

---

# **Brewtils Documentation**

***Release 2.4.2***

**Logan Asher Jones**

**Oct 08, 2018**



---

## Contents

---

<b>1</b>	<b>Brewtils</b>	<b>3</b>
1.1	Features . . . . .	3
1.2	Installation . . . . .	3
1.3	Quick Start . . . . .	3
1.4	Documentation . . . . .	5
<b>2</b>	<b>Installation</b>	<b>7</b>
2.1	Stable release . . . . .	7
2.2	From sources . . . . .	7
<b>3</b>	<b>Usage</b>	<b>9</b>
<b>4</b>	<b>Brewtils</b>	<b>11</b>
4.1	Features . . . . .	11
4.2	Installation . . . . .	11
4.3	Quick Start . . . . .	11
4.4	Documentation . . . . .	13
<b>5</b>	<b>Contributing</b>	<b>15</b>
5.1	Types of Contributions . . . . .	15
5.2	Get Started! . . . . .	16
5.3	Pull Request Guidelines . . . . .	17
5.4	Tips . . . . .	17
<b>6</b>	<b>Credits</b>	<b>19</b>
6.1	Development Leads . . . . .	19
6.2	Contributors . . . . .	19
<b>7</b>	<b>Brewtils Changelog</b>	<b>21</b>
7.1	2.4.2 . . . . .	21
7.2	2.4.1 . . . . .	21
7.3	2.4.0 . . . . .	21
7.4	2.3.7 . . . . .	22
7.5	2.3.6 . . . . .	22
7.6	2.3.5 . . . . .	22
7.7	2.3.4 . . . . .	23
7.8	2.3.3 . . . . .	23

7.9	2.3.2	23
7.10	2.3.1	24
7.11	2.3.0	24
7.12	2.2.1	24
7.13	2.2.0	24
7.14	2.1.1	25

Contents:



Brewtils is the Python library for interfacing with Beergarden systems. If you are planning on writing beer-garden plugins, this is the correct library for you. In addition to writing plugins, it provides simple ways to query the API and is officially supported by the beer-garden team.

## 1.1 Features

Brewtils helps you interact with beer-garden.

- Easy way to create beer-garden plugins
- Full support of the entire Beer-Garden API
- Officially supported by the beer-garden team

## 1.2 Installation

To install brewtils, run this command in your terminal:

```
$ pip install brewtils
```

Or add it to your `requirements.txt`

```
$ cat brewtils >> requirements.txt  
$ pip install -r requirements.txt
```

## 1.3 Quick Start

You can create your own beer-garden plugins without much problem at all. To start, we'll create the obligatory hello-world plugin. Creating a plugin is as simple as:

```
from brewtils.decorators import system, parameter, command
from brewtils.plugin import RemotePlugin

@system
class HelloWorld(object):

    @parameter(key="message", description="The message to echo", type="String")
    def say_hello(self, message="World!"):
        print("Hello, %s!" % message)
        return "Hello, %s!" % message

if __name__ == "__main__":
    client = HelloWorld()
    plugin = RemotePlugin(client,
                          name="hello",
                          version="0.0.1",
                          bg_host='127.0.0.1',
                          bg_port=2337)

    plugin.run()
```

Assuming you have a Beer Garden running on port 2337 on localhost, running this will register and start your plugin! You now have your first plugin running in beer-garden. Let's use another part of the `brewtils` library to exercise your plugin from python.

The `SystemClient` is designed to help you interact with registered Systems as if they were native Python objects.

```
from brewtils.rest.system_client import SystemClient

hello_client = SystemClient('localhost', 2337, 'hello')

request = hello_client.say_hello(message="from system client")

print(request.status) # 'SUCCESS'
print(request.output) # Hello, from system client!
```

In the background, the `SystemClient` has executed an HTTP POST with the payload required to get beer-garden to execute your command. The `SystemClient` is how most people interact with beer-garden when they are in the context of python and want to be making requests.

Of course, the rest of the API is accessible through the `brewtils` package. The `EasyClient` provides simple convenient methods to call the API and auto-serialize the responses. Suppose you want to get a list of all the commands on all systems:

```
from brewtils.rest.easy_client import EasyClient

client = EasyClient('localhost', 2337)

systems = client.find_systems()

for system in systems:
    for command in system.commands:
        print(command.name)
```

This is just a small taste of what is possible with the `EasyClient`. Feel free to explore all the methods that are exposed.

For more detailed information and better walkthroughs, checkout the full documentation!



## 1.4 Documentation

- Full Beer Garden documentation is available at <https://beer-garden.io>
- Brewtils Documentation is available at <https://brewtils.readthedocs.io>



### 2.1 Stable release

To install Brewtils, run this command in your terminal:

```
$ pip install brewtils
```

This is the preferred method to install Brewtils, as it will always install the most recent stable release.

If you don't have [pip](#) installed, this [Python installation guide](#) can guide you through the process.

### 2.2 From sources

The sources for Brewtils can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git@github.com:beer-garden/brewtils.git
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/beer-garden/brewtils/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```



## CHAPTER 3

---

Usage

---



Brewtils is the Python library for interfacing with Beergarden systems. If you are planning on writing beer-garden plugins, this is the correct library for you. In addition to writing plugins, it provides simple ways to query the API and is officially supported by the beer-garden team.

### 4.1 Features

Brewtils helps you interact with beer-garden.

- Easy way to create beer-garden plugins
- Full support of the entire Beer-Garden API
- Officially supported by the beer-garden team

### 4.2 Installation

To install brewtils, run this command in your terminal:

```
$ pip install brewtils
```

Or add it to your `requirements.txt`

```
$ cat brewtils >> requirements.txt  
$ pip install -r requirements.txt
```

### 4.3 Quick Start

You can create your own beer-garden plugins without much problem at all. To start, we'll create the obligatory hello-world plugin. Creating a plugin is as simple as:

```
from brewtils.decorators import system, parameter, command
from brewtils.plugin import RemotePlugin

@system
class HelloWorld(object):

    @parameter(key="message", description="The message to echo", type="String")
    def say_hello(self, message="World!"):
        print("Hello, %s!" % message)
        return "Hello, %s!" % message

if __name__ == "__main__":
    client = HelloWorld()
    plugin = RemotePlugin(client,
                          name="hello",
                          version="0.0.1",
                          bg_host='127.0.0.1',
                          bg_port=2337)

    plugin.run()
```

Assuming you have a Beer Garden running on port 2337 on localhost, running this will register and start your plugin! You now have your first plugin running in beer-garden. Let's use another part of the `brewtils` library to exercise your plugin from python.

The `SystemClient` is designed to help you interact with registered Systems as if they were native Python objects.

```
from brewtils.rest.system_client import SystemClient

hello_client = SystemClient('localhost', 2337, 'hello')

request = hello_client.say_hello(message="from system client")

print(request.status) # 'SUCCESS'
print(request.output) # Hello, from system client!
```

In the background, the `SystemClient` has executed an HTTP POST with the payload required to get beer-garden to execute your command. The `SystemClient` is how most people interact with beer-garden when they are in the context of python and want to be making requests.

Of course, the rest of the API is accessible through the `brewtils` package. The `EasyClient` provides simple convenient methods to call the API and auto-serialize the responses. Suppose you want to get a list of all the commands on all systems:

```
from brewtils.rest.easy_client import EasyClient

client = EasyClient('localhost', 2337)

systems = client.find_systems()

for system in systems:
    for command in system.commands:
        print(command.name)
```

This is just a small taste of what is possible with the `EasyClient`. Feel free to explore all the methods that are exposed.

For more detailed information and better walkthroughs, checkout the full documentation!



## 4.4 Documentation

- Full Beer Garden documentation is available at <https://beer-garden.io>
- Brewtils Documentation is available at <https://brewtils.readthedocs.io>



Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at <https://github.com/beer-garden/brewtils/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

### 5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

### 5.1.4 Write Documentation

Brewtils could always use more documentation, whether as part of the official Brewtils docs, in docstrings, or even on the web in blog posts, articles, and such.

### 5.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/beer-garden/brewtils/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 5.2 Get Started!

Ready to contribute? Here's how to set up `brewtils` for local development.

1. Fork the `brewtils` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/brewtils.git
```

3. Install your local copy into a virtualenv. Assuming you have `virtualenvwrapper` installed, this is how you set up your fork for local development:

```
$ mkvirtualenv brewtils
$ cd brewtils/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass `flake8` and the tests, including testing other Python versions with `tox`:

```
$ flake8 brewtils test
$ nosetests
$ tox
```

To get `flake8` and `tox`, just `pip` install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.5, and 3.6. Check [https://travis-ci.org/beer-garden/brewtils/pull\\_requests](https://travis-ci.org/beer-garden/brewtils/pull_requests) and make sure that the tests pass for all supported Python versions.

## 5.4 Tips

To run a subset of tests:

```
$ nosetests test/models_test.py:SystemTest.test_instance_names
```



### 6.1 Development Leads

- Logan Asher Jones <loganasherjones@gmail.com>
- Matt Patrick

### 6.2 Contributors

None yet. Why not be the first?





### 7.1 2.4.2

Date: 10/7/18

#### 7.1.1 New Features

- Ability to specify a timeout for Beergarden communication (beer-garden/#87)
- `parameters` decorator for cleaner command definitions (beer-garden/#82)

#### 7.1.2 Bug Fixes

- Fixed error when republishing a message to RabbitMQ (beer-garden/#88)

### 7.2 2.4.1

Date: 09/11/18

#### 7.2.1 Other Changes

- Changed Plugin warning type so it won't be displayed by default

### 7.3 2.4.0

Date: 09/5/18

### 7.3.1 New Features

- Added job scheduling capability (beer-garden/#10)
- Added support for authentication / users (beer-garden/#35)
- Plugins will load log level from the environment (bartender/#4)
- RestClient now exposes `base_url` (#58)
- SystemClient can wait for a request to complete instead of polling (#54)
- Allowing custom argument parser when loading configuration (#67)
- Support for TLS connections to RabbitMQ (#74)
- Warning for future change to plugin `max_concurrent` default value (#79)
- Added methods `get_config` to RestClient, `can_connect` to EasyClient

### 7.3.2 Other Changes

- Renamed PluginBase to Plugin (old name is aliased)

## 7.4 2.3.7

Date: 07/11/18

### 7.4.1 New Features

- Current request can be accessed using `self._current_request` (beer-garden/#78)

### 7.4.2 Bug Fixes

- Updating import problem from lark-parser #61
- Pinning setup.py versions to prevent future breaks

## 7.5 2.3.6

Date: 06/06/18

### 7.5.1 Other Changes

- Added `has_parent` to request model

## 7.6 2.3.5

Date: 4/17/18

### 7.6.1 Bug Fixes

- Using *simplejson* package to fix JSON parsing issue in Python 3.4 & 3.5 (#48, #49)

## 7.7 2.3.4

Date: 4/5/18

### 7.7.1 New Features

- Python 3.4 is now supported (#43)
- Now using *Yapconf* for configuration parsing (#34)
- Parameter types can now be specified as native Python types (#29)
- Added flag to raise an exception if a request created with *SystemClient* completes with an 'ERROR' status (#28)

### 7.7.2 Other Changes

- All exceptions now inherit from *BrewtilsException* (#45)
- Removed references to *Brewmaster* exception classes (#44)
- Requests with JSON *command\_type* are smarter about formatting exceptions (#27)
- Decorators, *RemotePlugin*, and *SystemClient* can now be imported directly from the *brewtils* package

## 7.8 2.3.3

Date: 3/20/18

### 7.8.1 Bug Fixes

- Fixed bug where request updating could retry forever (#39)

## 7.9 2.3.2

Date: 3/7/18

### 7.9.1 Bug Fixes

- Fixed issue with multi-instance remote plugins failing to initialize (#35)

## 7.10 2.3.1

Date: 2/22/18

### 7.10.1 New Features

- Added `description` keyword argument to `@command` decorator

## 7.11 2.3.0

Date: 1/26/18

### 7.11.1 New Features

- Added methods for interacting with the Queue API to `RestClient` and `EasyClient`
- Clients and Plugins can now be configured to skip server certificate verification when making HTTPS requests
- Timestamps now have true millisecond precision on platforms that support it
- Added `form_input_type` to `Parameter` model
- Plugins can now be stopped correctly by calling their `_stop` method
- Added Event model

### 7.11.2 Bug Fixes

- Plugins now additionally look for `ca_cert` and `client_cert` in `BG_CA_CERT` and `BG_CLIENT_CERT`

### 7.11.3 Other Changes

- Better data integrity by only allowing certain Request status transitions

## 7.12 2.2.1

Date: 1/11/18

### 7.12.1 Bug Fixes

- Nested requests that reference a different beer-garden no longer fail

## 7.13 2.2.0

Date: 10/23/17

### 7.13.1 New Features

- Command descriptions can now be changed without updating the System version
- Standardized Remote Plugin logging configuration
- Added domain-specific language for dynamic choices configuration
- Added `metadata` field to Instance model

### 7.13.2 Bug Fixes

- Removed some default values from model `__init__` functions
- System descriptors (description, display name, icon name, metadata) now always updated during startup
- Requests with output type 'JSON' will now have JSON error messages

### 7.13.3 Other changes

- Added license file

## 7.14 2.1.1

Date: 8/25/17

### 7.14.1 New Features

- Added `updated_at` field to Request model
- `SystemClient` now allows specifying a `client_cert`
- `RestClient` now reuses the same session for subsequent connections
- `SystemClient` can now make non-blocking requests
- `RestClient` and `EasyClient` now support PATCHing a System

### 7.14.2 Deprecations / Removals

- `multithreaded` argument to `PluginBase` has been superseded by `max_concurrent`
- These decorators are now deprecated - `@command_registrar`, instead use `@system - @plugin_param`, instead use `@parameter - @register`, instead use `@command`
- These classes are now deprecated - `BrewmasterSchemaParser`, instead use `SchemaParser` - `BrewmasterRestClient`, instead use `RestClient` - `BrewmasterEasyClient`, instead use `EasyClient` - `BrewmasterSystemClient`, instead use `SystemClient`

### 7.14.3 Bug Fixes

- Reworked message processing to remove the possibility of a failed request being stuck in `IN_PROGRESS`
- Correctly handle custom form definitions with a top-level array
- Smarter reconnect logic when the RabbitMQ connection fails

### 7.14.4 Other changes

- Removed dependency on `pyopenssl` so there's need to compile any Python extensions
- Request processing now occurs inside of a `ThreadPoolExecutor` thread
- Better serialization handling for epoch fields