pryv Documentation

Release 1.0.0a1

Léopold Houdin

Nov 04, 2018

Contents

Quickstart

If you want to quickly have a server running up so to publish, install and explore packages, you found the right place! This page will guide you in the very first steps of installing and configuring a *pryv* server.

pryv is available on https://pypi.org. Use the following command to install it:

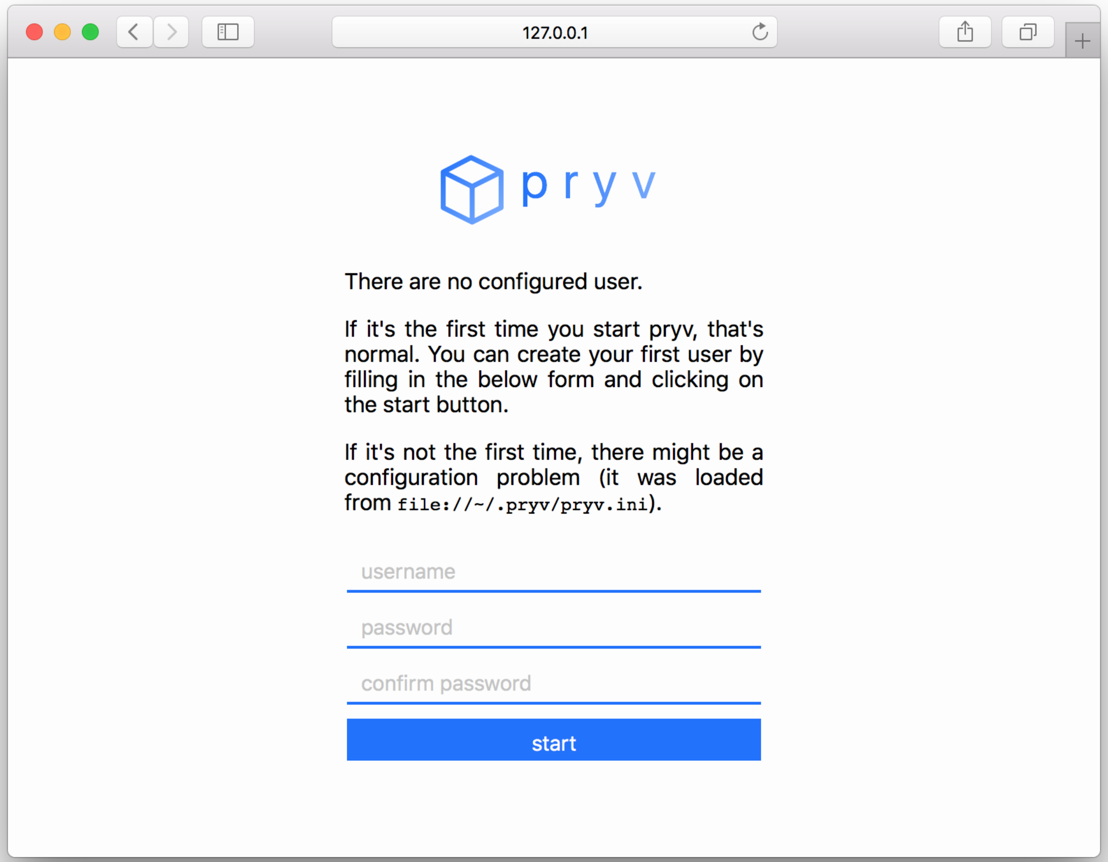
```
pip install pryv
```

You can as well install *pryv* in a *virtualenv*, however it is not recommended for production environment.

Now that *pryv* is installed on your machine, you can start it by using:

```
pryv start
```

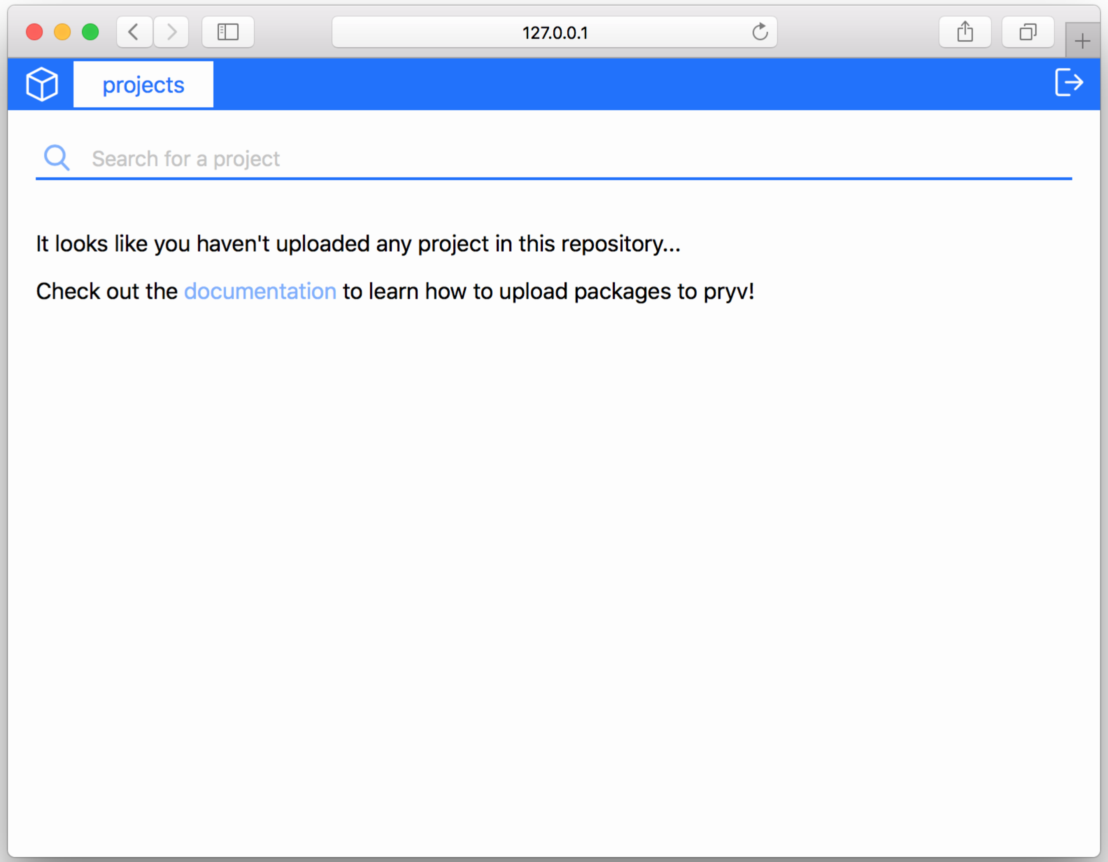
The api and the explorer are then available. Open your browser and navigate to http://127.0.0.1:5555. You should land on pryv's setup interface, that looks like this:


There are no configured user.

If it's the first time you start pryv, that's normal. You can create your first user by filling in the below form and clicking on the start button.

If it's not the first time, there might be a configuration problem (it was loaded from `file://~/.pryv/pryv.ini`).

Once you have filled in the username, password and password confirmation fields, click on the start button. You have just created your first user and *pryv* is now ready for use. You should land on a page that looks like:



It looks like you haven't uploaded any project in this repository...

Check out the documentation to learn how to upload packages to pryv!

As mentioned on the above image, we now recommend you to have a look at the *packages section* in order to read more about how to create projects, upload and install packages to and from your *pryv* server.

You could as well visit the *configuration section* to learn more on how to configure *pryv* to meet your requirements.

CHAPTER 2

Packages

Publishing to your *pryv* server allows you to manage your private projects and packages. It also enables you, and others, to access and install your private libraries and applications while keeping them out of public visibility and use. In this section, you will find details on the core concepts of *pryv* and guides on how to publish and install packages to and from *pryv*.

2.1 Core concepts

The fundamentals behind *pryv* mechanisms are pretty simple. In this section, we provide details on the model used by *pryv* and explain what each entity was designed for and how to manage it.

These were not invented by *pryv*'s **creators**. They are only formalized here, for their use in the context of *pryv*.

2.1.1 Overview

The following figure provides an overview of the different entities and their relation. A detailed description is provided for each of them below.

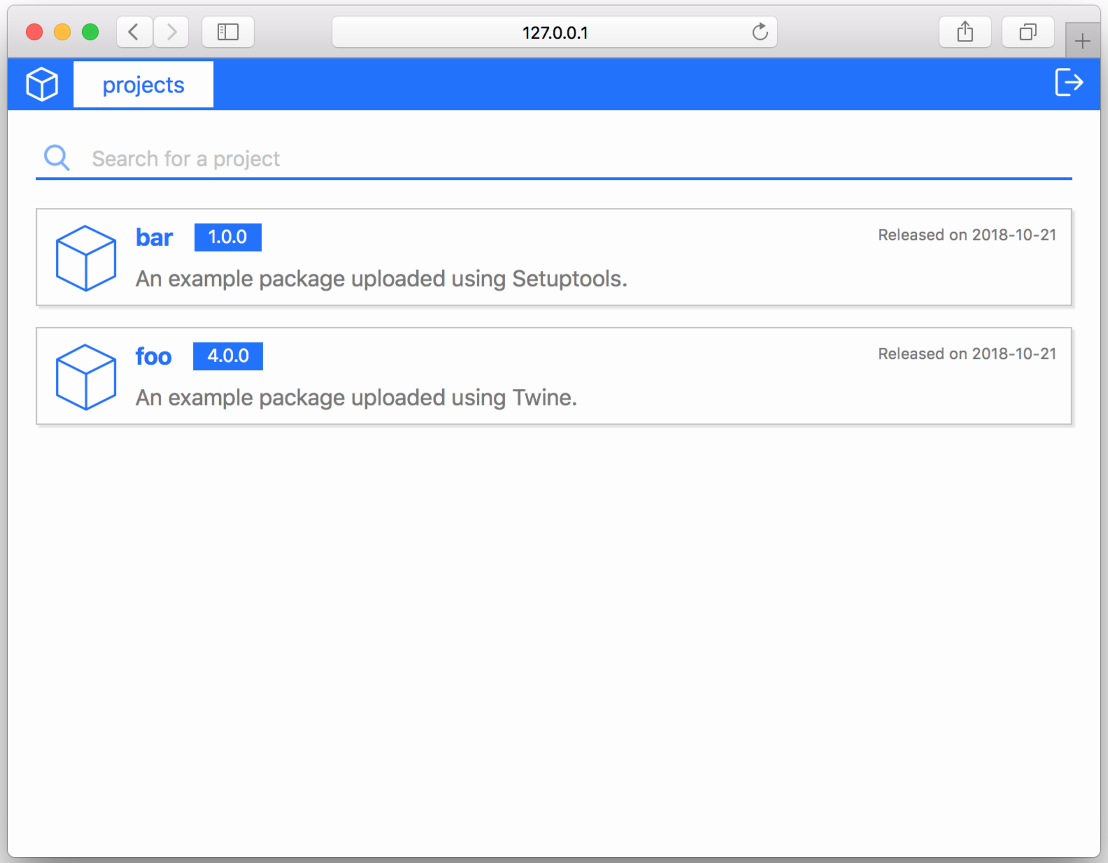
2.1.2 Project

A project is a set of source files that aim at solving a problem. In *pryv*, a project is a collection of releases and is identified by its name.

Different projects in *pryv* **must** have different names. The name used by *pryv* to resolve a project is its normalized Python name (see PEP 503 for details). **Hence several projects with names that evaluate to the same normalized name are considered as the same project by** *pryv* (for example `my-project`, `my.project` and `My_Project` all evaluate to the normalized name `my-project` and thus are considered as a single project named `my-project`).

When *uploading a package* to *pryv*, you **automatically** create a project and its associated release.

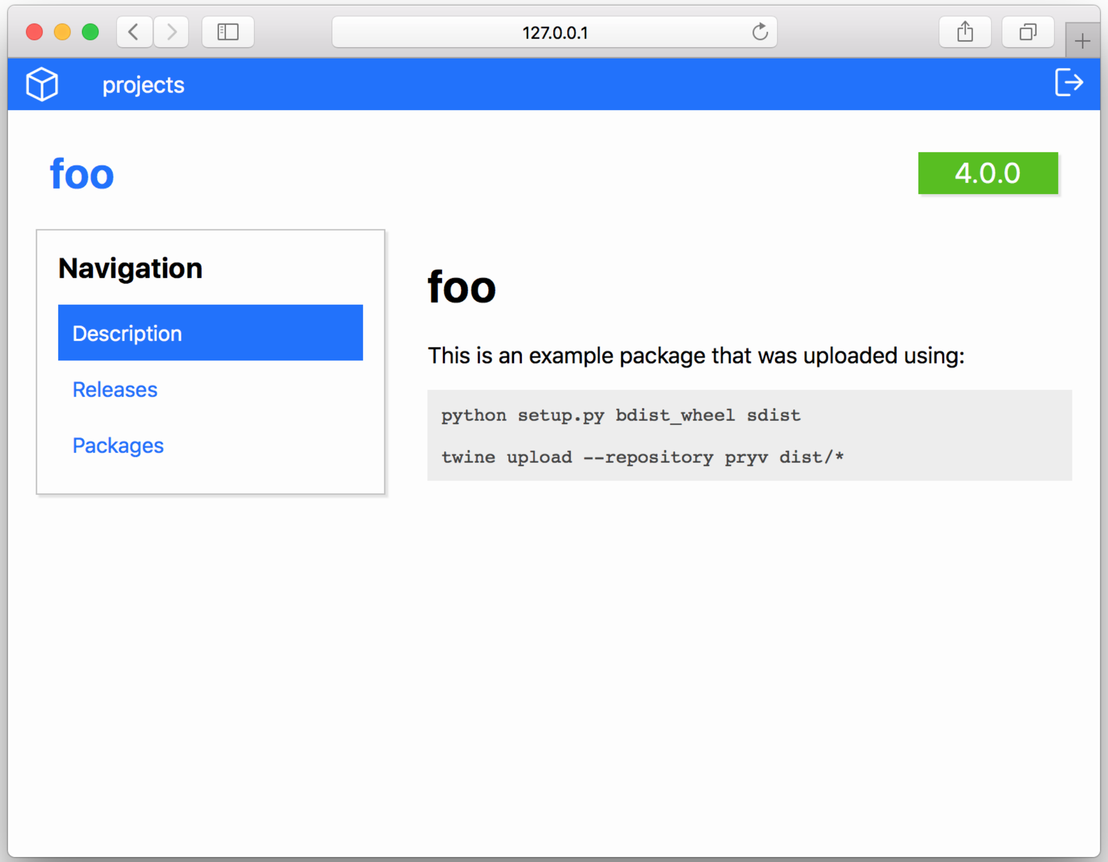
You can see all projects that were created in your *pryv* server by browsing the explorer to the *projects* page. The following image shows an example:



bar 1.0.0 Released on 2018-10-21
An example package uploaded using Setuptools.

foo 4.0.0 Released on 2018-10-21
An example package uploaded using Twine.

By clicking on a project you access the project's details:



foo

This is an example package that was uploaded using:

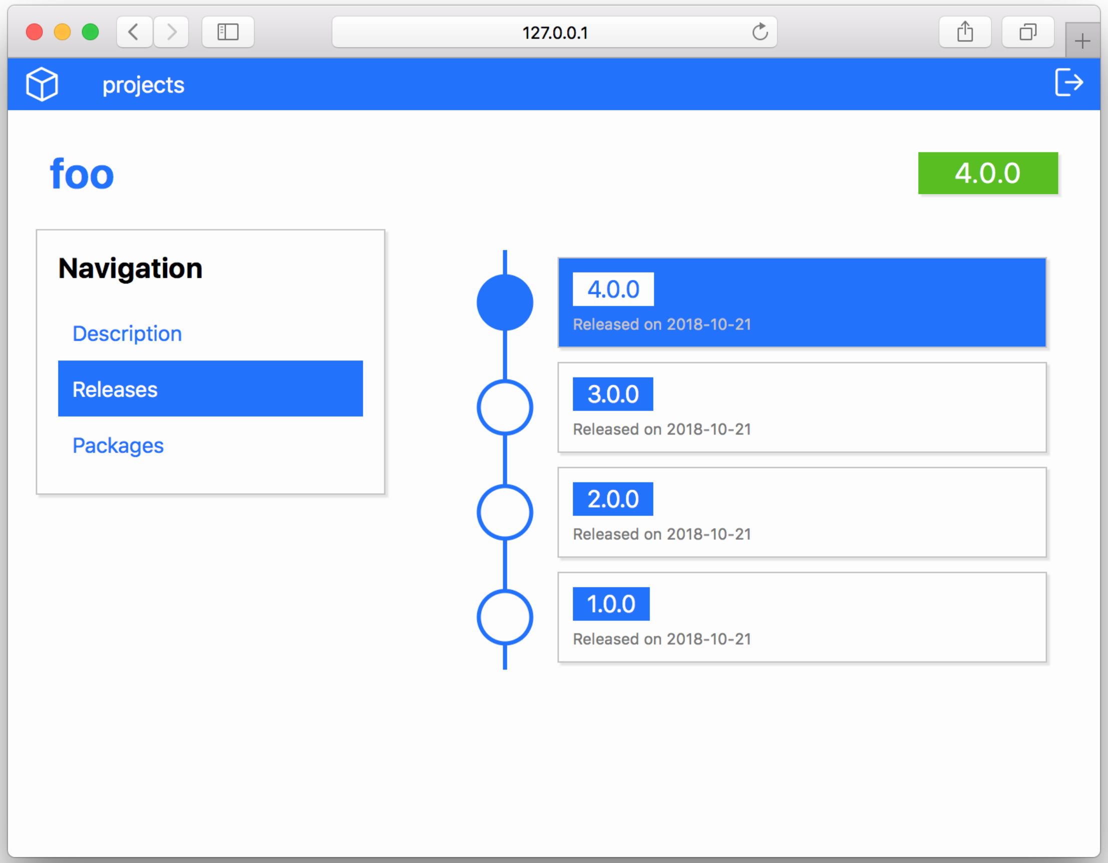
```
python setup.py bdist_wheel sdist
twine upload --repository pryv dist/*
```

2.1.3 Release

A release is a snapshot of a project at a given time. It is uniquely identified within a project by its version number (see PEP 440 for details).

When *uploading a package* to *pryv*, you **automatically** create a release. The first time you upload a package, you will create a release for a project. Subsequent package uploads will create a new release **only for releases with a new version number** (i.e. only if no release with the same version number exist in *pryv*).

You can see all releases of a project by going to the *Releases* tab of the project's page:



2.1.4 Metadata

Metadata provide information on a release (see PEP 566 for details). They are loaded when *uploading a package* to *pryv*. Subsequent uploads of packages to the same release with different metadata overwrite the original metadata.

Metadata are defined in the `setup.py` file of a project (see something for details). Using a tool, such as Twine or Setuptools, for uploading your packages ensures metadata are correctly provided.

The list of metadata fields supported by *pryv* is:

- `metadata_version`: **Only metadata with version** `2.1` **were tested**. Other versions of metadata are not guaranteed to work.
- `name`: This field is used to determine the name of the project.
- `version`: This field is used to determine the version of the release.
- `summary`
- `description`: This field is used to display the project's description.
- `description_content_type`: This field is used to determine the description's content type. Currently only `text/x-rst` is supported. Other content types might be incorrectly rendered.
- `keywords`

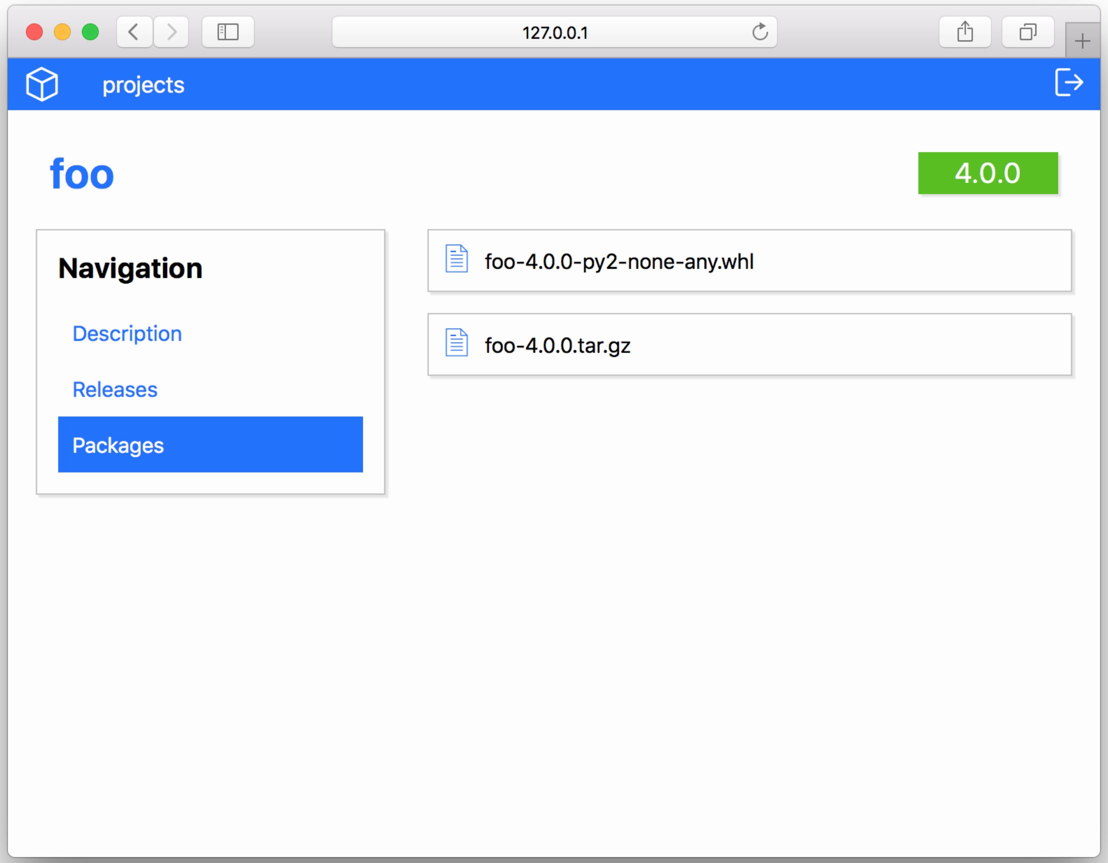
- `home_page`
- `download_url`
- `author`
- `author_email`
- `maintainer`
- `maintainer_email`
- `license`
- `classifiers`
- `requires_dist`
- `requires_python`
- `requires_external`
- `project_url`
- `provides_extra`

2.1.5 Package

A package is a file that contains everything required for others to use a specific release of a project. There might be several packages per project's release. This is up to you.

Read the *uploading* and *using* sections to learn more about how to upload and use packages to and from *pryv*.

You can see all packages that belong to a project's release by going to the *Packages* tab of the project's page:



foo-4.0.0-py2-none-any.whl

foo-4.0.0.tar.gz

2.2 Uploading a package

2.2.1 Prerequisites

pryv expects packaged Python's file to be uploaded. We assume you have basic skills on packaging your code. We only provide a quick overview on how to proceed.

You need to install Setuptools on your machine:

```
pip install --upgrade setuptools
```

You should architecture your project's directory similarly to the following:

```
myproject/
  setup.py
  myproject/
    __init__.py
    other modules...
```

A minimal `setup.py` file looks like the following:

```
from setuptools import setup, find_packages

setup(
    name = 'MyProject',
    version = '1.0.0',
    url = 'https://github.com/me/myproject.git',
    author = 'Me',
    author_email = 'me@gmail.com',
    description = 'Short description of my project',
    packages = find_packages(),
    install_requires = [],
)
```

Now, in order to upload your project to *pryv*, you'll need to package it. You can do the following:

```
python setup.py sdist bdist_wheel
```

Note: To ensure your project's packages to not be published on the public Pypi, you can add the following to your `setup.py`:

```
setup(
    # ...
    classifiers=[
        'Private :: Do Not Upload to pypi server',
    ],
)
```

See also:

- Packages - Python's official documentation
- Packaging Python Projects - Python's official documentation

2.2.2 Uploading a package with Twine

You can upload your packages to *pryv* using Twine.

First, install `setuptools` and `twine`:

```
pip install --upgrade setuptools twine
```

Then you should package your code using `setuptools`. Use the following for both source and wheel distributions:

```
python setup.py sdist bdist_wheel
```

Finally, you upload your freshly built distribution using:

```
twine upload --repository-url http://username:password@127.0.0.1:5555/simple dist/*
```

Alternatively, if you don't want to write the URL to your *pryv* server each time you upload a package, you can use a `.pypirc` configuration file. To do so, create a `.pypirc` file either at the root of your project's directory or in your home directory (i.e. `~/.pypirc`). Set its content to:

```
[distutils]
index-servers =
    pryv

[pryv]
repository = http://127.0.0.1:5555/simple
username =
password =
```

You can now use the following shorthand syntaxes for uploading. If you created the `.pypirc` file at the root of your project's directory:

```
twine upload --config-file .pypirc --repository pryv dist/*
```

Or, if you created a `~/.pypirc` file:

```
twine upload --repository pryv dist/*
```

See also:

- Twine - Project on Pypi
- Twine - Documentation on Read the Docs

2.2.3 Uploading a package with Setuptools

You can also upload your packages to *pryv* by simply using Setuptools:

```
python setup.py sdist bdist_wheel upload --repository pryv
```

Note: In the above example, we assume that a `~/.pypirc` file exists with a `pryv` index server configured, as detailed in the *Twine paragraph*.

Warning: Uploading packages with Setuptools is now deprecated. Use Twine instead.

pryv was tested with Setuptools, but no maintenance is guaranteed on this feature. You should consider using Twine for compatibility.

See also:

- Setuptools - Documentation on Read the Docs
- distutils - Official Python's documentation - The Python Package Index (PyPI)

2.3 Using a package

Using a package uploaded to *pryv* is pretty simple. You will need pip to be installed.

To install a package uploaded to *pryv*, use the following:

```
pip install myproject --index-url http://username:password@127.0.0.1:5555/simple
```

Note: If your *pryv* server is accessed by *HTTPS*, you might consider using:

```
pip install myproject \
--index-url https://username:password@pryv.host.com/simple \
--trusted-host pryv.host.com
```

Alternatively, if you don't want to write the URL to your *pryv* server each time you install a package, you can use a `pip.conf` file located under `~/.pip` (i.e. `~/.pip/pip.conf`). Set its content to:

```
[global]
extra-index-url = https://username:password@pryv.host.com/simple
trusted-host = pryv.host.com
```

Now, you are able to install packages as usual, directly from your *pryv* repository:

```
pip install myproject
```

CHAPTER 3

Users

Users management in *pryv* is done by editing files. You can use either:

- `users.json` (which path can be set using the *user path* option)
- `.htpasswd` (which path can be set using the *htpasswd path* option).

Currently, you can only use either, not both. If the `htpasswd-path` configuration option is set, the `user-path` option will be ignored.

3.1 `users.json`

This is a flat JSON file, with one field per user, its username. Each user entry has one field, the user's password hash (SHA-256).

Its content looks like:

```
{
  "user1": {
    "password": "superlongsha256"
  },
  "user2": {
    "password": "anothersuperlongsha256"
  }
}
```

To add a new user, simply add a new entry to this file.

3.2 `.htpasswd`

This file uses Apache's `.htpasswd` file format (see apache's documentation for details).

Use this file if you want to share your users credentials among several applications using `.htpasswd` files on your machine.

CHAPTER 4

Configuration

pryv can be configured using a configuration file. By default, the configuration is read from `~/.pryv/pryv.ini`, however you can set the path to the configuration path be either using the `PRYV_CONFIG_PATH` environment variable or by setting the `-c`/`--config` option at start-up, i.e.:

```
pryv start -c path/to/pryv.ini
```

The configuration file has the following format:

```
# Comments
[section]
option1 = value1
option2 = value2
```

pryv uses three different sections:

4.1 Core

This section configures the core of *pryv*: where projects data and packages are stored.

4.1.1 Projects path

This option defines the path to the **directory** where projects data are stored.

Usage

```
[core]
projects-path = file:///path/to/projects
```

Default

```
file://~/.pryv/projects
```

4.1.2 Packages path

This option defines the path to the **directory** where packages are stored.

Usage

```
[core]
packages-path = file:///path/to/packages
```

Default

```
file://~/.pryv/packages
```

4.2 User

This section configures the path to the files were users data are loaded from. These files are used to load authentication data.

4.2.1 Users path

This option defines the path to the **file** where users data are stored.

Note: If the `htpasswd-path` option is set, this option is not used.

Warning: The users file contains sensitive data (username and password hash). Make sure its location is safe.

Usage

```
[user]
users-path = file:///path/to/users.json
```

Default

```
file://~/.pryv/users.json
```

4.2.2 `.htpasswd` path

This option defines the path to the `.htpasswd` where users credentials are stored. Use this option if you want to share the credentials among different servers on the same machine. The file at the configured path is expected to be `.htpasswd` file that meets the required format (see apache's documentation for details).

Usage

```
[user]
htpasswd-path = file:///path/to/.htpasswd
```

Default: None

4.3 Server

This section configures the server's options.

4.3.1 Host

This option defines the host name the server listens to. By default, it is set such that the server only listens for requests from the same machine (i.e. `127.0.0.1`). Set its value to `0.0.0.0` to listen to everyone.

Usage

```
[server]
host = 123.234.123.234
```

Default

```
127.0.0.1
```

4.3.2 Port

This options defines the port the server listens to.

Usage

```
[server]
port = 1234
```

Default

```
5555
```

4.3.3 Secret key

This option defines the secret key used by the explorer to sign cookies.

Usage

```
[server]
secret-key = super-secret-key
```

Default: By default, the server generates a new random secret key each time you start it. It uses the host operating system's `urandom` function.

Note: Generating a new random secret key at each start up means all old sessions are obsolete. The users will have to login again.

4.3.4 Debug

This option defines whether the server should show logs or not. If set, logs will be shown on request.

Usage

```
[server]
debug = true
```

Default: None

CHAPTER 5

Reference

Warning: This section is not meant for *pryv*'s users, but targets developers.

This section provides with the reference for *pryv*'s **internals**.

5.1 `pryv.core`

5.1.1 `pryv.core.project`

5.1.2 `pryv.core.release`

5.1.3 `pryv.core.metadata`

5.1.4 `pryv.core.package`

5.2 `pryv.user`

5.3 `pryv.api`

5.3.1 `pryv.api.simple`

5.3.2 `pryv.api.package`

5.4 `pryv.explorer`

5.4.1 `pryv.explorer.auth`

5.4.2 `pryv.explorer.project`

5.5 `pryv.proxy`

This module defines proxy access functions to file storage. It enables manipulating files in the local file system and AWS S3 in a unified way.

`pryv.proxy.`**`ensure_directory`**`(`**paths*`)`

Gets a file system proxy to the given root path and ensures it is a directory.

Parameters **`paths`** (`str`) – The components of the root path.

Returns A file system proxy.

Return type *`FileSystemProxy`*

`pryv.proxy.`**`get`**`(`**paths*`)`

Gets a proxy to the root path.

Parameters **`paths`** (`str`) – The components of the root path.

Returns A file system proxy.

Return type *`FileSystemProxy`*

5.5.1 `pryv.proxy.base`

This module defines the *`FileSystemProxy`* and *`FileProxy`* classes. These are the base classes for the file proxies.

class `pryv.proxy.base.`**`FileProxy`**

Bases: `object`

This class defines a file proxy.

A file proxy enables to manipulate files located on the local system or on an AWS S3 bucket in the same manner.

`__init__``()`

Creates a new file proxy.

`flush``()`

Flushes the file proxy.

> **Warning:** To override.

`read``(`*size=0*`)`

Reads data from the file proxy.

> **Warning:** To override.

Parameters **`size`** (`int`) – The number of bytes to read.

Returns Data read.

`readline``()`

Reads a line of data from the file proxy.

> **Warning:** To override.

Returns Data read.

`write``(`*data*`)`

Writes data to the file proxy.

> **Warning:** To override.

Parameters **`data`** – The data to write.

class `pryv.proxy.base.`**`FileSystemProxy`**

Bases: `object`

This class defines a file system proxy.

A file system proxy enables interacting files located on different storage in an unified manner.

`__init__``()`

Creates a new abstract file system proxy.

`exists``(`**paths*, ***kwargs*`)`

Gets a value indicating whether the file at the given path exists or not.

Note: The path can be given either as a single parameter or by components.

Parameters **`path`** – The path to the file.

Type path: `str`

Returns `True` if the path targets a file that exist; `False` otherwise.

Return type `bool`

`is_directory`(**paths*, ***kwargs*)

Gets a value indicating whether the file at the given path is a directory or not.

Note: The path can be given either as a single parameter or by components.

Parameters **`path`** – The path to the file.

Type path: `str`

Returns `True` if the path targets a directory; `False` otherwise.

Return type `bool`

`is_file`(**paths*, ***kwargs*)

Gets a value indicating whether the file at the given path is a file or not.

Note: The path can be given either as a single parameter or by components.

Parameters **`path`** – The path to the file.

Type path: `str`

Returns `True` if the path targets a file; `False` otherwise.

Return type `bool`

`list_directories`(**paths*, ***kwargs*)

Lists the directories located at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters

- **`path`** – The path.
- **`exclude`** (`list`) – A list of directory names to exclude.

Type path: `str`

Returns A list of directory names.

Return type `list`

list_files(**paths*, ***kwargs*)

Lists the files located at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters

- **path** – The path to the file.
- **exclude** (`list`) – A list of file names to exclude.

Type path: `str`

Returns A list of file names.

Return type `list`

make_directory(**paths*, ***kwargs*)

Creates a directory at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters **path** – The path.

Type path: `str`

open_file(**paths*, ***kwargs*)

Opens the file located at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters

- **path** – The path to the file.
- **mode** (`str`) – The opening mode (`r` or `w`).

Type path: `str`

Returns A file instance.

Return type *`FileProxy`*

remove_file(**paths*, ***kwargs*)

Removes the file located at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters **path** – The path to the file.

Type path: `str`

`pryv.proxy.base.`**join_path**(**paths*)

Joins the given path components into a single path.

Parameters **`paths`** (`str`) – The path components.

Returns A path.

Return type `str`

`pryv.proxy.base.`**`with_path`**(*func*)

Decorates the given function as requiring a path. It converts the given path components into a single path.

Parameters **`func`** (`function`) – The function to decorate.

Returns A decorated function.

Return type `function`

5.5.2 `pryv.proxy.local`

This module defines the *`LocalFileSystemProxy`* and *`LocalFileProxy`* classes that enable manipulating files on the local file system.

`class` `pryv.proxy.local.`**`LocalFileProxy`**(*path*, *mode*)

Bases: *`pryv.proxy.base.FileProxy`*

This class defines a proxy to a file located on the local file system.

`__init__`(*path*, *mode*)

Creates a new instance of a local file proxy.

Parameters

- **`path`** (`str`) – The path to the file.
- **`mode`** (`str`) – The opening mode of the file.

`flush`()

Flushes the file proxy.

Warning: To override.

`read`(*size=None*)

Reads data from the file proxy.

Warning: To override.

Parameters **`size`** (`int`) – The number of bytes to read.

Returns Data read.

`readline`()

Reads a line of data from the file proxy.

Warning: To override.

Returns Data read.

write(*data*)
Writes data to the file proxy.

Warning: To override.

Parameters **data** – The data to write.

class pryv.proxy.local.**LocalFileSystemProxy**(*path*)
Bases: *pryv.proxy.base.FileSystemProxy*

This class defines a proxy to the local file system.

__init__(*path*)
Creates a new instance of a proxy to the local file system. :param path:

exists(**paths*, ***kwargs*)
Gets a value indicating whether the file at the given path exists or not.

Note: The path can be given either as a single parameter or by components.

Parameters **path** – The path to the file.

Type path: `str`

Returns `True` if the path targets a file that exist; `False` otherwise.

Return type `bool`

is_directory(**paths*, ***kwargs*)
Gets a value indicating whether the file at the given path is a directory or not.

Note: The path can be given either as a single parameter or by components.

Parameters **path** – The path to the file.

Type path: `str`

Returns `True` if the path targets a directory; `False` otherwise.

Return type `bool`

is_file(**paths*, ***kwargs*)
Gets a value indicating whether the file at the given path is a file or not.

Note: The path can be given either as a single parameter or by components.

Parameters **path** – The path to the file.

Type path: `str`

Returns `True` if the path targets a file; `False` otherwise.

Return type `bool`

`list_directories`(**paths*, ***kwargs*)
Lists the directories located at the given path.

> **Note:** The path can be given either as a single parameter or by components.

Parameters

- **`path`** – The path.
- **`exclude`** (`list`) – A list of directory names to exclude.

Type path: `str`

Returns A list of directory names.

Return type `list`

`list_files`(**paths*, ***kwargs*)
Lists the files located at the given path.

> **Note:** The path can be given either as a single parameter or by components.

Parameters

- **`path`** – The path to the file.
- **`exclude`** (`list`) – A list of file names to exclude.

Type path: `str`

Returns A list of file names.

Return type `list`

`make_directory`(**paths*, ***kwargs*)
Creates a directory at the given path.

> **Note:** The path can be given either as a single parameter or by components.

Parameters **`path`** – The path.

Type path: `str`

`open_file`(**paths*, ***kwargs*)
Opens the file located at the given path.

> **Note:** The path can be given either as a single parameter or by components.

Parameters

- **`path`** – The path to the file.
- **`mode`** (`str`) – The opening mode (`r` or `w`).

Type path: `str`

Returns A file instance.

Return type *FileProxy*

remove_file(**paths*, ***kwargs*)
Removes the file located at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters **path** – The path to the file.

Type path: `str`

`pryv.proxy.local.`**with_full_path**(*func*)
Decorates the given function as requiring a full path. This decorator merges the root path of decorated file system instance and the given path (either as a single parameter or as a list of path components).

Parameters **func** (`function`) – The function to decorate.

Returns A decorated function.

Return type `function`

5.5.3 pryv.proxy.s3

This module defines the *S3FileSystemProxy* and `S3FileProxy` classes that enable manipulating files located on an AWS S3 bucket.

class `pryv.proxy.s3.`**S3FileSystemProxy**(*bucket*, *prefix*)
Bases: *pryv.proxy.base.FileSystemProxy*

This class defines a proxy to a file system located on an AWS S3 bucket.

__init__(*bucket*, *prefix*)
Creates a new instance of a proxy to an AWS S3 bucket.

Parameters

- **bucket** (`str`) – The name of the bucket.
- **prefix** (`str`) – The key prefix to the objects.

exists(**paths*, ***kwargs*)
Gets a value indicating whether the file at the given path exists or not.

Note: The path can be given either as a single parameter or by components.

Parameters **path** – The path to the file.

Type path: `str`

Returns `True` if the path targets a file that exist; `False` otherwise.

Return type `bool`

is_directory(**paths*, ***kwargs*)

Gets a value indicating whether the file at the given path is a directory or not.

Note: The path can be given either as a single parameter or by components.

Parameters **path** – The path to the file.

Type path: `str`

Returns `True` if the path targets a directory; `False` otherwise.

Return type `bool`

is_file(**paths*, ***kwargs*)

Gets a value indicating whether the file at the given path is a file or not.

Note: The path can be given either as a single parameter or by components.

Parameters **path** – The path to the file.

Type path: `str`

Returns `True` if the path targets a file; `False` otherwise.

Return type `bool`

list_directories(**paths*, ***kwargs*)

Lists the directories located at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters

- **path** – The path.
- **exclude** (`list`) – A list of directory names to exclude.

Type path: `str`

Returns A list of directory names.

Return type `list`

list_files(**paths*, ***kwargs*)

Lists the files located at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters

- **path** – The path to the file.
- **exclude** (`list`) – A list of file names to exclude.

Type path: `str`

Returns A list of file names.

Return type `list`

`make_directory`(**paths*, ***kwargs*)
Creates a directory at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters **`path`** – The path.

Type path: `str`

`open_file`(**paths*, ***kwargs*)
Opens the file located at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters

- **`path`** – The path to the file.
- **`mode`** (`str`) – The opening mode (`r` or `w`).

Type path: `str`

Returns A file instance.

Return type *`FileProxy`*

`remove_file`(**paths*, ***kwargs*)
Removes the file located at the given path.

Note: The path can be given either as a single parameter or by components.

Parameters **`path`** – The path to the file.

Type path: `str`

5.6 `pryv.config`

This module defines the configuration interface. It reads the file `file://~/.pryv/pryv.ini` and exposes its values.

The *`Config`* and *`ConfigSection`* wrap the class `ConfigParser.ConfigParser` in order to enable one to access configuration section and options directly from the code, i.e. using attribute notation. In addition, it handles automatically default values.

class `pryv.config.`**`Config`**(*path='file://~/.pryv/pryv.ini'*)
Bases: `object`

This class defines a configuration object. It is a wrapper around the standard `ConfigParser.ConfigParser` class.

It is designed as a convenience class to access configuration sections and options from code and return default values dynamically.

Example:

```
config = pryv.config.Config()

config.my_section.my_option # -> my-value
```

__init__(*path='file://~/.pryv/pryv.ini'*)
Creates a new instance of a configuration object. This method loads the configuration from the file `file://~/.pryv/pryv.ini`.

Parameters **path** (`str`) – The path to the configuration file.

class `pryv.config.`**ConfigSection**(*name, options*)
Bases: `object`

This class defines a configuration section.

It is designed as a convenience class to access configuration options from code and set default values to options that are not present in the configuration file.

Example:

```
section = pryv.config.ConfigSection('my-section', {'my-option': 'my-value'})

section.my_option # -> my-value

# Assuming my-other-option has for default value my-other-value
section.my_other_option # -> my-other-value
```

__init__(*name, options*)
Creates a new instance of a configuration section.

Parameters

- **name** (`str`) – The name of the section.
- **options** (`dict`) – The options of the section.

set(*option_name, option_value*)
Sets the value of the option with the given name.

Parameters

- **option_name** (`` `str ``) – The name of the option.
- **option_value** – The value of the option.

`pryv.config.`**init**(*path*)
Initializes the configuration to read options from the file with the given path.

Note: The path can be either a path to a file on the local file system (i.e. not prefixed or using the `file://` prefix) or a file located on an AWS S3 bucket (i.e. using the `s3://` prefix).

Parameters **path** (`str`) – The path to the configuration file.

`pryv.config.`**pythonize_name**(*name*)
Pythonizes an option name.

Example:

```
pythonize_name('my-option') # -> my_option
```

Parameters **name** (str) – The name to pythonize.

Returns A pythonized name.

Return type str

`pryv.config.`**`unpythonize_name`**`(`*name*`)`

Unythonizes an attribute name.

Example:

```
unpythonize_name('my_option') # -> my-option
```

Parameters **name** (str) – The name to unpythonize.

Returns An option name.

Return type str

5.7 pryv.app

5.8 pryv.cli

This module defines pryv's Command Line Interface. It defines the interface's groups and commands.

`pryv.cli.`**`start_options`**`(`*cmd*`)`

Applies the start options to the given command.

Parameters **cmd** (function) – The command.

Returns A command.

Return type function

CHAPTER 6

Contributing

pryv welcomes all contributions! Whether you want to report a bug, request a new feature, contribute code or documentation, you found the right place.

6.1 Reporting a bug or submitting a feature request

If you want to either report a bug or request a feature, you should create a new issue on Github.

For bug reports, include the version of *pryv* you observer the bug to happen. You can get *pryv*'s version by running:

```
pryv version
```

6.2 Contributing code

If you want to contribute with code, the following guide is made for you. You might also want to have a look at the *reference*.

6.2.1 Version control

pryv is versioned using Git and its code is available on Github on its main repository. To contribute with code to *pryv* you will need to create a Github account.

6.2.2 Getting the code

The easiest and preferred way to contribute to *pryv* is to fork its main repository on Github. To do so, browse the project's repository and click on the **Fork** button.

Now, clone the forked project to your local machine:

```
git clone git@github.com:your-username/pryv.git
```

6.2.3 Isolating your environment

In order not to *pollute* your local setup and to ensure you are using the same environment as the one required by *pryv*, you should use a virtual environment. virtualenv is the preferred method.

Install it using pip:

```
pip install --upgrade virtualenv
```

Go to the root directory of *pryv* and create a `virtualenv` (named `venv` for example) using:

```
virtualenv venv
```

Enter (activate) the `virtualenv` (**note that you need to do the following command each time you want to enter the** `virtualenv`):

```
source venv/bin/activate
```

You can now develop and issue all required installations without *polluting* your machine's Python environment.

When you are finished developing, you can exit the `virtualenv` using:

```
deactivate
```

6.2.4 Creating a branch

Never work directly on `master`, **always** branch before making any changes. Grab *ID* of the issue you target.

If the issue is a feature request, use the following to create a new branch:

```
git checkout -b feat/123-short-description
```

If the issue is a bug, use:

```
git checkout -b fix/123-short-description
```

`123` is to be replaced with the *ID* of the issue and `short-description` is a short description of the issue. Its value is up to you, just try to be as specific and concise as the same time (i.e. max 3 words). Use hyphences (`-`) in place of spaces.

6.2.5 Commiting code

Try to commit your code regularly. Commit files and changes related to each other.

pryv uses a similar commit naming convention than the Angular's one:

```
<type>(<scope>): <subject>

- details description
- details description

<footer>
```

where `<type>` can be:

- `feat`: for new feature
- `fix`: for bug fix
- `refactor`: for code refactoring
- `style`: for code formatting
- `test`: when adding tests or updating existing ones
- `chore`: for maintenance

`<scope>` is the scope of the change:

- for meta scopes:
  - `readme`
  - `doc`
  - `setup`
- for modules and packages:
  - `core`
  - `api`
  - `config`
  - `cli`
  - `...`

and `<subject>` is a small description of the change starting with an imperative (`change`, `add`, `fix`, **not** `changed` nor `fixing`), in lower case, with no punctuation (i.e. no final `.`).

The body of the message (optional, but recommended for bigger commits) **must** be a list of detailed changes, using imperative tenses as well, in lower case, without punctuation.

The footer (optional) can be:

```
Closes #123
```

if the commit closes an issue. For a commit that closes *several* issues, use:

```
Closes #123, #456, #789
```

6.2.6 Pushing changes

Once you have finished modifying the code (don't forget to update the documentation), you can push your local changes to Github using:

```
git push origin feat/my-branch
```

6.2.7 Opening a pull request

To merge your changes into *pryv*'s main repository, you will have to open a pull request.

To do so, go to your repository on Github, click on the **Pull request** button. Make sure all commits and all changes are there, then fill in the pull request's description and click on the **Send pull request** button.

Your changes are now sent to *pryv*'s maintainers for code review. They might accept your changes or request modifications before merging to *pryv*'s code base.

6.3 Contributing documentation

pryv uses reStructuredText and Sphinx for its documentation. It is located in the `docs/` directory of the project (and in the source file for packages, modules, classes and functions documentation).

Since contributing documentation is in many manner, and in particular since documentation is code, you should have a look at the *contributing code* paragraph of this page.

Building the documentation is all automated thanks to tox. Install it using:

```
pip install --upgrade tox
```

Then, build the documentation using:

```
tox -e docs
```

CHAPTER 7

Changelog

7.1 1.X.X

7.1.1 1.0.X

1.0.0

- Initial release

Python Module Index

Index