- **session** (*Session*) – A session instance (must be authenticated)
- **repository_info** (*IRepositoryInfo*) – The metadata for the repository to fetch.

has_package (*package_metadata*)

Returns True if the given package is available in this repository

Parameters **package_metadata** (*PackageMetadata*) – The package to look for.

Returns **ret** (*bool*) – True if the package is in the repository, false otherwise.

iter_most_recent_packages ()

Iter over each package of the repository, but only the most recent version of a given package

Returns **packages** (*iterable*) – Iterable of the corresponding RemotePackageMetadata-like instances.

iter_packages ()

Iter over each package of the repository

Returns **packages** (*iterable*) – Iterable of RemotePackageMetadata-like.

3.2.3 Session

class `enstaller.session.Session`(*authenticator*, *cache_directory*, *proxies=None*, *verify=True*,
max_retries=0)

Simple class to handle http session management

It also ensures connection settings such as proxy, SSL CA certification, etc... are handled consistently).

Parameters

authenticator [IAuthManager] An authenticator instance

cache_directory [str] A writeable directory to cache data and eggs.

proxies [dict] A proxy dict as expected by requests (and provided by Configuration.proxy_dict).

verify [bool] If True, SSL CA are verified (default).

classmethod `authenticated_from_configuration` (*configuration*)

Create a new authenticated session from a configuration.

Parameters `configuration` ([Configuration](#)) – The configuration to use.

download (*url*, *target=None*)

Safely download the content at the give url.

Safely here is understood as not leaving stalled content if the download fails or is canceled. It uses streaming as so not to use too much memory.

classmethod `from_configuration` (*configuration*)

Create a new session from a configuration.

Parameters `configuration` ([Configuration](#)) – The configuration to use.

3.3 File formats reference

This section documents the various file formats used/generated by enstaller, such as our custom egg format.

3.3.1 Enstaller egg format

Our egg format is an extension of the existing setuptools's egg.

A few notations:

- `$PREFIX`: is understood as the prefix of the current python. In a standard install, `$PREFIX/bin/python` will be the python binary on unix, `$PREFIX/python.exe` on windows.
- `$BINDIR`: where 'binaries' are installed. Generally `$PREFIX/bin` on Unix, `$PREFIX\Scripts` on windows.
- `$METADIR`: package-specific directory where files/metadata get installed. Generally `$PREFIX/EGG-INFO/$package_name`
- `basename(path)`: the basename of that path (as computed by `os.path.basename`)

Metadata

All the metadata are contained within the EGG-INFO subdirectory.

This subdirectory contains all the metadata needed by egginst to install an egg properly. Those are set within different free-format text files:

- EGG-INFO/info.json
- EGG-INFO/inst/appinst.dat
- EGG-INFO/inst/files_to_install.txt
- EGG-INFO/spec/depend
- EGG-INFO/spec/lib-depend
- EGG-INFO/spec/lib-provide

info.json

Only eggs built through build-egg (in endist) contain that file.

The code to write this file is in endist.build_egg, and used to build pypi eggs as well.

The following fields *may* be present:

- name: the package name
- version: the version string
- build: the build # (as an int). Must be ≥ 0 .
- arch: string or null. Seems to be only 'x86', 'amd64' or null, although this is not currently enforced.
- platform: string or null. Seems to be only sys.platform or null, although this is not currently enforced.
- osdist: string or null. The string can be a bit of anything, you should not rely on it.
- python: string or null. major.minor format (e.g. '2.7'), though not enforced either.
- packages: a list of dependencies. Name and version are *space* separated. The version part is actually optional.

Also enapp-related metadata (also optional):

- app: boolean. Used by egginst to determine whether to deal with enapp features or not.
- app_entry: string. Used by egginst to create entry points. Not sure what happens when this conflicts with setuptools entry point.
- app_icon_data: ?
- app_icon_data: string. The filename of the icon (the icon is expected to be in \$METADIR)

Note: if both this file and EGG-INFO/spec/depend are present, then info.json overrides the attributes set in spec/depend (see egginst.eggmeta.info_from_z).

inst subdirectory

appinst.dat A python script that is used by 'applications' during the install process. Is generally defined in the recipe files directory (as appinst.dat), and explicitly included in our eggs through workbench.eggcreator.EggCreator.add_appinst_dat()

Mostly used for setting up application shortcuts.

files_to_install.txt This file is used to define so-called proxies (a clumsy way to emulate softlinks on windows) and support softlinks on non-windows platform. The file defines one entry per line, and each entry is a space separated set of two items.

On linux and os x, each entry looks as follows:

EGG-INFO/usr/lib/libzmq.so	libzmq.so.0.0.0
----------------------------	-----------------

This defines a link join(prefix, 'lib/libzmq.so') to libzmq.so.0.0.0. More precisely:

- the left part is used to define the link name, the right part the target of the link.
- the actual link name will be a join of the prefix + the part that comes after EGG-INFO/usr.

Entries may also look as follows:

EGG-INFO/usr/bin/xslt-config	False
------------------------------	-------

This does not define a link to False, but instead tells egginst to ignore this entry.

A third format only encountered on windows' eggs:

{TARGET} {ACTION}

where {TARGET} must be in the zip archive, and where {ACTION} may be one of the following:

- **PROXY**: a proxy to the left part is created. A proxy is a set of two files, both written in the \$BINDIR
 - one small exe which is a copy of the setuptools' cli.exe, renamed to basename({TARGET}).
 - another file {TARGET_NO_EXTENSION}-script.py where TARGET_NO_EXTENSION = basename(splitext({TARGET}))
- Anything else: understood as a directory. In that case, {TARGET} will be copied into \$PREFIX\{ACTION}\basename({TARGET})

A **PROXY** example:

EGG-INFO/usr/bin/ar.exe	PROXY
-------------------------	-------

Egginst will create the following:

<pre># A copy of cli.exe \$BINDIR\\ar.exe # the python script called by \$BINDIR\\ar.exe, itself calling # \$METADIR\\usr\\bin\\ar.exe \$BINDIR\\ar-script.py</pre>

A non-**PROXY** example:

EGG-INFO/usr/bin/ar.exe	EGG-INFO/mingw/usr/i686-w64-mingw32/bin
-------------------------	---

Egginst will create the following:

<pre># A copy of EGG-INFO/usr/bin/ar.exe \$METADIR\\usr\\i686-w64-mingw32\\bin\\ar.exe</pre>
--

Misc I have seen a few other files in EGG-INFO/inst that seem bogus:

- `install_path.dat` (in the sip-4.8.2-1.egg only), refer to some machine-specific installation path ?
- `app_install.py` and `app_uninstall.py`. Coming from the obsolete `enpisi` (see `buildsystem` commit `eb83c96aa2e1ccca78329faa0d7ddbca6da4a631`). I am not sure whether `enstaller` is doing anything with them anymore

Icons may be found there as well, installed manually from recipes (see e.g. `idle-2.7.3` recipe).

spec subdirectory

depend This file contains all the metadata required to solve dependencies.

It is a python script, and is exec'ed by `egginst/enstaller` to get the actual data (see `egginst.eggmeta.parse_rawspec`).

It is generally written by various functions in `workbench.spec`.

Typical format:

```
metadata_version = '1.1'
name = 'numpy'
version = '1.7.1'
build = 3

arch = 'x86'
platform = 'linux2'
osdist = 'RedHat_5'
python = '2.7'
packages = [
    'MKL 10.3-1',
]
```

Regarding the content:

- `metadata_version` is only used in our old style, obsolete (?) repo in `enstaller.indexed_repo`. It needs to be `>= '1.1'` (indeed as a string, this is not a typo).
- `name`: this is the name of the package. May use upper-case (e.g. for `PIL`, name will be `'PIL'`). This is the name defined in our recipe.
- `version`: the upstream version
- `build`: the build #, as defined in the recipe.
- `arch/platform/osdist`: should be one of the value in the corresponding attributes of `epd_repo.platforms.Platform` instances.

Note: those metadata are guessed from the egg content (See the code in `workbench.spec.update_egg`). I don't know what `osdist` is used for, and it can be `None`.

- `python`: the python version, or `None`. As for `arch/platform/osdist`, this is not set directly, but guessed by looking into the `.pyc` code inside the egg. Unless you define that field explicitly that is (see `greenlet` recipe for an example of this technique).
- `packages`: a list of dependencies, as defined in the `PISI pspec.xml` file. Note that if the platform is not correctly guessed, the dependencies will be silently ignoring the platform label. You will also note that name and version are space separated. The version part is actually optional.

summary A copy of the Summary field in our `pspec.xml`. The code writing this is also in `workbench.spec`.

lib-depend Free-form text format, contains the consolidated output of ldd or otool -L of each library/python extension.

lib-provide Free-form text format, contains the list of provided libraries in that egg. While lib-depend unzip the egg to look for files, lib-provide uses the list of files in files_to_install.txt and do a simple pattern matching to find out what to write.

e

`enstaller.config`, [11](#)
`enstaller.repository`, [14](#)
`enstaller.session`, [16](#)

A

api_url (enstaller.config.Configuration attribute), 13
auth (enstaller.config.Configuration attribute), 13
authenticated_from_configuration() (en-
staller.session.Session class method), 16
autoupdate (enstaller.config.Configuration attribute), 13

C

Configuration (class in enstaller.config), 13

D

delete_package() (enstaller.repository.Repository
method), 14
download() (enstaller.session.Session method), 16

E

enstaller.config (module), 11, 13
enstaller.repository (module), 14
enstaller.session (module), 16

F

filename (enstaller.config.Configuration attribute), 13
find_latest_package() (enstaller.repository.Repository
method), 14
find_package() (enstaller.repository.Repository method),
14
find_package_from_requirement() (en-
staller.repository.Repository method), 15
find_packages() (enstaller.repository.Repository method),
15
find_sorted_packages() (enstaller.repository.Repository
method), 15
from_configuration() (enstaller.session.Session class
method), 16
from_file() (enstaller.config.Configuration class method),
13
from_repository_info() (enstaller.repository.Repository
class method), 15

H

has_package() (enstaller.repository.Repository method),
15

I

indexed_repositories (enstaller.config.Configuration at-
tribute), 13
indices (enstaller.config.Configuration attribute), 13
iter_most_recent_packages() (en-
staller.repository.Repository method), 15
iter_packages() (enstaller.repository.Repository method),
15

M

max_retries (enstaller.config.Configuration attribute), 13

N

noapp (enstaller.config.Configuration attribute), 13

P

prefix (enstaller.config.Configuration attribute), 13
proxy (enstaller.config.Configuration attribute), 13
proxy_dict (enstaller.config.Configuration attribute), 13

R

Repository (class in enstaller.repository), 14
repository_cache (enstaller.config.Configuration at-
tribute), 13

S

Session (class in enstaller.session), 16
set_repositories_from_names() (en-
staller.config.Configuration method), 14
store_kind (enstaller.config.Configuration attribute), 14
store_url (enstaller.config.Configuration attribute), 14

U

update() (enstaller.config.Configuration method), 14
use_pypi (enstaller.config.Configuration attribute), 14

`use_webservice` (enstaller.config.Configuration attribute),
[14](#)

V

`verify_ssl` (enstaller.config.Configuration attribute), [14](#)

W

`webservice_entry_point` (enstaller.config.Configuration
attribute), [14](#)

`write()` (enstaller.config.Configuration method), [14](#)

`write_default_config()` (in module enstaller.config), [14](#)