# Brewtils Documentation

***Release 2.4.10***

**Logan Asher Jones**

**Nov 12, 2019**

# Contents


**1 Brewtils** **3**
- 1.1 Features . . . 3
- 1.2 Installation . . . 3
- 1.3 Quick Start . . . 3
- 1.4 Documentation . . . 5

**2 Installation** **7**
- 2.1 Stable release . . . 7
- 2.2 From sources . . . 7

**3 Usage** **9**

**4 Brewtils** **11**
- 4.1 Features . . . 11
- 4.2 Installation . . . 11
- 4.3 Quick Start . . . 11
- 4.4 Documentation . . . 13

**5 brewtils** **15**
- 5.1 brewtils package . . . 15

**6 Contributing** **75**
- 6.1 Types of Contributions . . . 75
- 6.2 Get Started! . . . 76
- 6.3 Pull Request Guidelines . . . 77
- 6.4 Tips . . . 77

**7 Credits** **79**
- 7.1 Development Leads . . . 79
- 7.2 Contributors . . . 79

**8 Brewtils Changelog** **81**
- 8.1 2.4.10 . . . 81
- 8.2 2.4.9 . . . 81
- 8.3 2.4.8 . . . 81
- 8.4 2.4.7 . . . 82
- 8.5 2.4.6 . . . 82

8.6 2.4.5 . . . 82
8.7 2.4.4 . . . 83
8.8 2.4.3 . . . 83
8.9 2.4.2 . . . 83
8.10 2.4.1 . . . 83
8.11 2.4.0 . . . 84
8.12 2.3.7 . . . 84
8.13 2.3.6 . . . 84
8.14 2.3.5 . . . 85
8.15 2.3.4 . . . 85
8.16 2.3.3 . . . 85
8.17 2.3.2 . . . 86
8.18 2.3.1 . . . 86
8.19 2.3.0 . . . 86
8.20 2.2.1 . . . 86
8.21 2.2.0 . . . 87
8.22 2.1.1 . . . 87

**Python Module Index** **89**

**Index** **91**

Contents:

CHAPTER 1

# Brewtils

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

CHAPTER 2

# Installation

## 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

CHAPTER 4

# Brewtils

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

CHAPTER 5

# brewtils

## 5.1 brewtils package

### 5.1.1 Subpackages

#### brewtils.rest package

#### Submodules

#### brewtils.rest.client module

**class** brewtils.rest.client.**BrewmasterRestClient**(**args*, ***kwargs*)
Bases: *brewtils.rest.client.RestClient*

**class** brewtils.rest.client.**RestClient**(*bg_host=None*, *bg_port=None*, *ssl_enabled=False*, *api_version=None*, *logger=None*, *ca_cert=None*, *client_cert=None*, *url_prefix=None*, *ca_verify=True*, ***kwargs*)
Bases: object

Simple Rest Client for communicating to with beer-garden.

The is the low-level client responsible for making the actual REST calls. Other clients (e.g. *brewtils.rest.easy_client.EasyClient*) build on this by providing useful abstractions.

**Parameters**

- **bg_host** – beer-garden REST API hostname.
- **bg_port** – beer-garden REST API port.
- **ssl_enabled** – Flag indicating whether to use HTTPS when communicating with beer-garden.
- **api_version** – The beer-garden REST API version. Will default to the latest version.

- **logger** – The logger to use. If None one will be created.
- **ca_cert** – beer-garden REST API server CA certificate.
- **client_cert** – The client certificate to use when making requests.
- **url_prefix** – beer-garden REST API Url Prefix.
- **ca_verify** – Flag indicating whether to verify server certificate when making a request.
- **username** – Username for Beergarden authentication
- **password** – Password for Beergarden authentication
- **access_token** – Access token for Beergarden authentication
- **refresh_token** – Refresh token for Beergarden authentication
- **client_timeout** – Max time to will wait for server response

**JSON_HEADERS = {'Accept': 'text/plain', 'Content-type': 'application/json'}**

**LATEST_VERSION = 1**

**delete_instance**(**args*, ***kwargs*)
Performs a DELETE on an Instance URL

**Parameters** **instance_id** – The ID of the instance to remove

**Returns** Response to the request

**delete_job**(**args*, ***kwargs*)
Performs a DELETE on a Job URL

**Parameters** **job_id** – The ID of the job to remove

**Returns** Response to the request

**delete_queue**(**args*, ***kwargs*)
Performs a DELETE on a specific Queue URL

**Returns** Response to the request

**delete_queues**(**args*, ***kwargs*)
Performs a DELETE on the Queues URL

**Returns** Response to the request

**delete_system**(**args*, ***kwargs*)
Performs a DELETE on a System URL

**Parameters** **system_id** – The ID of the system to remove

**Returns** Response to the request

**get_command**(**args*, ***kwargs*)
Performs a GET on the Command URL

**Parameters** **command_id** – ID of command

**Returns** Response to the request

**get_commands**(**args*, ***kwargs*)
Performs a GET on the Commands URL

**get_config**(**args*, ***kwargs*)
Perform a GET to the config URL

**Parameters** **kwargs** – Passed to underlying Requests method

**Returns** The request response

**get_instance**(**args*, ***kwargs*)
Performs a GET on the Instance URL

**Parameters** **instance_id** – ID of instance

**Returns** Response to the request

**get_job**(**args*, ***kwargs*)
Performs a GET on the Job URL

**Parameters** **job_id** – ID of job

**Returns** Response to the request

**get_jobs**(**args*, ***kwargs*)
Performs a GET on the Jobs URL.

Returns: Response to the request

**get_logging_config**(**args*, ***kwargs*)
Perform a GET to the logging config URL

**Parameters** **kwargs** – Parameters to be used in the GET request

**Returns** The request response

**get_queues**(**args*, ***kwargs*)
Performs a GET on the Queues URL

**Returns** Response to the request

**get_request**(**args*, ***kwargs*)
Performs a GET on the Request URL

**Parameters** **request_id** – ID of request

**Returns** Response to the request

**get_requests**(**args*, ***kwargs*)
Performs a GET on the Requests URL

**Parameters** **kwargs** – Parameters to be used in the GET request

**Returns** Response to the request

**get_system**(**args*, ***kwargs*)
Performs a GET on the System URL

**Parameters**

- **system_id** – ID of system
- **kwargs** – Parameters to be used in the GET request

**Returns** Response to the request

**get_systems**(**args*, ***kwargs*)
Perform a GET on the System collection URL

**Parameters** **kwargs** – Parameters to be used in the GET request

**Returns** The request response

**get_tokens**(*username=None*, *password=None*)
Use a username and password to get access and refresh tokens

**Parameters**

- **username** – Beergarden username
- **password** – Beergarden password

**Returns** Response object

**get_user**(**args*, ***kwargs*)
Performs a GET on the specific User URL

**Returns** Response to the request

**Parameters** **user_identifier** – ID or username of User

**get_version**(**args*, ***kwargs*)
Perform a GET to the version URL

**Parameters** **kwargs** – Parameters to be used in the GET request

**Returns** The request response

**patch_instance**(**args*, ***kwargs*)
Performs a PATCH on the instance URL

**Parameters**

- **instance_id** – ID of instance
- **payload** – The update specification

**Returns** Response

**patch_job**(**args*, ***kwargs*)
Performs a PATCH on the Job URL

**Parameters**

- **job_id** – ID of request
- **payload** – New job definition

**Returns** Response to the request

**patch_request**(**args*, ***kwargs*)
Performs a PATCH on the Request URL

**Parameters**

- **request_id** – ID of request
- **payload** – New request definition

**Returns** Response to the request

**patch_system**(**args*, ***kwargs*)
Performs a PATCH on a System URL

**Parameters**

- **system_id** – ID of system
- **payload** – The update specification

**Returns** Response

**post_event**(**args*, ***kwargs*)
Performs a POST on the event URL

**Parameters**

- **payload** – New event definition
- **publishers** – Array of publishers to use

**Returns** Response to the request

**post_jobs**(**args*, ***kwargs*)
Performs a POST on the Job URL

**Parameters** **payload** – New job definition

**Returns** Response to the request

**post_requests**(**args*, ***kwargs*)
Performs a POST on the Request URL

**Parameters**

- **payload** – New request definition
- **kwargs** – Extra request parameters

**Keyword Arguments**

- **blocking** – Wait for request to complete
- **timeout** – Maximum seconds to wait

**Returns** Response to the request

**post_systems**(**args*, ***kwargs*)
Performs a POST on the System URL

**Parameters** **payload** – New request definition

**Returns** Response to the request

**refresh**(*refresh_token=None*)
Use a refresh token to obtain a new access token

**Parameters** **refresh_token** – Refresh token to use

**Returns** Response object

**class** brewtils.rest.client.**TimeoutAdapter**(***kwargs*)
Bases: requests.adapters.HTTPAdapter

Transport adapter with a default request timeout

**send**(**args*, ***kwargs*)
Sends PreparedRequest object. Returns Response object.

**Parameters**

- **request** – The PreparedRequest being sent.
- **stream** – (optional) Whether to stream the request content.
- **timeout** (*float or tuple or urllib3 Timeout object*) – (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.
- **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** – (optional) Any user-provided SSL certificate to be trusted.

- **proxies** – (optional) The proxies dictionary to apply to the request.

**Return type** requests.Response

brewtils.rest.client.**enable_auth**(*method*)
Decorate methods with this to enable using authentication

### brewtils.rest.easy_client module

**class** brewtils.rest.easy_client.**BrewmasterEasyClient**(**args*, ***kwargs*)
Bases: *brewtils.rest.easy_client.EasyClient*

**class** brewtils.rest.easy_client.**EasyClient**(*bg_host=None*, *bg_port=None*, *ssl_enabled=False*, *api_version=None*, *ca_cert=None*, *client_cert=None*, *parser=None*, *logger=None*, *url_prefix=None*, *ca_verify=True*, ***kwargs*)
Bases: object

Client for simplified communication with Beergarden

This class is intended to be a middle ground between the RestClient and SystemClient. It provides a 'cleaner' interface to some common Beergarden operations than is exposed by the lower-level RestClient. On the other hand, the SystemClient is much better for generating Beergarden Requests.

**Keyword Arguments**

- **bg_host** (*str*) – Beergarden hostname
- **bg_port** (*int*) – Beergarden port
- **ssl_enabled** (*Optional[bool]*) – Whether to use SSL (HTTP vs HTTPS)
- **api_version** (*Optional[int]*) – The REST API version
- **ca_cert** (*Optional[str]*) – Path to CA certificate file
- **client_cert** (*Optional[str]*) – Path to client certificate file
- **parser** (*Optional[SchemaParser]*) – Parser to use
- **logger** (*Optional[Logger]*) – Logger to use
- **url_prefix** (*Optional[str]*) – Beergarden REST API prefix
- **ca_verify** (*Optional[bool]*) – Whether to verify the server cert hostname
- **username** (*Optional[str]*) – Username for authentication
- **password** (*Optional[str]*) – Password for authentication
- **access_token** (*Optional[str]*) – Access token for authentication
- **refresh_token** (*Optional[str]*) – Refresh token for authentication
- **client_timeout** (*Optional[float]*) – Max time to wait for a server response

**can_connect**(***kwargs*)
Determine if the Beergarden server is responding.

**Kwargs:** Arguments passed to the underlying Requests method

**Returns** A bool indicating if the connection attempt was successful. Will return False only if a ConnectionError is raised during the attempt. Any other exception will be re-raised.

**Raises** `requests.exceptions.RequestException` – The connection attempt resulted in an exception that indicates something other than a basic connection error. For example, an error with certificate verification.

**clear_all_queues**()
Cancel and remove all Requests in all queues

**Returns** True if the clear was successful

**Return type** bool

**clear_queue**(*queue_name*)
Cancel and remove all Requests from a message queue

**Parameters** **queue_name** (*str*) – The name of the queue to clear

**Returns** True if the clear was successful

**Return type** bool

**create_job**(*job*)
Create a new Job

**Parameters** **job** (Job) – New Job definition

**Returns** The newly-created Job

**Return type** *Job*

**create_request**(*request*, ***kwargs*)
Create a new Request

**Parameters**

- **request** – New request definition
- **kwargs** – Extra request parameters

**Keyword Arguments**

- **blocking** (*bool*) – Wait for request to complete before returning
- **timeout** (*int*) – Maximum seconds to wait for completion

**Returns** The newly-created Request

**Return type** *Request*

**create_system**(*system*)
Create a new System

**Parameters** **system** (System) – The System to create

**Returns** The newly-created system

**Return type** *System*

**find_jobs**(***kwargs*)
Find Jobs using keyword arguments as search parameters

**Parameters** ****kwargs** – Search parameters

**Returns** List of Jobs matching the search parameters

**Return type** List[*Job*]

**find_requests**(***kwargs*)
Find Requests using keyword arguments as search parameters

**Parameters** ****kwargs** – Search parameters

**Returns** List of Systems matching the search parameters

**Return type** List[*Request*]

**find_systems**(***kwargs*)
Find Systems using keyword arguments as search parameters

**Parameters** ****kwargs** – Search parameters

**Returns** List of Systems matching the search parameters

**Return type** List[*System*]

**find_unique_request**(***kwargs*)
Find a unique request

**Note:** If 'id' is a given keyword argument then all other parameters will be ignored.

**Parameters** ****kwargs** – Search parameters

**Returns** The Request if found, None otherwise

**Return type** *Request*, None

**Raises** `FetchError` – More than one matching Request was found

**find_unique_system**(***kwargs*)
Find a unique system

**Note:** If 'id' is a given keyword argument then all other parameters will be ignored.

**Parameters** ****kwargs** – Search parameters

**Returns** The System if found, None otherwise

**Return type** *System*, None

**Raises** `FetchError` – More than one matching System was found

**get_instance**(*instance_id*)
Get an Instance

**Parameters** **instance_id** – The Id

**Returns** The Instance

**get_instance_status**(*instance_id*)
Get an Instance

WARNING: This method currently returns the Instance, not the Instance's status. This behavior will be corrected in 3.0.

To prepare for this change please use get_instance() instead of this method.

**Parameters** **instance_id** – The Id

**Returns** The status

**get_logging_config**(*system_name*)
Get logging configuration for a System

**Parameters** **system_name** (*str*) – The name of the System

**Returns** The configuration object

**Return type** *LoggingConfig*

**get_queues**()
Retrieve all queue information

**Returns** The response

**get_user**(*user_identifier*)
Find a user

**Parameters** **user_identifier** (*str*) – User ID or username

**Returns** The User

**Return type** *Principal*

**get_version**(***kwargs*)
Get Bartender, Brew-view, and API version information

**Parameters** ****kwargs** – Extra parameters

**Returns** Response object with version information in the body

**Return type** dict

**initialize_instance**(*instance_id*)
Start an Instance

**Parameters** **instance_id** (*str*) – The Instance ID

**Returns** The updated Instance

**Return type** *Instance*

**instance_heartbeat**(*instance_id*)
Send an Instance heartbeat

**Parameters** **instance_id** (*str*) – The Instance ID

**Returns** True if the heartbeat was successful

**Return type** bool

**pause_job**(*job_id*)
Pause a Job

**Parameters** **job_id** (*str*) – The Job ID

**Returns** The updated Job

**Return type** *Job*

**publish_event**(**args*, ***kwargs*)
Publish a new event

**Parameters**

- ***args** – If a positional argument is given it's assumed to be an Event and will be used
- ****kwargs** – Will be used to construct a new Event to publish if no Event is given in the positional arguments

**Keyword Arguments** **_publishers** (*Optional[List[str]]*) – List of publisher names. If given the Event will only be published to the specified publishers. Otherwise all publishers known to Beergarden will be used.

**Returns** True if the publish was successful

**Return type** bool

**remove_instance**(*instance_id*)
Remove an Instance

**Parameters** **instance_id** (*str*) – The Instance ID

**Returns** True if the remove was successful

**Return type** bool

**remove_job**(*job_id*)
Remove a unique Job

**Parameters** **job_id** (*str*) – The Job ID

**Returns** True if removal was successful

**Return type** bool

**Raises** DeleteError – Couldn't remove Job

**remove_system**(***kwargs*)
Remove a unique System

**Parameters** ****kwargs** – Search parameters

**Returns** True if removal was successful

**Return type** bool

**Raises** FetchError – Couldn't find a System matching given parameters

**resume_job**(*job_id*)
Resume a Job

**Parameters** **job_id** (*str*) – The Job ID

**Returns** The updated Job

**Return type** *Job*

**update_instance_status**(*instance_id*, *new_status*)
Update an Instance status

**Parameters**

- **instance_id** (*str*) – The Instance ID
- **new_status** (*str*) – The new status

**Returns** The updated Instance

**Return type** *Instance*

**update_request**(*request_id*, *status=None*, *output=None*, *error_class=None*)
Update a Request

**Parameters**

- **request_id** (*str*) – The Request ID
- **status** (*Optional[str]*) – New Request status

- **output** (*Optional[str]*) – New Request output
- **error_class** (*Optional[str]*) – New Request error class

**Returns** The updated response

**Return type** Response

**update_system**(*system_id*, *new_commands=None*, ***kwargs*)
Update a System

**Parameters**

- **system_id** (*str*) – The System ID
- **new_commands** (*Optional[List[Command]]*) – New System commands

**Keyword Arguments**

- **metadata** (*dict*) – New System metadata
- **description** (*str*) – New System description
- **display_name** (*str*) – New System display name
- **icon_name** (*str*) – New System icon name

**Returns** The updated system

**Return type** *System*

**who_am_i**()
Find user using the current set of credentials

**Returns** The User

**Return type** *Principal*

brewtils.rest.easy_client.**handle_response_failure**(*response*, *default_exc=<class 'brewtils.errors.RestError'>*, *raise_404=True*)
Deal with a response with non-2xx status code

**Parameters**

- **response** – The response object
- **default_exc** – The exception to raise if no specific exception is warranted
- **raise_404** – If True a response with status code 404 will raise a NotFoundError. If False the method will return None.

**Returns** None - this function will always raise

**Raises**

- NotFoundError – Status code 404 and raise_404 is True
- WaitExceededError – Status code 408
- ConflictError – Status code 409
- ValidationError – Any other 4xx status codes
- RestConnectionError – Status code 503
- default_exc – Any other status code

brewtils.rest.easy_client.**wrap_response**(*return_boolean=False*, *parse_method=''*, *parse_many=False*, *default_exc=<class 'brewtils.errors.RestError'>*, *raise_404=True*)

Decorator to consolidate response parsing and error handling

**Parameters**

- **return_boolean** – If True, a successful response will also return True
- **parse_method** – The response's json will be passed to this method of the SchemaParser
- **parse_many** – This will be passed as the 'many' parameter when parsing the response
- **default_exc** – Will be passed to handle_response_failure for failed responses
- **raise_404** – Will be passed to handle_response_failure for failed responses

**Returns**

- True if return_boolean is True and the response status code is 2xx.
- The response object if return_boolean is False and parse_method is ""
- A parsed Brewtils model if return_boolean is False and parse_method is defined

**Raises** RestError – The response has a non-2xx status code. Note that the specific exception raised depends on the response status code and the argument passed as the default_exc parameter.

### brewtils.rest.system_client module

**class** brewtils.rest.system_client.**BrewmasterSystemClient**(**args*, ***kwargs*)

Bases: *brewtils.rest.system_client.SystemClient*

**class** brewtils.rest.system_client.**SystemClient**(*bg_host=None*, *bg_port=None*, *system_name=None*, *version_constraint='latest'*, *default_instance='default'*, *always_update=False*, *timeout=None*, *max_delay=30*, *api_version=None*, *ssl_enabled=False*, *ca_cert=None*, *blocking=True*, *max_concurrent=None*, *client_cert=None*, *url_prefix=None*, *ca_verify=True*, *raise_on_error=False*, ***kwargs*)

Bases: object

High-level client for generating requests for a beer-garden System.

**SystemClient creation:** This class is intended to be the main way to create beer-garden requests. Create an instance with beer-garden connection information (optionally including a url_prefix) and a system name:

```
client = SystemClient(host, port, 'example_system', ssl_enabled=True, url_
↪prefix=None)
```

Pass additional keyword arguments for more granularity:

**version_constraint:** Allows specifying a particular system version. Can be a version literal ('1.0.0') or the special value 'latest.' Using 'latest' will allow the the SystemClient to retry a request if it fails due to a missing system (see Creating Requests).

**default_instance:** The instance name to use when creating a request if no other instance name is specified. Since each request must be addressed to a specific instance this is a convenience to prevent needing to specify the 'default' instance for each request.

**always_update:** Always attempt to reload the system definition before making a request. This is useful to ensure Requests are always made against the latest version of the system. If not set the System definition will be loaded once (upon making the first request) and then only reloaded if a Request fails.

**Loading the System:** The System definition is lazily loaded, so nothing happens until the first attempt to send a Request. At that point the SystemClient will query beer-garden to get a system definition that matches the system_name and version_constraint. If no matching system can be found a FetchError will be raised. If always_update was set to True this will happen before making each request, not just the first.

**Making a Request:** The standard way to create and send requests is by calling object attributes:

```
request = client.example_command(param_1='example_param')
```

In the normal case this will block until the request completes. Request completion is determined by periodically polling beer-garden to check the Request status. The time between polling requests starts at 0.5s and doubles each time the request has still not completed, up to max_delay. If a timeout was specified and the Request has not completed within that time a `ConnectionTimeoutError` will be raised.

It is also possible to create the SystemClient in non-blocking mode by specifying blocking=False. In this case the request creation will immediately return a Future and will spawn a separate thread to poll for Request completion. The max_concurrent parameter is used to control the maximum threads available for polling.

```
# Create a SystemClient with blocking=False
client = SystemClient(host, port, 'example_system', ssl_enabled=True,␣
↪blocking=False)

# Create and send 5 requests without waiting for request completion
futures = [client.example_command(param_1=number) for number in range(5)]

# Now wait on all requests to complete
concurrent.futures.wait(futures)
```

If the request creation process fails (e.g. the command failed validation) and version_constraint is 'latest' then the SystemClient will check to see if a different version is available, and if so it will attempt to make the request on that version. This is so users of the SystemClient that don't necessarily care about the target system version don't need to be restarted if the target system is updated.

**Tweaking beer-garden Request Parameters:** There are several parameters that control how beer-garden routes / processes a request. To denote these as intended for beer-garden itself (rather than a parameter to be passed to the Plugin) prepend a leading underscore to the argument name.

Sending to another instance:

```
request = client.example_command(_instance_name='instance_2', param_1=
↪'example_param')
```

Request with a comment:

```
request = client.example_command(_comment='I'm a beer-garden comment!',
                                 param_1='example_param')
```

Without the leading underscore the arguments would be treated the same as param_1 - another parameter to be passed to the plugin.

**Parameters**

- **host** – beer-garden REST API hostname.
- **port** – beer-garden REST API port.
- **system_name** – The name of the system to use.
- **version_constraint** – The system version to use. Can be specific or 'latest'.
- **default_instance** – The instance to use if not specified when creating a request.
- **always_update** – Should check for a newer System version before each request.
- **timeout** – Length of time to wait for a request to complete. 'None' means wait forever.
- **max_delay** – Maximum time to wait between checking the status of a created request.
- **api_version** – beer-garden API version.
- **ssl_enabled** – Flag indicating whether to use HTTPS when communicating with beer-garden.
- **ca_cert** – beer-garden REST API server CA certificate.
- **blocking** – Block after request creation until the request completes.
- **max_concurrent** – Maximum number of concurrent requests allowed.
- **client_cert** – The client certificate to use when making requests.
- **url_prefix** – beer-garden REST API URL Prefix.
- **ca_verify** – Flag indicating whether to verify server certificate when making a request.
- **raise_on_error** – Raises an error if the request ends in an error state.
- **username** – Username for Beergarden authentication
- **password** – Password for Beergarden authentication
- **access_token** – Access token for Beergarden authentication
- **refresh_token** – Refresh token for Beergarden authentication
- **client_timeout** – Max time to will wait for server response

**create_bg_request**(*command_name*, ***kwargs*)

Create a callable that will execute a beer-garden request when called.

Normally you interact with the SystemClient by accessing attributes, but there could be certain cases where you want to create a request without sending it.

Example:

```
client = SystemClient(host, port, 'system', blocking=False)
requests = []

# No arguments
requests.append(client.create_bg_request('command_1'))

# arg_1 will be passed as a parameter
requests.append(client.create_bg_request('command_2', arg_1='Hi!'))
```

(continues on next page)

(continued from previous page)

```
futures = [request() for request in requests]    # Calling creates and sends␣
↪the request
concurrent.futures.wait(futures)                 # Wait for all the futures to␣
↪complete
```

**Parameters**

- **command_name** – The name of the command that will be sent.
- **kwargs** – Additional arguments to pass to send_bg_request.

**Raises** **AttributeError** – The system does not have a command with the given command_name.

**Returns** A partial that will create and execute a beer-garden request when called.

**load_bg_system**()

Query beer-garden for a System definition

This method will make the query to beer-garden for a System matching the name and version constraints specified during SystemClient instance creation.

If this method completes successfully the SystemClient will be ready to create and send Requests.

**Raises** ***FetchError*** – If unable to find a matching System

**Returns** None

**send_bg_request**(***kwargs*)

Actually create a Request and send it to beer-garden

**Note:** This method is intended for advanced use only, mainly cases where you're using the SystemClient without a predefined System. It assumes that everything needed to construct the request is being passed in kwargs. If this doesn't sound like what you want you should check out create_bg_request.

**Parameters** **kwargs** – All necessary request parameters, including beer-garden internal parameters

**Raises** ***ValidationError*** – If the Request creation failed validation on the server

**Returns** If the SystemClient was created with blocking=True a completed request object, otherwise a Future that will return the Request when it completes.

## Module contents

brewtils.rest.**normalize_url_prefix**(*url_prefix*)

Enforce a consistent URL representation

The normalized prefix will begin and end with '/'. If there is no prefix the normalized form will be '/'.

### Examples

INPUT NORMALIZED None '/' '' '/' '/' '/' 'example' '/example/' '/example' '/example/' 'example/' '/example/' '/example/' '/example/'

**Parameters** **url_prefix** (*str*) – The prefix

**Returns** The normalized prefix

**Return type** str

### 5.1.2 Submodules

### 5.1.3 brewtils.choices module

**class** brewtils.choices.**FunctionTransformer**
Bases: lark.visitors.Transformer

**static arg_pair**(*s*)

**static func**(*s*)

**func_args**
alias of __builtin__.list

**static reference**(*s*)

**static url**(*s*)

**url_args**
alias of __builtin__.list

brewtils.choices.**parse**(*input_string*, *parse_as=None*)
Attempt to parse a string into a choices dictionary.

**Parameters**

- **input_string** – The string to parse
- **parse_as** – String specifying how to parse *input_string*. Valid values are 'func' or 'url'. Will try all valid values if None.

**Returns** A dictionary containing the results of the parse

**Raises** **lark.common.ParseError** – The parser was not able to find a valid parsing of *input_string*

### 5.1.4 brewtils.decorators module

brewtils.decorators.**system**(*cls=None*, *bg_name=None*, *bg_version=None*)
Class decorator that marks a class as a beer-garden System

**Creates some properties on the class:**

- _commands: holds all registered commands
- _name: an optional system name
- _version: an optional system version
- _current_request: Reference to the currently executing request

**Parameters**

- **cls** – The class to decorated
- **bg_name** – Optional plugin name
- **bg_version** – Optional plugin version

**Returns** The decorated class

brewtils.decorators.**parameter**(*_wrapped=None*, *key=None*, *type=None*, *multi=None*, *display_name=None*, *optional=None*, *default=None*, *description=None*, *choices=None*, *nullable=None*, *maximum=None*, *minimum=None*, *regex=None*, *is_kwarg=None*, *model=None*, *form_input_type=None*)

Decorator that enables Parameter specifications for a beer-garden Command

This decorator is intended to be used when more specification is desired for a Parameter.

For example:

```
@parameter(key="message", description="Message to echo", optional=True, type=
↪"String",
           default="Hello, World!")
def echo(self, message):
    return message
```

**Parameters**

- **_wrapped** – The function to decorate. This is handled as a positional argument and shouldn't be explicitly set.
- **key** – String specifying the parameter identifier. Must match an argument name of the decorated function.
- **type** – String indicating the type to use for this parameter.
- **multi** – Boolean indicating if this parameter is a multi. See documentation for discussion of what this means.
- **display_name** – String that will be displayed as a label in the user interface.
- **optional** – Boolean indicating if this parameter must be specified.
- **default** – The value this parameter will be assigned if not overridden when creating a request.
- **description** – An additional string that will be displayed in the user interface.
- **choices** – List or dictionary specifying allowed values. See documentation for more information.
- **nullable** – Boolean indicating if this parameter is allowed to be null.
- **maximum** – Integer indicating the maximum value of the parameter.
- **minimum** – Integer indicating the minimum value of the parameter.
- **regex** – String describing a regular expression constraint on the parameter.
- **is_kwarg** – Boolean indicating if this parameter is meant to be part of the decorated function's kwargs.
- **model** – Class to be used as a model for this parameter. Must be a Python type object, not an instance.
- **form_input_type** – Only used for string fields. Changes the form input field (e.g. textarea)

**Returns** The decorated function.

brewtils.decorators.**parameters**(**args*)
Specify multiple Parameter definitions at once

This can be useful for commands which have a large number of complicated parameters but aren't good candidates for a Model.

```
@parameter(**params[cmd1][param1])
@parameter(**params[cmd1][param2])
@parameter(**params[cmd1][param3])
def cmd1(self, **kwargs):
    pass
```

Can become:

```
@parameters(params[cmd1])
def cmd1(self, **kwargs):
    pass
```

**Parameters** ***args** (*iterable*) – Positional arguments The first (and only) positional argument must be a list containing dictionaries that describe parameters.

**Returns** The decorated function

**Return type** func

brewtils.decorators.**command**(*_wrapped=None*, *command_type='ACTION'*, *output_type='STRING'*, *schema=None*, *form=None*, *template=None*, *icon_name=None*, *description=None*)
Decorator that marks a function as a beer-garden command

For example:

```
@command(output_type='JSON')
def echo_json(self, message):
    return message
```

**Parameters**

- **_wrapped** – The function to decorate. This is handled as a positional argument and shouldn't be explicitly set.
- **command_type** – The command type. Valid options are Command.COMMAND_TYPES.
- **output_type** – The output type. Valid options are Command.OUTPUT_TYPES.
- **schema** – A custom schema definition.
- **form** – A custom form definition.
- **template** – A custom template definition.
- **icon_name** – The icon name. Should be either a FontAwesome or a Glyphicon name.
- **description** – The command description. Will override the function's docstring.

**Returns** The decorated function.

brewtils.decorators.**command_registrar**(*cls=None*, *bg_name=None*, *bg_version=None*)
Class decorator that marks a class as a beer-garden System

**Creates some properties on the class:**

- `_commands`: holds all registered commands
- `_name`: an optional system name
- `_version`: an optional system version
- `_current_request`: Reference to the currently executing request

**Parameters**

- **cls** – The class to decorated
- **bg_name** – Optional plugin name
- **bg_version** – Optional plugin version

**Returns** The decorated class

brewtils.decorators.**plugin_param**(*_wrapped=None*, *key=None*, *type=None*, *multi=None*, *display_name=None*, *optional=None*, *default=None*, *description=None*, *choices=None*, *nullable=None*, *maximum=None*, *minimum=None*, *regex=None*, *is_kwarg=None*, *model=None*, *form_input_type=None*)

Decorator that enables Parameter specifications for a beer-garden Command

This decorator is intended to be used when more specification is desired for a Parameter.

For example:

```
@parameter(key="message", description="Message to echo", optional=True, type=
↪"String",
           default="Hello, World!")
def echo(self, message):
    return message
```

**Parameters**

- **_wrapped** – The function to decorate. This is handled as a positional argument and shouldn't be explicitly set.
- **key** – String specifying the parameter identifier. Must match an argument name of the decorated function.
- **type** – String indicating the type to use for this parameter.
- **multi** – Boolean indicating if this parameter is a multi. See documentation for discussion of what this means.
- **display_name** – String that will be displayed as a label in the user interface.
- **optional** – Boolean indicating if this parameter must be specified.
- **default** – The value this parameter will be assigned if not overridden when creating a request.
- **description** – An additional string that will be displayed in the user interface.
- **choices** – List or dictionary specifying allowed values. See documentation for more information.
- **nullable** – Boolean indicating if this parameter is allowed to be null.
- **maximum** – Integer indicating the maximum value of the parameter.
- **minimum** – Integer indicating the minimum value of the parameter.

- **regex** – String describing a regular expression constraint on the parameter.
- **is_kwarg** – Boolean indicating if this parameter is meant to be part of the decorated function's kwargs.
- **model** – Class to be used as a model for this parameter. Must be a Python type object, not an instance.
- **form_input_type** – Only used for string fields. Changes the form input field (e.g. textarea)

**Returns** The decorated function.

brewtils.decorators.**register**(*_wrapped=None*, *command_type='ACTION'*, *output_type='STRING'*, *schema=None*, *form=None*, *template=None*, *icon_name=None*, *description=None*)

Decorator that marks a function as a beer-garden command

For example:

```
@command(output_type='JSON')
def echo_json(self, message):
    return message
```

**Parameters**

- **_wrapped** – The function to decorate. This is handled as a positional argument and shouldn't be explicitly set.
- **command_type** – The command type. Valid options are Command.COMMAND_TYPES.
- **output_type** – The output type. Valid options are Command.OUTPUT_TYPES.
- **schema** – A custom schema definition.
- **form** – A custom form definition.
- **template** – A custom template definition.
- **icon_name** – The icon name. Should be either a FontAwesome or a Glyphicon name.
- **description** – The command description. Will override the function's docstring.

**Returns** The decorated function.

### 5.1.5 brewtils.errors module

**exception** brewtils.errors.**AckAndContinueException**

Bases: *brewtils.errors.RequestProcessException*

**exception** brewtils.errors.**AckAndDieException**

Bases: *brewtils.errors.RequestProcessException*

**exception** brewtils.errors.**AuthorizationRequired**

Bases: *brewtils.errors.RestClientError*

Error indicating a 401 was raised on the server

brewtils.errors.**BGConflictError**

alias of *brewtils.errors.ConflictError*

brewtils.errors.**BGNotFoundError**
alias of *brewtils.errors.NotFoundError*

brewtils.errors.**BGRequestFailedError**
alias of *brewtils.errors.RequestFailedError*

brewtils.errors.**BrewmasterConnectionError**
alias of *brewtils.errors.RestConnectionError*

brewtils.errors.**BrewmasterDeleteError**
alias of *brewtils.errors.DeleteError*

brewtils.errors.**BrewmasterFetchError**
alias of *brewtils.errors.FetchError*

brewtils.errors.**BrewmasterModelError**
alias of *brewtils.errors.ModelError*

brewtils.errors.**BrewmasterModelValidationError**
alias of *brewtils.errors.ModelValidationError*

brewtils.errors.**BrewmasterRestClientError**
alias of *brewtils.errors.RestClientError*

brewtils.errors.**BrewmasterRestError**
alias of *brewtils.errors.RestError*

brewtils.errors.**BrewmasterRestServerError**
alias of *brewtils.errors.RestServerError*

brewtils.errors.**BrewmasterSaveError**
alias of *brewtils.errors.SaveError*

brewtils.errors.**BrewmasterTimeoutError**
alias of *brewtils.errors.TimeoutExceededError*

brewtils.errors.**BrewmasterValidationError**
alias of *brewtils.errors.ValidationError*

**exception** brewtils.errors.**BrewtilsException**
Bases: exceptions.Exception

Base exception

**exception** brewtils.errors.**ConflictError**
Bases: *brewtils.errors.RestClientError*

Error indicating a 409 was raised on the server

brewtils.errors.**ConnectionTimeoutError**
alias of *brewtils.errors.TimeoutExceededError*

**exception** brewtils.errors.**DeleteError**
Bases: *brewtils.errors.RestServerError*

Error Indicating a server Error occurred performing a DELETE

**exception** brewtils.errors.**DiscardMessageException**
Bases: *brewtils.errors.RequestProcessException*

Raising an instance will result in a message not being requeued

**exception** brewtils.errors.**ErrorLogLevelCritical**
Bases: exceptions.Exception

Mixin to log an exception at the CRITICAL level

**exception** brewtils.errors.**ErrorLogLevelDebug**
Bases: exceptions.Exception

Mixin to log an exception at the DEBUG level

**exception** brewtils.errors.**ErrorLogLevelError**
Bases: exceptions.Exception

Mixin to log an exception at the ERROR level

**exception** brewtils.errors.**ErrorLogLevelInfo**
Bases: exceptions.Exception

Mixin to log an exception at the INFO level

**exception** brewtils.errors.**ErrorLogLevelWarning**
Bases: exceptions.Exception

Mixin to log an exception at the WARNING level

**exception** brewtils.errors.**FetchError**
Bases: *brewtils.errors.RestError*

Error Indicating a server Error occurred performing a GET

**exception** brewtils.errors.**ModelError**
Bases: *brewtils.errors.BrewtilsException*

Base exception for model errors

**exception** brewtils.errors.**ModelValidationError**
Bases: *brewtils.errors.ModelError*

Invalid model

**exception** brewtils.errors.**NoAckAndDieException**
Bases: *brewtils.errors.RequestProcessException*

**exception** brewtils.errors.**NotFoundError**
Bases: *brewtils.errors.RestClientError*

Error Indicating a 404 was raised on the server

**exception** brewtils.errors.**PluginError**
Bases: *brewtils.errors.BrewtilsException*

Generic error class

**exception** brewtils.errors.**PluginParamError**
Bases: *brewtils.errors.PluginError*

Error used when plugins have illegal parameters

**exception** brewtils.errors.**PluginValidationError**
Bases: *brewtils.errors.PluginError*

Plugin could not be validated successfully

**exception** brewtils.errors.**RepublishRequestException**(*request*, *headers*)
Bases: *brewtils.errors.RequestProcessException*

Republish to the end of the message queue

**Parameters**

- **request** (*brewtils.models.Request*) – The Request to republish
- **headers** – A dictionary of headers to be used by *brewtils.request_consumer.RequestConsumer*

**exception** brewtils.errors.**RequestFailedError**(*request*)
Bases: *brewtils.errors.RestError*

Request returned with a 200, but the status was ERROR

**exception** brewtils.errors.**RequestForbidden**
Bases: *brewtils.errors.RestClientError*

Error indicating a 403 was raised on the server

**exception** brewtils.errors.**RequestProcessException**
Bases: *brewtils.errors.BrewtilsException*

Base for exceptions that occur during request processing

**exception** brewtils.errors.**RequestProcessingError**
Bases: *brewtils.errors.AckAndContinueException*

**exception** brewtils.errors.**RequestPublishException**
Bases: *brewtils.errors.BrewtilsException*

Error while publishing request

**exception** brewtils.errors.**RequestStatusTransitionError**
Bases: *brewtils.errors.ModelValidationError*

A status update was an invalid transition

**exception** brewtils.errors.**RestClientError**
Bases: *brewtils.errors.RestError*

Wrapper for all 4XX errors

**exception** brewtils.errors.**RestConnectionError**
Bases: *brewtils.errors.RestServerError*

Error indicating a connection error while performing a request

**exception** brewtils.errors.**RestError**
Bases: *brewtils.errors.BrewtilsException*

Base exception for REST errors

**exception** brewtils.errors.**RestServerError**
Bases: *brewtils.errors.RestError*

Wrapper for all 5XX errors

**exception** brewtils.errors.**SaveError**
Bases: *brewtils.errors.RestServerError*

Error Indicating a server Error occurred performing a POST/PUT

**exception** brewtils.errors.**SuppressStacktrace**
Bases: exceptions.Exception

Mixin that will suppress stacktrace logging

**exception** brewtils.errors.**TimeoutExceededError**
Bases: *brewtils.errors.RestClientError*

Error indicating a timeout occurred waiting for a request to complete

**exception** brewtils.errors.**ValidationError**
Bases: *brewtils.errors.RestClientError*

Error Indicating a client (400) Error occurred performing a POST/PUT

brewtils.errors.**WaitExceededError**
alias of *brewtils.errors.TimeoutExceededError*

brewtils.errors.**parse_exception_as_json**(*exc*)
Attempt to parse an Exception to a JSON string.

If the exception has a single argument, no attributes, and the attribute can be converted to a valid JSON string, then that will be returned.

Otherwise, a string version of the following form will be returned:

**{** "message": "", "arguments": [], "attributes": {}

}

Where "message" is just str(exc), "arguments" is a list of all the arguments passed to the exception attempted to be converted to a valid JSON string, and "attributes" are the attributes of the exception class.

If parsing fails at all, then a simple str() will be applied either the argument or attribute value.

---

**Note:** On python version 2, errors with custom attributes do not list those attributes as arguments.

---

**Parameters** **exc** (*Exception*) – The exception you would like to format as JSON.

**Raises** ValueError – If the exception passed in is not an Exception.

**Returns** A valid JSON string representing (the best we can) the exception.

### 5.1.6 brewtils.log module

Brewtils Logging Utilities

This module streamlines loading logging configuration from Beergarden.

#### Example

To use this just call configure_logging sometime before you initialize your Plugin object:

```
from brewtils import configure_logging, get_connection_info, Plugin

# Load BG connection info from environment and command line args
connection_info = get_connection_info(sys.argv[1:])

configure_logging(system_name='systemX', **connection_info)

plugin = Plugin(
    my_client,
    name='systemX,
    version='0.0.1',
    **connection_info
)
plugin.run()
```

brewtils.log.**configure_logging**(*system_name=None*, ***kwargs*)
Load and enable a logging configuration from Beergarden

NOTE: This method will overwrite the current logging configuration.

**Parameters**

- **system_name** – Name of the system to load
- ****kwargs** – Beergarden connection parameters

**Returns** None

brewtils.log.**convert_logging_config**(*logging_config*)
Transform a LoggingConfig object into a Python logging configuration

**Parameters** **logging_config** – Beergarden logging config

**Returns** The logging configuration

**Return type** dict

brewtils.log.**get_logging_config**(*system_name=None*, ***kwargs*)
Retrieve a logging configuration from Beergarden

**Parameters**

- **system_name** – Name of the system to load
- ****kwargs** – Beergarden connection parameters

**Returns** The logging configuration for the specified system

**Return type** dict

brewtils.log.**get_python_logging_config**(*bg_host*, *bg_port*, *system_name*, *ca_cert=None*, *client_cert=None*, *ssl_enabled=None*)
DEPRECATED: Get Beergarden's logging configuration

This method is deprecated - consider using `get_logging_config()`

**Parameters**

- **bg_host** (*str*) – Beergarden host
- **bg_port** (*int*) – Beergarden port
- **system_name** (*str*) – Name of the system
- **ca_cert** (*str*) – Path to CA certificate file
- **client_cert** (*str*) – Path to client certificate file
- **ssl_enabled** (*bool*) – Use SSL when connection to Beergarden

**Returns** The logging configuration for the specified system

**Return type** dict

brewtils.log.**setup_logger**(*bg_host*, *bg_port*, *system_name*, *ca_cert=None*, *client_cert=None*, *ssl_enabled=None*)
DEPRECATED: Set Python logging to use configuration from Beergarden API

This method is deprecated - consider using `configure_logging()`

This method will overwrite the current logging configuration.

**Parameters**

- **bg_host** (*str*) – Beergarden host

- **bg_port** (*int*) – Beergarden port
- **system_name** (*str*) – Name of the system
- **ca_cert** (*str*) – Path to CA certificate file
- **client_cert** (*str*) – Path to client certificate file
- **ssl_enabled** (*bool*) – Use SSL when connection to Beergarden

Returns: None

### 5.1.7 brewtils.models module

**class** brewtils.models.**System**(*name=None*, *description=None*, *version=None*, *id=None*, *max_instances=None*, *instances=None*, *commands=None*, *icon_name=None*, *display_name=None*, *metadata=None*)

Bases: object

**get_command_by_name**(*command_name*)

Retrieve a particular command from the system

**Parameters** **command_name** – Name of the command to retrieve

**Returns** The command object. None if the given command name does not exist in this system.

**get_instance**(*name*)

Get an instance that currently exists in the system

**Parameters** **name** – The name of the instance to search

**Returns** The instance with the given name exists for this system, None otherwise

**has_different_commands**(*commands*)

Check if a set of commands is different than the current commands

**Parameters** **commands** – The set commands to compare against the current set

**Returns** True if the sets are different, False if the sets are the same

**has_instance**(*name*)

Determine if an instance currently exists in the system

**Parameters** **name** – The name of the instance to search

**Returns** True if an instance with the given name exists for this system, False otherwise.

**instance_names**

**schema = 'SystemSchema'**

**class** brewtils.models.**Instance**(*name=None*, *description=None*, *id=None*, *status=None*, *status_info=None*, *queue_type=None*, *queue_info=None*, *icon_name=None*, *metadata=None*)

Bases: object

**INSTANCE_STATUSES = set(['DEAD', 'INITIALIZING', 'PAUSED', 'RUNNING', 'STARTING', 'STO**

**schema = 'InstanceSchema'**

**class** brewtils.models.**Command**(*name=None*, *description=None*, *id=None*, *parameters=None*, *command_type=None*, *output_type=None*, *schema=None*, *form=None*, *template=None*, *icon_name=None*, *system=None*)

Bases: object

**COMMAND_TYPES = ('ACTION', 'INFO', 'EPHEMERAL')**

**OUTPUT_TYPES = ('STRING', 'JSON', 'XML', 'HTML')**

**get_parameter_by_key**(*key*)
Given a Key, it will return the parameter (or None) with that key

**Parameters** **key** –

**Return parameter**

**has_different_parameters**(*parameters*)
Given a set of parameters, determines if the parameters provided differ from the parameters already defined on this command.

**Parameters** **parameters** –

**Return boolean**

**parameter_keys**()
Convenience Method for returning all the keys of this command's parameters.

**Return list_of_parameters**

**schema = 'CommandSchema'**

**class** brewtils.models.**Parameter**(*key*, *type=None*, *multi=None*, *display_name=None*, *optional=None*, *default=None*, *description=None*, *choices=None*, *parameters=None*, *nullable=None*, *maximum=None*, *minimum=None*, *regex=None*, *form_input_type=None*)
Bases: object

**FORM_INPUT_TYPES = ('textarea',)**

**TYPES = ('String', 'Integer', 'Float', 'Boolean', 'Any', 'Dictionary', 'Date', 'DateTi**

**is_different**(*other*)

**schema = 'ParameterSchema'**

**class** brewtils.models.**Request**(*system=None*, *system_version=None*, *instance_name=None*, *command=None*, *id=None*, *parent=None*, *children=None*, *parameters=None*, *comment=None*, *output=None*, *output_type=None*, *status=None*, *command_type=None*, *created_at=None*, *error_class=None*, *metadata=None*, *updated_at=None*, *has_parent=None*, *requester=None*)
Bases: *brewtils.models.RequestTemplate*

**COMMAND_TYPES = ('ACTION', 'INFO', 'EPHEMERAL')**

**COMPLETED_STATUSES = ('CANCELED', 'SUCCESS', 'ERROR')**

**OUTPUT_TYPES = ('STRING', 'JSON', 'XML', 'HTML')**

**STATUS_LIST = ('CREATED', 'RECEIVED', 'IN_PROGRESS', 'CANCELED', 'SUCCESS', 'ERROR')**

**is_ephemeral**

**is_json**

**schema = 'RequestSchema'**

**status**

**class** brewtils.models.**PatchOperation**(*operation=None*, *path=None*, *value=None*)
Bases: object

**schema = 'PatchSchema'**

**class** brewtils.models.**Choices**(*type=None*, *display=None*, *value=None*, *strict=None*, *details=None*)

Bases: object

**DISPLAYS = ('select', 'typeahead')**

**TYPES = ('static', 'url', 'command')**

**schema = 'ChoicesSchema'**

**class** brewtils.models.**LoggingConfig**(*level=None*, *handlers=None*, *formatters=None*, *loggers=None*)

Bases: object

**DEFAULT_FORMAT = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'**

**DEFAULT_HANDLER = {'class': 'logging.StreamHandler', 'formatter': 'default', 'stream**

**LEVELS = ('DEBUG', 'INFO', 'WARN', 'ERROR')**

**SUPPORTED_HANDLERS = ('stdout', 'file', 'logstash')**

**formatter_names**

**get_plugin_log_config**(***kwargs*)

Get a specific plugin logging configuration.

It is possible for different systems to have different logging configurations. This method will create the correct plugin logging configuration and return it. If a specific logger is not found for a system, then the current logging configuration will be returned.

**Parameters** **kwargs** – Identifying information for a system (i.e. system_name)

**Returns**

**handler_names**

**schema = 'LoggingConfigSchema'**

**class** brewtils.models.**Event**(*name=None*, *payload=None*, *error=None*, *metadata=None*, *timestamp=None*)

Bases: object

**schema = 'EventSchema'**

**class** brewtils.models.**Events**

Bases: enum.Enum

**ALL_QUEUES_CLEARED = 15**

**BARTENDER_STARTED = 3**

**BARTENDER_STOPPED = 4**

**BREWVIEW_STARTED = 1**

**BREWVIEW_STOPPED = 2**

**INSTANCE_INITIALIZED = 8**

**INSTANCE_STARTED = 9**

**INSTANCE_STOPPED = 10**

**QUEUE_CLEARED = 14**

**REQUEST_COMPLETED = 7**

**REQUEST_CREATED = 5**

**REQUEST_STARTED = 6**

**SYSTEM_CREATED = 11**

**SYSTEM_REMOVED = 13**

**SYSTEM_UPDATED = 12**

**class** brewtils.models.**Queue**(*name=None*, *system=None*, *version=None*, *instance=None*, *system_id=None*, *display=None*, *size=None*)

Bases: object

**schema = 'QueueSchema'**

**class** brewtils.models.**Principal**(*id=None*, *username=None*, *roles=None*, *permissions=None*, *preferences=None*, *metadata=None*)

Bases: object

**schema = 'PrincipalSchema'**

**class** brewtils.models.**Role**(*id=None*, *name=None*, *description=None*, *roles=None*, *permissions=None*)

Bases: object

**schema = 'RoleSchema'**

**class** brewtils.models.**RefreshToken**(*id=None*, *issued=None*, *expires=None*, *payload=None*)

Bases: object

**schema = 'RefreshTokenSchema'**

**class** brewtils.models.**Job**(*id=None*, *name=None*, *trigger_type=None*, *trigger=None*, *request_template=None*, *misfire_grace_time=None*, *coalesce=None*, *next_run_time=None*, *success_count=None*, *error_count=None*, *status=None*, *max_instances=None*)

Bases: object

**STATUS_TYPES = set(['PAUSED', 'RUNNING'])**

**TRIGGER_TYPES = set(['cron', 'date', 'interval'])**

**schema = 'JobSchema'**

**class** brewtils.models.**RequestTemplate**(*system=None*, *system_version=None*, *instance_name=None*, *command=None*, *parameters=None*, *comment=None*, *metadata=None*)

Bases: object

**schema = 'RequestTemplateSchema'**

**class** brewtils.models.**DateTrigger**(*run_date=None*, *timezone=None*)

Bases: object

**schema = 'DateTriggerSchema'**

**class** brewtils.models.**CronTrigger**(*year=None*, *month=None*, *day=None*, *week=None*, *day_of_week=None*, *hour=None*, *minute=None*, *second=None*, *start_date=None*, *end_date=None*, *timezone=None*, *jitter=None*)

Bases: object

**schema = 'CronTriggerSchema'**

**class** brewtils.models.**IntervalTrigger**(*weeks=None*, *days=None*, *hours=None*, *minutes=None*, *seconds=None*, *start_date=None*, *end_date=None*, *timezone=None*, *jitter=None*, *reschedule_on_finish=None*)

Bases: object

**schema = 'IntervalTriggerSchema'**

### 5.1.8 brewtils.plugin module

**class** brewtils.plugin.**Plugin**(*client*, *bg_host=None*, *bg_port=None*, *ssl_enabled=None*, *ca_cert=None*, *client_cert=None*, *system=None*, *name=None*, *description=None*, *version=None*, *icon_name=None*, *instance_name=None*, *logger=None*, *parser=None*, *multithreaded=None*, *metadata=None*, *max_concurrent=None*, *bg_url_prefix=None*, ***kwargs*)

Bases: object

A beer-garden Plugin.

This class represents a beer-garden Plugin - a continuously-running process that can receive and process Requests.

To work, a Plugin needs a Client instance - an instance of a class defining which Requests this plugin can accept and process. The easiest way to define

a Client is by annotating a class with the @system decorator.

When creating a Plugin you can pass certain keyword arguments to let the Plugin know how to communicate with the beer-garden instance. These are:

- bg_host
- bg_port
- ssl_enabled
- ca_cert
- client_cert
- bg_url_prefix

A Plugin also needs some identifying data. You can either pass parameters to the Plugin or pass a fully defined System object (but not both). Note that some fields are optional:

```
Plugin(
    name="Test",
    version="1.0.0",
    instance_name="default",
    description="A Test",
)
```

or:

```
the_system = System(
    name="Test",
    version="1.0.0",
    instance_name="default,
    description="A Test",
```

(continues on next page)

(continued from previous page)

```
)
Plugin(system=the_system)
```

If passing parameters directly note that these fields are required:

**name** Environment variable `BG_NAME` will be used if not specified

**version** Environment variable `BG_VERSION` will be used if not specified

**instance_name** Environment variable `BG_INSTANCE_NAME` will be used if not specified. 'default' will be used if not specified and loading from environment variable was unsuccessful

And these fields are optional:

- description (Will use docstring summary line from Client if unspecified)
- icon_name
- metadata
- display_name

Plugins service requests using a `concurrent.futures.ThreadPoolExecutor`. The maximum number of threads available is controlled by the max_concurrent argument (the 'multithreaded' argument has been deprecated).

**Warning:** The default value for `max_concurrent` is 1. This means that a Plugin that invokes a Command on itself in the course of processing a Request will deadlock! If you intend to do this, please set `max_concurrent` to a value that makes sense and be aware that Requests are processed in separate thread contexts!

**Parameters**

- **client** – Instance of a class annotated with @system.
- **bg_host** (*str*) – Hostname of a beer-garden.
- **bg_port** (*int*) – Port beer-garden is listening on.
- **ssl_enabled** (*bool*) – Whether to use SSL for beer-garden communication.
- **ca_cert** – Certificate that issued the server certificate used by the beer-garden server.
- **client_cert** – Certificate used by the server making the connection to beer-garden.
- **system** – The system definition.
- **name** – The system name.
- **description** – The system description.
- **version** – The system version.
- **icon_name** – The system icon name.
- **instance_name** (*str*) – The name of the instance.
- **logger** (*logging.Logger.*) – A logger that will be used by the Plugin.
- **parser** (*brewtils.schema_parser.SchemaParser.*) – The parser to use when communicating with beer-garden.
- **multithreaded** (*bool*) – DEPRECATED Process requests in a separate thread.

- **worker_shutdown_timeout** (*int*) – Time to wait during shutdown to finish processing.
- **metadata** (*dict*) – Metadata specific to this plugin.
- **max_concurrent** (*int*) – Maximum number of requests to process concurrently.
- **bg_url_prefix** (*str*) – URL Prefix beer-garden is on.
- **display_name** (*str*) – The display name to use for the system.
- **max_attempts** (*int*) – Number of times to attempt updating the request before giving up (default -1 aka never).
- **max_timeout** (*int*) – Maximum amount of time to wait before retrying to update a request.
- **starting_timeout** (*int*) – Initial time to wait before the first retry.
- **mq_max_attempts** (*int*) – Number of times to attempt reconnection to message queue before giving up (default -1 aka never).
- **mq_max_timeout** (*int*) – Maximum amount of time to wait before retrying to connect to message queue.
- **mq_starting_timeout** (*int*) – Initial time to wait before the first message queue connection retry.
- **max_instances** (*int*) – Max number of instances allowed for the system.
- **ca_verify** (*bool*) – Verify server certificate when making a request.
- **username** (*str*) – The username for Beergarden authentication
- **password** (*str*) – The password for Beergarden authentication
- **access_token** – Access token for Beergarden authentication
- **refresh_token** – Refresh token for Beergarden authentication

**process_admin_message**(*message*, *headers*)

**process_message**(*target*, *request*, *headers*)
Process a message. Intended to be run on an Executor.

**Parameters**

- **target** – The object to invoke received commands on. (self or self.client)
- **request** – The parsed Request object
- **headers** – Dictionary of headers from the *brewtils.request_consumer.RequestConsumer*

**Returns** None

**process_request_message**(*message*, *headers*)
Processes a message from a RequestConsumer

**Parameters**

- **message** – A valid string representation of a *brewtils.models.Request*
- **headers** – A dictionary of headers from the *brewtils.request_consumer.RequestConsumer*

**Returns** A *concurrent.futures.Future*

**run**()

brewtils.plugin.**PluginBase**
alias of *brewtils.plugin.Plugin*

**class** brewtils.plugin.**RemotePlugin**(*client*, *bg_host=None*, *bg_port=None*, *ssl_enabled=None*, *ca_cert=None*, *client_cert=None*, *system=None*, *name=None*, *description=None*, *version=None*, *icon_name=None*, *instance_name=None*, *logger=None*, *parser=None*, *multithreaded=None*, *metadata=None*, *max_concurrent=None*, *bg_url_prefix=None*, ***kwargs*)
Bases: *brewtils.plugin.Plugin*

### 5.1.9 brewtils.queues module

**class** brewtils.queues.**PikaClient**(*host='localhost'*, *port=5672*, *user='guest'*, *password='guest'*, *connection_attempts=3*, *heartbeat_interval=3600*, *virtual_host='/'*, *exchange='beer_garden'*, *ssl=None*, *blocked_connection_timeout=None*, ***kwargs*)
Bases: object

Base class for connecting to RabbitMQ using Pika

**Parameters**

- **host** – RabbitMQ host
- **port** – RabbitMQ port
- **user** – RabbitMQ user
- **password** – RabbitMQ password
- **connection_attempts** – Maximum number of retry attempts
- **heartbeat** – Time between RabbitMQ heartbeats
- **heartbeat_interval** – DEPRECATED, use heartbeat
- **virtual_host** – RabbitMQ virtual host
- **exchange** – Default exchange that will be used
- **ssl** – SSL Options
- **blocked_connection_timeout** – If not None, the value is a non-negative timeout, in seconds, for the connection to remain blocked (triggered by Connection.Blocked from broker); if the timeout expires before connection becomes unblocked, the connection will be torn down, triggering the adapter-specific mechanism for informing client app about the closed connection (e.g., on_close_callback or ConnectionClosed exception) with *reason_code* of *InternalCloseReasons.BLOCKED_CONNECTION_TIMEOUT*.

**connection_parameters**(***kwargs*)
Get ConnectionParameters associated with this client

Will construct a ConnectionParameters object using parameters passed at initialization as defaults. Any parameters passed in kwargs will override initialization parameters.

**Parameters** ****kwargs** – Overrides for specific parameters

**Returns** ConnectionParameters object

**Return type** pika.ConnectionParameters

**connection_url**
Connection URL for this client's connection information

**Type** str

### 5.1.10 brewtils.request_consumer module

**class** brewtils.request_consumer.**RequestConsumer**(*amqp_url=None*, *queue_name=None*, *on_message_callback=None*, *panic_event=None*, *logger=None*, *thread_name=None*, ***kwargs*)

Bases: threading.Thread

RabbitMQ message consumer

This consumer is designed to be fault-tolerant - if RabbitMQ closes the connection the consumer will attempt to reopen it. There are limited reasons why the connection may be closed from the broker side and usually indicates permission related issues or socket timeouts.

Unexpected channel closures can indicate a problem with a command that was issued.

**Parameters**

- **amqp_url** – (str) The AMQP url to connect to
- **queue_name** – (str) The name of the queue to connect to
- **on_message_callback** (*func*) – function called to invoke message
- **Must return a Future.** (*processing.*) –
- **panic_event** (*threading.Event*) – Event to be set on a catastrophic failure
- **logger** (*logging.Logger*) – A configured Logger
- **thread_name** (*str*) – Name to use for this thread
- **max_concurrent** – (int) Maximum requests to process concurrently

**finish_message**(*basic_deliver*, *future*)
Finish processing a message

This should be invoked as the final part of message processing. It's responsible for acking / nacking messages back to the broker.

The main complexity here depends on whether the request processing future has an exception:

- If there is no exception it acks the message
- If there is an exception: - If the exception is an instance of DiscardMessageException it nacks the message and does not requeue it
  - If the exception is an instance of RepublishRequestException it will construct an entirely new BlockingConnection, use that to publish a new message, and then ack the original message
  - If the exception is not an instance of either the panic_event is set and the consumer will self-destruct

Also, if there's ever an error acking a message the panic_event is set and the consumer will self-destruct.

**Parameters**

- **basic_deliver** –

- **future** – Completed future

**Returns** None

**is_connected**()
Determine if the underlying connection is open

**Returns** True if the connection exists and is open, False otherwise

**on_channel_closed**(*channel*, **args*)
Channel closed callback

This method is invoked by pika when the channel is closed. Channels are usually closed as a result of something that violates the protocol, such as attempting to re-declare an exchange or queue with different parameters.

This indicates that something has gone wrong, so just close the connection (if it's still open) to reset.

**Parameters**

- **channel** – The channel
- **args** – Tuple of arguments describing why the channel closed pika < 1:

  reply_code: Numeric code indicating close reason reply_text: String describing close reason

  **pika >= 1:** exc: Exception describing close

**Returns** None

**on_channel_open**(*channel*)
Channel open success callback

This will add a close callback (on_channel_closed) the channel and will call start_consuming to begin receiving messages.

**Parameters** **channel** – The opened channel object

**Returns** None

**on_connection_closed**(*connection*, **args*)
Connection closed callback

This method is invoked by pika when the connection to RabbitMQ is closed.

If the connection is closed we terminate its IOLoop to stop the RequestConsumer. In the case of an unexpected connection closure we'll wait 5 seconds before terminating with the expectation that the plugin will attempt to restart the consumer once it's dead.

**Parameters**

- **connection** – The connection
- **args** – Tuple of arguments describing why the connection closed pika < 1:

  reply_code: Numeric code indicating close reason reply_text: String describing close reason

  **pika >= 1:** exc: Exception describing close

**Returns** None

**on_connection_open**(*connection*)
Connection open success callback

This method is called by pika once the connection to RabbitMQ has been established.

The only thing this actually does is call the open_channel method.

**Parameters** **connection** – The connection object

**Returns** None

**on_consumer_cancelled**(*method_frame*)
Consumer cancelled callback

This is only invoked if the consumer is cancelled by the broker. Since that effectively ends the request consuming we close the channel to start the process of terminating the RequestConsumer.

**Parameters** **method_frame** (*pika.frame.Method*) – The Basic.Cancel frame

**Returns** None

**on_message**(*channel*, *basic_deliver*, *properties*, *body*)
Invoked when a message is delivered from the queueing service

Invoked by pika when a message is delivered from RabbitMQ. The channel is passed for your convenience. The basic_deliver object that is passed in carries the exchange, routing key, delivery tag and a redelivered flag for the message. the properties passed in is an instance of BasicProperties with the message properties and the body is the message that was sent.

**Parameters**

- **channel** (*pika.channel.Channel*) – The channel object
- **basic_deliver** (*pika.Spec.Basic.Deliver*) – basic_deliver method
- **properties** (*pika.Spec.BasicProperties*) – Message properties
- **body** (*bytes*) – The message body

**on_message_callback_complete**(*basic_deliver*, *future*)
Invoked when the future returned by _on_message_callback completes.

This method will be invoked from the threadpool context. It's only purpose is to schedule the final processing steps to take place on the connection's ioloop.

**Parameters**

- **basic_deliver** –
- **future** – Completed future

**Returns** None

**open_channel**()
Open a channel

**open_connection**()
Opens a connection to RabbitMQ

This method immediately returns the connection object. However, whether the connection was successful is not know until a callback is invoked (either on_open_callback or on_open_error_callback).

**Returns** The SelectConnection object

**run**()
Run the consumer

Creates a connection to RabbitMQ and starts the IOLoop.

The IOLoop will block and allow the SelectConnection to operate. This means that to stop the RequestConsumer we just need to stop the IOLoop.

**Returns** None

**start_consuming**()
Begin consuming messages

The RabbitMQ prefetch is set to the maximum number of concurrent consumers. This ensures that messages remain in RabbitMQ until a consuming thread is available to process them.

An on_cancel_callback is registered so that the consumer is notified if it is canceled by the broker.

**Returns** None

**stop**()
Cleanly shutdown

It's a good idea to call stop_consuming before this to prevent new messages from being processed during shutdown.

This sets the shutdown_event to let callbacks know that this is an orderly (requested) shutdown. It then schedules a channel close on the IOLoop - the channel's on_close callback will close the connection, and the connection's on_close callback will terminate the IOLoop which will end the RequestConsumer.

**Returns** None

**stop_consuming**()
Stop consuming messages

Sends a Basic.Cancel command to the broker, which causes the broker to stop sending the consumer messages.

**Returns** None

### 5.1.11 brewtils.schema_parser module

**class** brewtils.schema_parser.**BrewmasterSchemaParser**
Bases: *brewtils.schema_parser.SchemaParser*

**class** brewtils.schema_parser.**SchemaParser**
Bases: object

Serialize and deserialize Brewtils models

**logger = <logging.Logger object>**

**classmethod parse_command**(*command*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a command model object

**Parameters**

- **command** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** A Command object

**classmethod parse_event**(*event*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to an event model object

**Parameters**

- **event** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** An Event object

**classmethod parse_instance**(*instance*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to an instance model object

**Parameters**

- **instance** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** An Instance object

**classmethod parse_job**(*job*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a job model object

**Parameters**

- **job** – Raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- ****kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** A Job object.

**classmethod parse_logging_config**(*logging_config*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a logging config model object

Note: for our logging_config, many is _always_ set to False. We will always return a dict from this method.

**Parameters**

- **logging_config** – The raw input
- **from_string** – True if 'input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** A LoggingConfig object

**classmethod parse_parameter**(*parameter*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a parameter model object

**Parameters**

- **parameter** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** An Parameter object

**classmethod parse_patch**(*patch*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a patch model object

Note: for our patches, many is _always_ set to True. We will always return a list from this method.

**Parameters**

- **patch** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** A PatchOperation object

**classmethod parse_principal**(*principal*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a principal model object

**Parameters**

- **principal** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** A Principal object

**classmethod parse_queue**(*queue*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a queue model object

**Parameters**

- **queue** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** A Queue object

**classmethod parse_refresh_token**(*refresh_token*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a refresh token object

**Parameters**

- **refresh_token** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** A RefreshToken object

**classmethod parse_request**(*request*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a request model object

**Parameters**

- **request** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** A Request object

**classmethod parse_role**(*role*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a role model object

**Parameters**

- **role** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** A Role object

**classmethod parse_system**(*system*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a system model object

**Parameters**

- **system** – The raw input
- **from_string** – True if input is a JSON string, False if a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** A System object

**classmethod serialize_command**(*command*, *to_string=True*, ***kwargs*)
Convert a command model into serialized form

**Parameters**

- **command** – The command object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation of command

**classmethod serialize_event**(*event*, *to_string=True*, ***kwargs*)
Convert a logging config model into serialized form

**Parameters**

- **event** – The event object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation of event

**classmethod serialize_instance**(*instance*, *to_string=True*, ***kwargs*)
Convert an instance model into serialized form

**Parameters**

- **instance** – The instance object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation of instance

**classmethod serialize_job**(*job*, *to_string=True*, ***kwargs*)
Convert a job model into serialized form.

**Parameters**

- **job** – The job object(s) to be serialized.
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary.
- ****kwargs** – Additional parameters to be passed to the shcema (e.g. many=True)

**Returns** Serialize representation of job.

**classmethod serialize_logging_config**(*logging_config*, *to_string=True*, ***kwargs*)
Convert a logging config model into serialize form

**Parameters**

- **logging_config** – The logging config object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation of logging config

**classmethod serialize_parameter**(*parameter*, *to_string=True*, ***kwargs*)
Convert a parameter model into serialized form

**Parameters**

- **parameter** – The parameter object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation of parameter

**classmethod serialize_patch**(*patch*, *to_string=True*, ***kwargs*)
Convert a patch model into serialized form

**Parameters**

- **patch** – The patch object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation of patch

**classmethod serialize_principal**(*principal*, *to_string=True*, ***kwargs*)
Convert a principal model into serialized form

**Parameters**

- **principal** – The principal object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation

**classmethod serialize_queue**(*queue*, *to_string=True*, ***kwargs*)
Convert a queue model into serialized form

**Parameters**

- **queue** – The queue object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation of queue

**classmethod serialize_refresh_token**(*refresh_token*, *to_string=True*, ***kwargs*)
Convert a role model into serialized form

**Parameters**

- **refresh_token** – The token object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary

- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation

**classmethod serialize_request**(*request*, *to_string=True*, ***kwargs*)
Convert a request model into serialized form

**Parameters**

- **request** – The request object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation of request

**classmethod serialize_role**(*role*, *to_string=True*, ***kwargs*)
Convert a role model into serialized form

**Parameters**

- **role** – The role object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation

**classmethod serialize_system**(*system*, *to_string=True*, *include_commands=True*, ***kwargs*)
Convert a system model into serialized form

**Parameters**

- **system** – The system object(s) to be serialized
- **to_string** – True to generate a JSON-formatted string, False to generate a dictionary
- **include_commands** – True if the system's command list should be included
- **kwargs** – Additional parameters to be passed to the Schema (e.g. many=True)

**Returns** Serialized representation of system

### 5.1.12 brewtils.schemas module

**class** brewtils.schemas.**SystemSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**InstanceSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**CommandSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**ParameterSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**RequestSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.RequestTemplateSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**PatchSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**unwrap_envelope**(*data*, *many*)

**wrap_envelope**(*data*, *many*)

**class** brewtils.schemas.**LoggingConfigSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**EventSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**QueueSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**PrincipalSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**RoleSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**RefreshTokenSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**JobSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**DateTriggerSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**IntervalTriggerSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

**class** brewtils.schemas.**CronTriggerSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

**opts = <marshmallow.schema.SchemaOpts object>**

### 5.1.13 brewtils.specification module

### 5.1.14 brewtils.stoppable_thread module

**class** brewtils.stoppable_thread.**StoppableThread**(***kwargs*)
Bases: threading.Thread

Thread class with a stop() method. The thread itself has to check regularly for the stopped() condition.

**stop**()
Sets the stop event

**stopped**()
Determines if stop has been called yet.

**wait**(*timeout=None*)
Delegate wait call to threading.Event

### 5.1.15 Module contents

brewtils.**command**(*_wrapped=None*, *command_type='ACTION'*, *output_type='STRING'*, *schema=None*, *form=None*, *template=None*, *icon_name=None*, *description=None*)
Decorator that marks a function as a beer-garden command

For example:

```
@command(output_type='JSON')
def echo_json(self, message):
    return message
```

**Parameters**

- **_wrapped** – The function to decorate. This is handled as a positional argument and shouldn't be explicitly set.
- **command_type** – The command type. Valid options are Command.COMMAND_TYPES.
- **output_type** – The output type. Valid options are Command.OUTPUT_TYPES.
- **schema** – A custom schema definition.
- **form** – A custom form definition.
- **template** – A custom template definition.
- **icon_name** – The icon name. Should be either a FontAwesome or a Glyphicon name.
- **description** – The command description. Will override the function's docstring.

**Returns** The decorated function.

brewtils.**parameter**(*_wrapped=None*, *key=None*, *type=None*, *multi=None*, *display_name=None*, *optional=None*, *default=None*, *description=None*, *choices=None*, *nullable=None*, *maximum=None*, *minimum=None*, *regex=None*, *is_kwarg=None*, *model=None*, *form_input_type=None*)
Decorator that enables Parameter specifications for a beer-garden Command

This decorator is intended to be used when more specification is desired for a Parameter.

For example:

```
@parameter(key="message", description="Message to echo", optional=True, type=
↪"String",
           default="Hello, World!")
def echo(self, message):
    return message
```

**Parameters**

- **_wrapped** – The function to decorate. This is handled as a positional argument and shouldn't be explicitly set.
- **key** – String specifying the parameter identifier. Must match an argument name of the decorated function.
- **type** – String indicating the type to use for this parameter.
- **multi** – Boolean indicating if this parameter is a multi. See documentation for discussion of what this means.
- **display_name** – String that will be displayed as a label in the user interface.
- **optional** – Boolean indicating if this parameter must be specified.
- **default** – The value this parameter will be assigned if not overridden when creating a request.
- **description** – An additional string that will be displayed in the user interface.
- **choices** – List or dictionary specifying allowed values. See documentation for more information.
- **nullable** – Boolean indicating if this parameter is allowed to be null.
- **maximum** – Integer indicating the maximum value of the parameter.
- **minimum** – Integer indicating the minimum value of the parameter.
- **regex** – String describing a regular expression constraint on the parameter.
- **is_kwarg** – Boolean indicating if this parameter is meant to be part of the decorated function's kwargs.
- **model** – Class to be used as a model for this parameter. Must be a Python type object, not an instance.
- **form_input_type** – Only used for string fields. Changes the form input field (e.g. textarea)

**Returns** The decorated function.

brewtils.**system**(*cls=None*, *bg_name=None*, *bg_version=None*)

Class decorator that marks a class as a beer-garden System

**Creates some properties on the class:**

- `_commands`: holds all registered commands
- `_name`: an optional system name
- `_version`: an optional system version
- `_current_request`: Reference to the currently executing request

**Parameters**

- **cls** – The class to decorated
- **bg_name** – Optional plugin name
- **bg_version** – Optional plugin version

**Returns** The decorated class

**class** brewtils.**Plugin**(*client*, *bg_host=None*, *bg_port=None*, *ssl_enabled=None*, *ca_cert=None*, *client_cert=None*, *system=None*, *name=None*, *description=None*, *version=None*, *icon_name=None*, *instance_name=None*, *logger=None*, *parser=None*, *multithreaded=None*, *metadata=None*, *max_concurrent=None*, *bg_url_prefix=None*, ***kwargs*)

Bases: object

A beer-garden Plugin.

This class represents a beer-garden Plugin - a continuously-running process that can receive and process Requests.

To work, a Plugin needs a Client instance - an instance of a class defining which Requests this plugin can accept and process. The easiest way to define

a Client is by annotating a class with the @system decorator.

When creating a Plugin you can pass certain keyword arguments to let the Plugin know how to communicate with the beer-garden instance. These are:

- bg_host
- bg_port
- ssl_enabled
- ca_cert
- client_cert
- bg_url_prefix

A Plugin also needs some identifying data. You can either pass parameters to the Plugin or pass a fully defined System object (but not both). Note that some fields are optional:

```
Plugin(
    name="Test",
    version="1.0.0",
    instance_name="default",
    description="A Test",
)
```

or:

```
the_system = System(
    name="Test",
    version="1.0.0",
    instance_name="default,
    description="A Test",
)
Plugin(system=the_system)
```

If passing parameters directly note that these fields are required:

**name** Environment variable BG_NAME will be used if not specified

**version** Environment variable `BG_VERSION` will be used if not specified

**instance_name** Environment variable `BG_INSTANCE_NAME` will be used if not specified. 'default' will be used if not specified and loading from environment variable was unsuccessful

And these fields are optional:

- description (Will use docstring summary line from Client if unspecified)
- icon_name
- metadata
- display_name

Plugins service requests using a `concurrent.futures.ThreadPoolExecutor`. The maximum number of threads available is controlled by the max_concurrent argument (the 'multithreaded' argument has been deprecated).

**Warning:** The default value for `max_concurrent` is 1. This means that a Plugin that invokes a Command on itself in the course of processing a Request will deadlock! If you intend to do this, please set `max_concurrent` to a value that makes sense and be aware that Requests are processed in separate thread contexts!

**Parameters**

- **client** – Instance of a class annotated with @system.
- **bg_host** (*str*) – Hostname of a beer-garden.
- **bg_port** (*int*) – Port beer-garden is listening on.
- **ssl_enabled** (*bool*) – Whether to use SSL for beer-garden communication.
- **ca_cert** – Certificate that issued the server certificate used by the beer-garden server.
- **client_cert** – Certificate used by the server making the connection to beer-garden.
- **system** – The system definition.
- **name** – The system name.
- **description** – The system description.
- **version** – The system version.
- **icon_name** – The system icon name.
- **instance_name** (*str*) – The name of the instance.
- **logger** (*logging.Logger.*) – A logger that will be used by the Plugin.
- **parser** (*brewtils.schema_parser.SchemaParser.*) – The parser to use when communicating with beer-garden.
- **multithreaded** (*bool*) – DEPRECATED Process requests in a separate thread.
- **worker_shutdown_timeout** (*int*) – Time to wait during shutdown to finish processing.
- **metadata** (*dict*) – Metadata specific to this plugin.
- **max_concurrent** (*int*) – Maximum number of requests to process concurrently.
- **bg_url_prefix** (*str*) – URL Prefix beer-garden is on.

- **display_name** (*str*) – The display name to use for the system.
- **max_attempts** (*int*) – Number of times to attempt updating the request before giving up (default -1 aka never).
- **max_timeout** (*int*) – Maximum amount of time to wait before retrying to update a request.
- **starting_timeout** (*int*) – Initial time to wait before the first retry.
- **mq_max_attempts** (*int*) – Number of times to attempt reconnection to message queue before giving up (default -1 aka never).
- **mq_max_timeout** (*int*) – Maximum amount of time to wait before retrying to connect to message queue.
- **mq_starting_timeout** (*int*) – Initial time to wait before the first message queue connection retry.
- **max_instances** (*int*) – Max number of instances allowed for the system.
- **ca_verify** (*bool*) – Verify server certificate when making a request.
- **username** (*str*) – The username for Beergarden authentication
- **password** (*str*) – The password for Beergarden authentication
- **access_token** – Access token for Beergarden authentication
- **refresh_token** – Refresh token for Beergarden authentication

**process_admin_message**(*message*, *headers*)

**process_message**(*target*, *request*, *headers*)
Process a message. Intended to be run on an Executor.

**Parameters**

- **target** – The object to invoke received commands on. (self or self.client)
- **request** – The parsed Request object
- **headers** – Dictionary of headers from the *brewtils.request_consumer.RequestConsumer*

**Returns** None

**process_request_message**(*message*, *headers*)
Processes a message from a RequestConsumer

**Parameters**

- **message** – A valid string representation of a *brewtils.models.Request*
- **headers** – A dictionary of headers from the *brewtils.request_consumer.RequestConsumer*

**Returns** A *concurrent.futures.Future*

**run**()

**class** brewtils.**RemotePlugin**(*client*, *bg_host=None*, *bg_port=None*, *ssl_enabled=None*, *ca_cert=None*, *client_cert=None*, *system=None*, *name=None*, *description=None*, *version=None*, *icon_name=None*, *instance_name=None*, *logger=None*, *parser=None*, *multithreaded=None*, *metadata=None*, *max_concurrent=None*, *bg_url_prefix=None*, ***kwargs*)
Bases: *brewtils.plugin.Plugin*

**class** brewtils.**EasyClient**(*bg_host=None*, *bg_port=None*, *ssl_enabled=False*, *api_version=None*, *ca_cert=None*, *client_cert=None*, *parser=None*, *logger=None*, *url_prefix=None*, *ca_verify=True*, ***kwargs*)

Bases: object

Client for simplified communication with Beergarden

This class is intended to be a middle ground between the RestClient and SystemClient. It provides a 'cleaner' interface to some common Beergarden operations than is exposed by the lower-level RestClient. On the other hand, the SystemClient is much better for generating Beergarden Requests.

**Keyword Arguments**

- **bg_host** (*str*) – Beergarden hostname
- **bg_port** (*int*) – Beergarden port
- **ssl_enabled** (*Optional[bool]*) – Whether to use SSL (HTTP vs HTTPS)
- **api_version** (*Optional[int]*) – The REST API version
- **ca_cert** (*Optional[str]*) – Path to CA certificate file
- **client_cert** (*Optional[str]*) – Path to client certificate file
- **parser** (*Optional[SchemaParser]*) – Parser to use
- **logger** (*Optional[Logger]*) – Logger to use
- **url_prefix** (*Optional[str]*) – Beergarden REST API prefix
- **ca_verify** (*Optional[bool]*) – Whether to verify the server cert hostname
- **username** (*Optional[str]*) – Username for authentication
- **password** (*Optional[str]*) – Password for authentication
- **access_token** (*Optional[str]*) – Access token for authentication
- **refresh_token** (*Optional[str]*) – Refresh token for authentication
- **client_timeout** (*Optional[float]*) – Max time to wait for a server response

**can_connect**(***kwargs*)

Determine if the Beergarden server is responding.

**Kwargs:** Arguments passed to the underlying Requests method

**Returns** A bool indicating if the connection attempt was successful. Will return False only if a ConnectionError is raised during the attempt. Any other exception will be re-raised.

**Raises** requests.exceptions.RequestException – The connection attempt resulted in an exception that indicates something other than a basic connection error. For example, an error with certificate verification.

**clear_all_queues**()

Cancel and remove all Requests in all queues

**Returns** True if the clear was successful

**Return type** bool

**clear_queue**(*queue_name*)

Cancel and remove all Requests from a message queue

**Parameters** **queue_name** (*str*) – The name of the queue to clear

**Returns** True if the clear was successful

**Return type** bool

**create_job**(*job*)
Create a new Job

**Parameters** **job** (Job) – New Job definition

**Returns** The newly-created Job

**Return type** *Job*

**create_request**(*request*, ***kwargs*)
Create a new Request

**Parameters**

- **request** – New request definition
- **kwargs** – Extra request parameters

**Keyword Arguments**

- **blocking** (*bool*) – Wait for request to complete before returning
- **timeout** (*int*) – Maximum seconds to wait for completion

**Returns** The newly-created Request

**Return type** *Request*

**create_system**(*system*)
Create a new System

**Parameters** **system** (System) – The System to create

**Returns** The newly-created system

**Return type** *System*

**find_jobs**(***kwargs*)
Find Jobs using keyword arguments as search parameters

**Parameters** ****kwargs** – Search parameters

**Returns** List of Jobs matching the search parameters

**Return type** List[*Job*]

**find_requests**(***kwargs*)
Find Requests using keyword arguments as search parameters

**Parameters** ****kwargs** – Search parameters

**Returns** List of Systems matching the search parameters

**Return type** List[*Request*]

**find_systems**(***kwargs*)
Find Systems using keyword arguments as search parameters

**Parameters** ****kwargs** – Search parameters

**Returns** List of Systems matching the search parameters

**Return type** List[*System*]

**find_unique_request**(***kwargs*)
Find a unique request

**Note:** If 'id' is a given keyword argument then all other parameters will be ignored.

**Parameters** ****kwargs** – Search parameters

**Returns** The Request if found, None otherwise

**Return type** *Request*, None

**Raises** FetchError – More than one matching Request was found

**find_unique_system**(***kwargs*)
Find a unique system

**Note:** If 'id' is a given keyword argument then all other parameters will be ignored.

**Parameters** ****kwargs** – Search parameters

**Returns** The System if found, None otherwise

**Return type** *System*, None

**Raises** FetchError – More than one matching System was found

**get_instance**(*instance_id*)
Get an Instance

**Parameters** **instance_id** – The Id

**Returns** The Instance

**get_instance_status**(*instance_id*)
Get an Instance

WARNING: This method currently returns the Instance, not the Instance's status. This behavior will be corrected in 3.0.

To prepare for this change please use get_instance() instead of this method.

**Parameters** **instance_id** – The Id

**Returns** The status

**get_logging_config**(*system_name*)
Get logging configuration for a System

**Parameters** **system_name** (*str*) – The name of the System

**Returns** The configuration object

**Return type** *LoggingConfig*

**get_queues**()
Retrieve all queue information

**Returns** The response

**get_user**(*user_identifier*)
Find a user

**Parameters** **user_identifier** (*str*) – User ID or username

**Returns** The User

**Return type** *Principal*

**get_version**(***kwargs*)
Get Bartender, Brew-view, and API version information

**Parameters** ****kwargs** – Extra parameters

**Returns** Response object with version information in the body

**Return type** dict

**initialize_instance**(*instance_id*)
Start an Instance

**Parameters** **instance_id** (*str*) – The Instance ID

**Returns** The updated Instance

**Return type** *Instance*

**instance_heartbeat**(*instance_id*)
Send an Instance heartbeat

**Parameters** **instance_id** (*str*) – The Instance ID

**Returns** True if the heartbeat was successful

**Return type** bool

**pause_job**(*job_id*)
Pause a Job

**Parameters** **job_id** (*str*) – The Job ID

**Returns** The updated Job

**Return type** *Job*

**publish_event**(**args*, ***kwargs*)
Publish a new event

**Parameters**

- ***args** – If a positional argument is given it's assumed to be an Event and will be used
- ****kwargs** – Will be used to construct a new Event to publish if no Event is given in the positional arguments

**Keyword Arguments** **_publishers** (*Optional[List[str]]*) – List of publisher names. If given the Event will only be published to the specified publishers. Otherwise all publishers known to Beergarden will be used.

**Returns** True if the publish was successful

**Return type** bool

**remove_instance**(*instance_id*)
Remove an Instance

**Parameters** **instance_id** (*str*) – The Instance ID

**Returns** True if the remove was successful

**Return type** bool

**remove_job**(*job_id*)
Remove a unique Job

**Parameters** **job_id** (*str*) – The Job ID

**Returns** True if removal was successful

**Return type** bool

**Raises** DeleteError – Couldn't remove Job

**remove_system**(***kwargs*)
Remove a unique System

**Parameters** ****kwargs** – Search parameters

**Returns** True if removal was successful

**Return type** bool

**Raises** FetchError – Couldn't find a System matching given parameters

**resume_job**(*job_id*)
Resume a Job

**Parameters** **job_id** (*str*) – The Job ID

**Returns** The updated Job

**Return type** *Job*

**update_instance_status**(*instance_id*, *new_status*)
Update an Instance status

**Parameters**

- **instance_id** (*str*) – The Instance ID
- **new_status** (*str*) – The new status

**Returns** The updated Instance

**Return type** *Instance*

**update_request**(*request_id*, *status=None*, *output=None*, *error_class=None*)
Update a Request

**Parameters**

- **request_id** (*str*) – The Request ID
- **status** (*Optional[str]*) – New Request status
- **output** (*Optional[str]*) – New Request output
- **error_class** (*Optional[str]*) – New Request error class

**Returns** The updated response

**Return type** Response

**update_system**(*system_id*, *new_commands=None*, ***kwargs*)
Update a System

**Parameters**

- **system_id** (*str*) – The System ID
- **new_commands** (*Optional[List[Command]]*) – New System commands

**Keyword Arguments**

- **metadata** (*dict*) – New System metadata
- **description** (*str*) – New System description
- **display_name** (*str*) – New System display name
- **icon_name** (*str*) – New System icon name

**Returns** The updated system

**Return type** *System*

**who_am_i**()

Find user using the current set of credentials

**Returns** The User

**Return type** *Principal*

**class** brewtils.**SystemClient**(*bg_host=None*, *bg_port=None*, *system_name=None*, *version_constraint='latest'*, *default_instance='default'*, *always_update=False*, *timeout=None*, *max_delay=30*, *api_version=None*, *ssl_enabled=False*, *ca_cert=None*, *blocking=True*, *max_concurrent=None*, *client_cert=None*, *url_prefix=None*, *ca_verify=True*, *raise_on_error=False*, ***kwargs*)

Bases: object

High-level client for generating requests for a beer-garden System.

**SystemClient creation:** This class is intended to be the main way to create beer-garden requests. Create an instance with beer-garden connection information (optionally including a url_prefix) and a system name:

```
client = SystemClient(host, port, 'example_system', ssl_enabled=True, url_
↪prefix=None)
```

Pass additional keyword arguments for more granularity:

**version_constraint:** Allows specifying a particular system version. Can be a version literal ('1.0.0') or the special value 'latest.' Using 'latest' will allow the the SystemClient to retry a request if it fails due to a missing system (see Creating Requests).

**default_instance:** The instance name to use when creating a request if no other instance name is specified. Since each request must be addressed to a specific instance this is a convenience to prevent needing to specify the 'default' instance for each request.

**always_update:** Always attempt to reload the system definition before making a request. This is useful to ensure Requests are always made against the latest version of the system. If not set the System definition will be loaded once (upon making the first request) and then only reloaded if a Request fails.

**Loading the System:** The System definition is lazily loaded, so nothing happens until the first attempt to send a Request. At that point the SystemClient will query beer-garden to get a system definition that matches the system_name and version_constraint. If no matching system can be found a FetchError will be raised. If always_update was set to True this will happen before making each request, not just the first.

**Making a Request:** The standard way to create and send requests is by calling object attributes:

```
request = client.example_command(param_1='example_param')
```

In the normal case this will block until the request completes. Request completion is determined by periodically polling beer-garden to check the Request status. The time between polling requests starts at

0.5s and doubles each time the request has still not completed, up to max_delay. If a timeout was specified and the Request has not completed within that time a `ConnectionTimeoutError` will be raised.

It is also possible to create the SystemClient in non-blocking mode by specifying blocking=False. In this case the request creation will immediately return a Future and will spawn a separate thread to poll for Request completion. The max_concurrent parameter is used to control the maximum threads available for polling.

```
# Create a SystemClient with blocking=False
client = SystemClient(host, port, 'example_system', ssl_enabled=True,␣
↪blocking=False)

# Create and send 5 requests without waiting for request completion
futures = [client.example_command(param_1=number) for number in range(5)]

# Now wait on all requests to complete
concurrent.futures.wait(futures)
```

If the request creation process fails (e.g. the command failed validation) and version_constraint is 'latest' then the SystemClient will check to see if a different version is available, and if so it will attempt to make the request on that version. This is so users of the SystemClient that don't necessarily care about the target system version don't need to be restarted if the target system is updated.

**Tweaking beer-garden Request Parameters:** There are several parameters that control how beer-garden routes / processes a request. To denote these as intended for beer-garden itself (rather than a parameter to be passed to the Plugin) prepend a leading underscore to the argument name.

Sending to another instance:

```
request = client.example_command(_instance_name='instance_2', param_1=
↪'example_param')
```

Request with a comment:

```
request = client.example_command(_comment='I'm a beer-garden comment!',
                                 param_1='example_param')
```

Without the leading underscore the arguments would be treated the same as param_1 - another parameter to be passed to the plugin.

**Parameters**

- **host** – beer-garden REST API hostname.
- **port** – beer-garden REST API port.
- **system_name** – The name of the system to use.
- **version_constraint** – The system version to use. Can be specific or 'latest'.
- **default_instance** – The instance to use if not specified when creating a request.
- **always_update** – Should check for a newer System version before each request.
- **timeout** – Length of time to wait for a request to complete. 'None' means wait forever.
- **max_delay** – Maximum time to wait between checking the status of a created request.
- **api_version** – beer-garden API version.
- **ssl_enabled** – Flag indicating whether to use HTTPS when communicating with beer-garden.

- **ca_cert** – beer-garden REST API server CA certificate.
- **blocking** – Block after request creation until the request completes.
- **max_concurrent** – Maximum number of concurrent requests allowed.
- **client_cert** – The client certificate to use when making requests.
- **url_prefix** – beer-garden REST API URL Prefix.
- **ca_verify** – Flag indicating whether to verify server certificate when making a request.
- **raise_on_error** – Raises an error if the request ends in an error state.
- **username** – Username for Beergarden authentication
- **password** – Password for Beergarden authentication
- **access_token** – Access token for Beergarden authentication
- **refresh_token** – Refresh token for Beergarden authentication
- **client_timeout** – Max time to will wait for server response

**create_bg_request**(*command_name*, ***kwargs*)

Create a callable that will execute a beer-garden request when called.

Normally you interact with the SystemClient by accessing attributes, but there could be certain cases where you want to create a request without sending it.

Example:

```
client = SystemClient(host, port, 'system', blocking=False)
requests = []

# No arguments
requests.append(client.create_bg_request('command_1'))

# arg_1 will be passed as a parameter
requests.append(client.create_bg_request('command_2', arg_1='Hi!'))

futures = [request() for request in requests]    # Calling creates and sends␣
↪the request
concurrent.futures.wait(futures)                 # Wait for all the futures to␣
↪complete
```

**Parameters**

- **command_name** – The name of the command that will be sent.
- **kwargs** – Additional arguments to pass to send_bg_request.

**Raises** **AttributeError** – The system does not have a command with the given command_name.

**Returns** A partial that will create and execute a beer-garden request when called.

**load_bg_system**()

Query beer-garden for a System definition

This method will make the query to beer-garden for a System matching the name and version constraints specified during SystemClient instance creation.

If this method completes successfully the SystemClient will be ready to create and send Requests.

**Raises** ***FetchError*** – If unable to find a matching System

**Returns** None

**send_bg_request**(***kwargs*)
Actually create a Request and send it to beer-garden

**Note:** This method is intended for advanced use only, mainly cases where you're using the SystemClient without a predefined System. It assumes that everything needed to construct the request is being passed in kwargs. If this doesn't sound like what you want you should check out create_bg_request.

**Parameters** **kwargs** – All necessary request parameters, including beer-garden internal parameters

**Raises** ***ValidationError*** – If the Request creation failed validation on the server

**Returns** If the SystemClient was created with blocking=True a completed request object, otherwise a Future that will return the Request when it completes.

brewtils.**get_easy_client**(***kwargs*)
Easy way to get an EasyClient

The benefit to this method over creating an EasyClient directly is that this method will also search the environment for parameters. Kwargs passed to this method will take priority, however.

**Parameters** ****kwargs** – Options for configuring the EasyClient

**Returns** The configured client

**Return type** *brewtils.rest.easy_client.EasyClient*

brewtils.**get_argument_parser**()
Get an ArgumentParser pre-populated with Brewtils arguments

This is helpful if you're expecting additional command line arguments to a plugin startup script.

This enables doing something like:

```
def main():
    parser = get_argument_parser()
    parser.add_argument('positional_arg')

    parsed_args = parser.parse_args(sys.argv[1:])

    # Now you can use the extra argument
    client = MyClient(parsed_args.positional_arg)

    # But you'll need to be careful when using the 'normal' Brewtils
    # configuration loading methods:

    # Option 1: Tell Brewtils about your customized parser
    connection = get_connection_info(cli_args=sys.argv[1:],
                                     argument_parser=parser)

    # Option 2: Use the parsed CLI as a dictionary
    connection = get_connection_info(**vars(parsed_args))

    # Now specify connection kwargs like normal
```

(continues on next page)

(continued from previous page)

```
    plugin = RemotePlugin(client, name=...
                          **connection)
```

IMPORTANT: Note that in both cases the returned `connection` object **will not** contain your new value. Both options just prevent normal CLI parsing from failing on the unknown argument.

**Returns** ArgumentParser: Argument parser with Brewtils arguments loaded

`brewtils.`**`get_connection_info`**(*cli_args=None*, *argument_parser=None*, ***kwargs*)
Wrapper around `load_config` that returns only connection parameters

**Parameters**

- **`cli_args`** (*`list, optional`*) – List of command line arguments for configuration loading
- **`argument_parser`** (*`ArgumentParser, optional`*) – Argument parser to use when parsing cli_args. Supplying this allows adding additional arguments prior to loading the configuration. This can be useful if your startup script takes additional arguments.
- **`**kwargs`** – Additional configuration overrides

**Returns** dict: Parameters needed to make a connection to Beergarden

`brewtils.`**`load_config`**(*cli_args=None*, *argument_parser=None*, ***kwargs*)
Load configuration using Yapconf

Configuration will be loaded from these sources, with earlier sources having higher priority:

1. `**kwargs` passed to this method
2. `cli_args` passed to this method
3. Environment variables using the `BG_` prefix
4. Default values in the brewtils specification

**Parameters**

- **`cli_args`** (*`list, optional`*) – List of command line arguments for configuration loading
- **`argument_parser`** (*`ArgumentParser, optional`*) – Argument parser to use when parsing cli_args. Supplying this allows adding additional arguments prior to loading the configuration. This can be useful if your startup script takes additional arguments. See get_argument_parser for additional information.
- **`**kwargs`** – Additional configuration overrides

**Returns** The resolved configuration object

**Return type** `box.Box`

`brewtils.`**`get_bg_connection_parameters`**(*cli_args=None*, *argument_parser=None*, ***kwargs*)
Wrapper around `load_config` that returns only connection parameters

**Parameters**

- **`cli_args`** (*`list, optional`*) – List of command line arguments for configuration loading

- **argument_parser** (*ArgumentParser, optional*) – Argument parser to use when parsing cli_args. Supplying this allows adding additional arguments prior to loading the configuration. This can be useful if your startup script takes additional arguments.
- ****kwargs** – Additional configuration overrides

**Returns** dict: Parameters needed to make a connection to Beergarden

brewtils.**configure_logging**(*system_name=None*, ***kwargs*)

Load and enable a logging configuration from Beergarden

NOTE: This method will overwrite the current logging configuration.

**Parameters**

- **system_name** – Name of the system to load
- ****kwargs** – Beergarden connection parameters

**Returns** None

CHAPTER 6

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 6.1 Types of Contributions

### 6.1.1 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.

### 6.1.2 Fix Bugs

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

### 6.1.3 Implement Features

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

### 6.1.4 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.

### 6.1.5 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 :)

## 6.2 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.

## 6.3 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 and make sure that the tests pass for all supported Python versions.

## 6.4 Tips

To run a subset of tests:

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

CHAPTER 7

# Credits

## 7.1 Development Leads

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

## 7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

# Brewtils Changelog

## 8.1 2.4.10

Date: 11/12/19

### 8.1.1 Bug Fixes

- Plugins can now survive a rabbitmq broker restart (beer-garden/#353, beer-garden/#359)

## 8.2 2.4.9

Date: 10/30/19

### 8.2.1 Bug Fixes

- Fixed issue with callbacks in RequestConsumer when using Pika v1 (beer-garden/#328)

## 8.3 2.4.8

Date: 9/5/19

### 8.3.1 New Features

- Better control over how specific error types are logged (beer-garden/#285)

### 8.3.2 Bug Fixes

- Decorators now work with non-JSON resources loaded from a URL (beer-garden/#310)

## 8.4 2.4.7

Date: 6/27/19

### 8.4.1 New Features

- Can now specify a name and version in the `system` decorator (beer-garden/#290)

### 8.4.2 Bug Fixes

- SystemClient now correctly handles versions with suffixes (beer-garden/#283)

### 8.4.3 Other Changes

- Added compatability with Pika v1 (#130)

## 8.5 2.4.6

Date: 4/19/19

### 8.5.1 Bug Fixes

- Using new pika heartbeat instead of heartbeat_interval (#118)
- @parameters now accepts any iterable, not just lists (beer-garden/#237)

### 8.5.2 Other Changes

- Support for new header-style authentication token (#122)
- Added EasyClient.get_instance, deprecated get_instance_status (beer-garden/#231)
- Parameters with is_kwarg on command without **kwargs will raise (beer-garden/#216)

## 8.6 2.4.5

Date: 2/14/19

### 8.6.1 Bug Fixes

- Fixed a warning occuring with newer versions of Marshmallow (#111)

### 8.6.2 Other Changes

- Adding EasyClient to __all__ (beer-garden/#233)

## 8.7 2.4.4

Date: 1/7/19

### 8.7.1 Bug Fixes

- RabbitMQ connections now deal with blocked connections (beer-garden/#203)
- Plugin will use url_prefix kwarg if bg_url_prefix not given (beer-garden/#186)
- Always respecting parameter choices definition changes (beer-garden/#58)

## 8.8 2.4.3

Date: 11/16/18

### 8.8.1 New Features

- Added instance retrieve and delete methods to clients (#91)

### 8.8.2 Bug Fixes

- Logging API now respects all connection parameters (#94)

## 8.9 2.4.2

Date: 10/7/18

### 8.9.1 New Features

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

### 8.9.2 Bug Fixes

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

## 8.10 2.4.1

Date: 09/11/18

### 8.10.1 Other Changes

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

## 8.11 2.4.0

Date: 09/5/18

### 8.11.1 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

### 8.11.2 Other Changes

- Renamed PluginBase to Plugin (old name is aliased)

## 8.12 2.3.7

Date: 07/11/18

### 8.12.1 New Features

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

### 8.12.2 Bug Fixes

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

## 8.13 2.3.6

Date: 06/06/18

### 8.13.1 Other Changes

- Added *has_parent* to request model

## 8.14 2.3.5

Date: 4/17/18

### 8.14.1 Bug Fixes

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

## 8.15 2.3.4

Date: 4/5/18

### 8.15.1 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)

### 8.15.2 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

## 8.16 2.3.3

Date: 3/20/18

### 8.16.1 Bug Fixes

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

## 8.17 2.3.2

Date: 3/7/18

### 8.17.1 Bug Fixes

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

## 8.18 2.3.1

Date: 2/22/18

### 8.18.1 New Features

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

## 8.19 2.3.0

Date: 1/26/18

### 8.19.1 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

### 8.19.2 Bug Fixes

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

### 8.19.3 Other Changes

- Better data integrity by only allowing certain Request status transitions

## 8.20 2.2.1

Date: 1/11/18

### 8.20.1 Bug Fixes

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

## 8.21 2.2.0

Date: 10/23/17

### 8.21.1 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

### 8.21.2 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

### 8.21.3 Other changes

- Added license file

## 8.22 2.1.1

Date: 8/25/17

### 8.22.1 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`

### 8.22.2 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`

### 8.22.3 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

### 8.22.4 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

# Python Module Index

## b


brewtils, 58
brewtils.choices, 30
brewtils.decorators, 30
brewtils.errors, 34
brewtils.models, 40
brewtils.plugin, 44
brewtils.queues, 47
brewtils.request_consumer, 48
brewtils.rest, 29
brewtils.rest.client, 15
brewtils.rest.easy_client, 20
brewtils.rest.system_client, 26
brewtils.schema_parser, 51
brewtils.schemas, 56
brewtils.specification, 58
brewtils.stoppable_thread, 58

# Index


## A

AckAndContinueException, 34
AckAndDieException, 34
ALL_QUEUES_CLEARED (*brewtils.models.Events attribute*), 42
arg_pair() (*brewtils.choices.FunctionTransformer static method*), 30
AuthorizationRequired, 34

## B

BARTENDER_STARTED (*brewtils.models.Events attribute*), 42
BARTENDER_STOPPED (*brewtils.models.Events attribute*), 42
BGConflictError (*in module brewtils.errors*), 34
BGNotFoundError (*in module brewtils.errors*), 34
BGRequestFailedError (*in module brewtils.errors*), 35
BrewmasterConnectionError (*in module brewtils.errors*), 35
BrewmasterDeleteError (*in module brewtils.errors*), 35
BrewmasterEasyClient (*class in brewtils.rest.easy_client*), 20
BrewmasterFetchError (*in module brewtils.errors*), 35
BrewmasterModelError (*in module brewtils.errors*), 35
BrewmasterModelValidationError (*in module brewtils.errors*), 35
BrewmasterRestClient (*class in brewtils.rest.client*), 15
BrewmasterRestClientError (*in module brewtils.errors*), 35
BrewmasterRestError (*in module brewtils.errors*), 35
BrewmasterRestServerError (*in module brewtils.errors*), 35
BrewmasterSaveError (*in module brewtils.errors*), 35
BrewmasterSchemaParser (*class in brewtils.schema_parser*), 51
BrewmasterSystemClient (*class in brewtils.rest.system_client*), 26
BrewmasterTimeoutError (*in module brewtils.errors*), 35
BrewmasterValidationError (*in module brewtils.errors*), 35
brewtils (*module*), 58
brewtils.choices (*module*), 30
brewtils.decorators (*module*), 30
brewtils.errors (*module*), 34
brewtils.log (*module*), 38
brewtils.models (*module*), 40
brewtils.plugin (*module*), 44
brewtils.queues (*module*), 47
brewtils.request_consumer (*module*), 48
brewtils.rest (*module*), 29
brewtils.rest.client (*module*), 15
brewtils.rest.easy_client (*module*), 20
brewtils.rest.system_client (*module*), 26
brewtils.schema_parser (*module*), 51
brewtils.schemas (*module*), 56
brewtils.specification (*module*), 58
brewtils.stoppable_thread (*module*), 58
BrewtilsException, 35
BREWVIEW_STARTED (*brewtils.models.Events attribute*), 42
BREWVIEW_STOPPED (*brewtils.models.Events attribute*), 42

## C

can_connect() (*brewtils.EasyClient method*), 63
can_connect() (*brewtils.rest.easy_client.EasyClient method*), 20
Choices (*class in brewtils.models*), 41
clear_all_queues() (*brewtils.EasyClient method*), 63

clear_all_queues() (*brewtils.rest.easy_client.EasyClient method*), 21
clear_queue() (*brewtils.EasyClient method*), 63
clear_queue() (*brewtils.rest.easy_client.EasyClient method*), 21
Command (*class in brewtils.models*), 40
command() (*in module brewtils*), 58
command() (*in module brewtils.decorators*), 32
command_registrar() (*in module brewtils.decorators*), 32
COMMAND_TYPES (*brewtils.models.Command attribute*), 40
COMMAND_TYPES (*brewtils.models.Request attribute*), 41
CommandSchema (*class in brewtils.schemas*), 56
COMPLETED_STATUSES (*brewtils.models.Request attribute*), 41
configure_logging() (*in module brewtils*), 73
configure_logging() (*in module brewtils.log*), 38
ConflictError, 35
connection_parameters() (*brewtils.queues.PikaClient method*), 47
connection_url (*brewtils.queues.PikaClient attribute*), 47
ConnectionTimeoutError (*in module brewtils.errors*), 35
convert_logging_config() (*in module brewtils.log*), 39
create_bg_request() (*brewtils.rest.system_client.SystemClient method*), 28
create_bg_request() (*brewtils.SystemClient method*), 70
create_job() (*brewtils.EasyClient method*), 64
create_job() (*brewtils.rest.easy_client.EasyClient method*), 21
create_request() (*brewtils.EasyClient method*), 64
create_request() (*brewtils.rest.easy_client.EasyClient method*), 21
create_system() (*brewtils.EasyClient method*), 64
create_system() (*brewtils.rest.easy_client.EasyClient method*), 21
CronTrigger (*class in brewtils.models*), 43
CronTriggerSchema (*class in brewtils.schemas*), 57

## D

DateTrigger (*class in brewtils.models*), 43
DateTriggerSchema (*class in brewtils.schemas*), 57
DEFAULT_FORMAT (*brewtils.models.LoggingConfig attribute*), 42
DEFAULT_HANDLER (*brewtils.models.LoggingConfig attribute*), 42
delete_instance() (*brewtils.rest.client.RestClient method*), 16
delete_job() (*brewtils.rest.client.RestClient method*), 16
delete_queue() (*brewtils.rest.client.RestClient method*), 16
delete_queues() (*brewtils.rest.client.RestClient method*), 16
delete_system() (*brewtils.rest.client.RestClient method*), 16
DeleteError, 35
DiscardMessageException, 35
DISPLAYS (*brewtils.models.Choices attribute*), 42

## E

EasyClient (*class in brewtils*), 63
EasyClient (*class in brewtils.rest.easy_client*), 20
enable_auth() (*in module brewtils.rest.client*), 20
ErrorLogLevelCritical, 35
ErrorLogLevelDebug, 36
ErrorLogLevelError, 36
ErrorLogLevelInfo, 36
ErrorLogLevelWarning, 36
Event (*class in brewtils.models*), 42
Events (*class in brewtils.models*), 42
EventSchema (*class in brewtils.schemas*), 57

## F

FetchError, 36
find_jobs() (*brewtils.EasyClient method*), 64
find_jobs() (*brewtils.rest.easy_client.EasyClient method*), 21
find_requests() (*brewtils.EasyClient method*), 64
find_requests() (*brewtils.rest.easy_client.EasyClient method*), 21
find_systems() (*brewtils.EasyClient method*), 64
find_systems() (*brewtils.rest.easy_client.EasyClient method*), 22
find_unique_request() (*brewtils.EasyClient method*), 64
find_unique_request() (*brewtils.rest.easy_client.EasyClient method*), 22
find_unique_system() (*brewtils.EasyClient method*), 65
find_unique_system() (*brewtils.rest.easy_client.EasyClient method*), 22
finish_message() (*brewtils.request_consumer.RequestConsumer method*), 48
FORM_INPUT_TYPES (*brewtils.models.Parameter attribute*), 41
formatter_names (*brewtils.models.LoggingConfig attribute*), 42

func() (*brewtils.choices.FunctionTransformer static method*), 30
func_args (*brewtils.choices.FunctionTransformer attribute*), 30
FunctionTransformer (*class in brewtils.choices*), 30

## G

get_argument_parser() (*in module brewtils*), 71
get_bg_connection_parameters() (*in module brewtils*), 72
get_command() (*brewtils.rest.client.RestClient method*), 16
get_command_by_name() (*brewtils.models.System method*), 40
get_commands() (*brewtils.rest.client.RestClient method*), 16
get_config() (*brewtils.rest.client.RestClient method*), 16
get_connection_info() (*in module brewtils*), 72
get_easy_client() (*in module brewtils*), 71
get_instance() (*brewtils.EasyClient method*), 65
get_instance() (*brewtils.models.System method*), 40
get_instance() (*brewtils.rest.client.RestClient method*), 17
get_instance() (*brewtils.rest.easy_client.EasyClient method*), 22
get_instance_status() (*brewtils.EasyClient method*), 65
get_instance_status() (*brewtils.rest.easy_client.EasyClient method*), 22
get_job() (*brewtils.rest.client.RestClient method*), 17
get_jobs() (*brewtils.rest.client.RestClient method*), 17
get_logging_config() (*brewtils.EasyClient method*), 65
get_logging_config() (*brewtils.rest.client.RestClient method*), 17
get_logging_config() (*brewtils.rest.easy_client.EasyClient method*), 22
get_logging_config() (*in module brewtils.log*), 39
get_parameter_by_key() (*brewtils.models.Command method*), 41
get_plugin_log_config() (*brewtils.models.LoggingConfig method*), 42
get_python_logging_config() (*in module brewtils.log*), 39
get_queues() (*brewtils.EasyClient method*), 65
get_queues() (*brewtils.rest.client.RestClient method*), 17
get_queues() (*brewtils.rest.easy_client.EasyClient method*), 23
get_request() (*brewtils.rest.client.RestClient method*), 17
get_requests() (*brewtils.rest.client.RestClient method*), 17
get_system() (*brewtils.rest.client.RestClient method*), 17
get_systems() (*brewtils.rest.client.RestClient method*), 17
get_tokens() (*brewtils.rest.client.RestClient method*), 17
get_user() (*brewtils.EasyClient method*), 65
get_user() (*brewtils.rest.client.RestClient method*), 18
get_user() (*brewtils.rest.easy_client.EasyClient method*), 23
get_version() (*brewtils.EasyClient method*), 66
get_version() (*brewtils.rest.client.RestClient method*), 18
get_version() (*brewtils.rest.easy_client.EasyClient method*), 23

## H

handle_response_failure() (*in module brewtils.rest.easy_client*), 25
handler_names (*brewtils.models.LoggingConfig attribute*), 42
has_different_commands() (*brewtils.models.System method*), 40
has_different_parameters() (*brewtils.models.Command method*), 41
has_instance() (*brewtils.models.System method*), 40

## I

initialize_instance() (*brewtils.EasyClient method*), 66
initialize_instance() (*brewtils.rest.easy_client.EasyClient method*), 23
Instance (*class in brewtils.models*), 40
instance_heartbeat() (*brewtils.EasyClient method*), 66
instance_heartbeat() (*brewtils.rest.easy_client.EasyClient method*), 23
INSTANCE_INITIALIZED (*brewtils.models.Events attribute*), 42
instance_names (*brewtils.models.System attribute*), 40

INSTANCE_STARTED (*brewtils.models.Events attribute*), 42
INSTANCE_STATUSES (*brewtils.models.Instance attribute*), 40
INSTANCE_STOPPED (*brewtils.models.Events attribute*), 42
InstanceSchema (*class in brewtils.schemas*), 56
IntervalTrigger (*class in brewtils.models*), 43
IntervalTriggerSchema (*class in brewtils.schemas*), 57
is_connected() (*brewtils.request_consumer.RequestConsumer method*), 49
is_different() (*brewtils.models.Parameter method*), 41
is_ephemeral (*brewtils.models.Request attribute*), 41
is_json (*brewtils.models.Request attribute*), 41

## J

Job (*class in brewtils.models*), 43
JobSchema (*class in brewtils.schemas*), 57
JSON_HEADERS (*brewtils.rest.client.RestClient attribute*), 16

## L

LATEST_VERSION (*brewtils.rest.client.RestClient attribute*), 16
LEVELS (*brewtils.models.LoggingConfig attribute*), 42
load_bg_system() (*brewtils.rest.system_client.SystemClient method*), 29
load_bg_system() (*brewtils.SystemClient method*), 70
load_config() (*in module brewtils*), 72
logger (*brewtils.schema_parser.SchemaParser attribute*), 51
LoggingConfig (*class in brewtils.models*), 42
LoggingConfigSchema (*class in brewtils.schemas*), 57

## M

ModelError, 36
ModelValidationError, 36

## N

NoAckAndDieException, 36
normalize_url_prefix() (*in module brewtils.rest*), 29
NotFoundError, 36

## O

on_channel_closed() (*brewtils.request_consumer.RequestConsumer method*), 49
on_channel_open() (*brewtils.request_consumer.RequestConsumer method*), 49
on_connection_closed() (*brewtils.request_consumer.RequestConsumer method*), 49
on_connection_open() (*brewtils.request_consumer.RequestConsumer method*), 49
on_consumer_cancelled() (*brewtils.request_consumer.RequestConsumer method*), 50
on_message() (*brewtils.request_consumer.RequestConsumer method*), 50
on_message_callback_complete() (*brewtils.request_consumer.RequestConsumer method*), 50
open_channel() (*brewtils.request_consumer.RequestConsumer method*), 50
open_connection() (*brewtils.request_consumer.RequestConsumer method*), 50
opts (*brewtils.schemas.CommandSchema attribute*), 56
opts (*brewtils.schemas.CronTriggerSchema attribute*), 57
opts (*brewtils.schemas.DateTriggerSchema attribute*), 57
opts (*brewtils.schemas.EventSchema attribute*), 57
opts (*brewtils.schemas.InstanceSchema attribute*), 56
opts (*brewtils.schemas.IntervalTriggerSchema attribute*), 57
opts (*brewtils.schemas.JobSchema attribute*), 57
opts (*brewtils.schemas.LoggingConfigSchema attribute*), 57
opts (*brewtils.schemas.ParameterSchema attribute*), 56
opts (*brewtils.schemas.PatchSchema attribute*), 57
opts (*brewtils.schemas.PrincipalSchema attribute*), 57
opts (*brewtils.schemas.QueueSchema attribute*), 57
opts (*brewtils.schemas.RefreshTokenSchema attribute*), 57
opts (*brewtils.schemas.RequestSchema attribute*), 57
opts (*brewtils.schemas.RoleSchema attribute*), 57
opts (*brewtils.schemas.SystemSchema attribute*), 56
OUTPUT_TYPES (*brewtils.models.Command attribute*), 40
OUTPUT_TYPES (*brewtils.models.Request attribute*), 41

## P

Parameter (*class in brewtils.models*), 41
parameter() (*in module brewtils*), 58
parameter() (*in module brewtils.decorators*), 31
parameter_keys() (*brewtils.models.Command method*), 41
parameters() (*in module brewtils.decorators*), 31

ParameterSchema (*class in brewtils.schemas*), 56
parse() (*in module brewtils.choices*), 30
parse_command() (*brewtils.schema_parser.SchemaParser class method*), 51
parse_event() (*brewtils.schema_parser.SchemaParser class method*), 51
parse_exception_as_json() (*in module brewtils.errors*), 38
parse_instance() (*brewtils.schema_parser.SchemaParser class method*), 52
parse_job() (*brewtils.schema_parser.SchemaParser class method*), 52
parse_logging_config() (*brewtils.schema_parser.SchemaParser class method*), 52
parse_parameter() (*brewtils.schema_parser.SchemaParser class method*), 52
parse_patch() (*brewtils.schema_parser.SchemaParser class method*), 52
parse_principal() (*brewtils.schema_parser.SchemaParser class method*), 53
parse_queue() (*brewtils.schema_parser.SchemaParser class method*), 53
parse_refresh_token() (*brewtils.schema_parser.SchemaParser class method*), 53
parse_request() (*brewtils.schema_parser.SchemaParser class method*), 53
parse_role() (*brewtils.schema_parser.SchemaParser class method*), 53
parse_system() (*brewtils.schema_parser.SchemaParser class method*), 54
patch_instance() (*brewtils.rest.client.RestClient method*), 18
patch_job() (*brewtils.rest.client.RestClient method*), 18
patch_request() (*brewtils.rest.client.RestClient method*), 18
patch_system() (*brewtils.rest.client.RestClient method*), 18
PatchOperation (*class in brewtils.models*), 41
PatchSchema (*class in brewtils.schemas*), 57
pause_job() (*brewtils.EasyClient method*), 66
pause_job() (*brewtils.rest.easy_client.EasyClient method*), 23
PikaClient (*class in brewtils.queues*), 47
Plugin (*class in brewtils*), 60
Plugin (*class in brewtils.plugin*), 44
plugin_param() (*in module brewtils.decorators*), 33
PluginBase (*in module brewtils.plugin*), 46
PluginError, 36
PluginParamError, 36
PluginValidationError, 36
post_event() (*brewtils.rest.client.RestClient method*), 18
post_jobs() (*brewtils.rest.client.RestClient method*), 19
post_requests() (*brewtils.rest.client.RestClient method*), 19
post_systems() (*brewtils.rest.client.RestClient method*), 19
Principal (*class in brewtils.models*), 43
PrincipalSchema (*class in brewtils.schemas*), 57
process_admin_message() (*brewtils.Plugin method*), 62
process_admin_message() (*brewtils.plugin.Plugin method*), 46
process_message() (*brewtils.Plugin method*), 62
process_message() (*brewtils.plugin.Plugin method*), 46
process_request_message() (*brewtils.Plugin method*), 62
process_request_message() (*brewtils.plugin.Plugin method*), 46
publish_event() (*brewtils.EasyClient method*), 66
publish_event() (*brewtils.rest.easy_client.EasyClient method*), 23

## Q

Queue (*class in brewtils.models*), 43
QUEUE_CLEARED (*brewtils.models.Events attribute*), 42
QueueSchema (*class in brewtils.schemas*), 57

## R

reference() (*brewtils.choices.FunctionTransformer static method*), 30
refresh() (*brewtils.rest.client.RestClient method*), 19
RefreshToken (*class in brewtils.models*), 43
RefreshTokenSchema (*class in brewtils.schemas*), 57
register() (*in module brewtils.decorators*), 34
RemotePlugin (*class in brewtils*), 62
RemotePlugin (*class in brewtils.plugin*), 47
remove_instance() (*brewtils.EasyClient method*), 66
remove_instance() (*brewtils.rest.easy_client.EasyClient method*), 24
remove_job() (*brewtils.EasyClient method*), 66
remove_job() (*brewtils.rest.easy_client.EasyClient method*), 24
remove_system() (*brewtils.EasyClient method*), 67
remove_system() (*brewtils.rest.easy_client.EasyClient method*), 24
RepublishRequestException, 36
Request (*class in brewtils.models*), 41

REQUEST_COMPLETED (*brewtils.models.Events attribute*), 42
REQUEST_CREATED (*brewtils.models.Events attribute*), 42
REQUEST_STARTED (*brewtils.models.Events attribute*), 42
RequestConsumer (*class in brewtils.request_consumer*), 48
RequestFailedError, 37
RequestForbidden, 37
RequestProcessException, 37
RequestProcessingError, 37
RequestPublishException, 37
RequestSchema (*class in brewtils.schemas*), 56
RequestStatusTransitionError, 37
RequestTemplate (*class in brewtils.models*), 43
RestClient (*class in brewtils.rest.client*), 15
RestClientError, 37
RestConnectionError, 37
RestError, 37
RestServerError, 37
resume_job() (*brewtils.EasyClient method*), 67
resume_job() (*brewtils.rest.easy_client.EasyClient method*), 24
Role (*class in brewtils.models*), 43
RoleSchema (*class in brewtils.schemas*), 57
run() (*brewtils.Plugin method*), 62
run() (*brewtils.plugin.Plugin method*), 46
run() (*brewtils.request_consumer.RequestConsumer method*), 50

## S

SaveError, 37
schema (*brewtils.models.Choices attribute*), 42
schema (*brewtils.models.Command attribute*), 41
schema (*brewtils.models.CronTrigger attribute*), 43
schema (*brewtils.models.DateTrigger attribute*), 43
schema (*brewtils.models.Event attribute*), 42
schema (*brewtils.models.Instance attribute*), 40
schema (*brewtils.models.IntervalTrigger attribute*), 44
schema (*brewtils.models.Job attribute*), 43
schema (*brewtils.models.LoggingConfig attribute*), 42
schema (*brewtils.models.Parameter attribute*), 41
schema (*brewtils.models.PatchOperation attribute*), 41
schema (*brewtils.models.Principal attribute*), 43
schema (*brewtils.models.Queue attribute*), 43
schema (*brewtils.models.RefreshToken attribute*), 43
schema (*brewtils.models.Request attribute*), 41
schema (*brewtils.models.RequestTemplate attribute*), 43
schema (*brewtils.models.Role attribute*), 43
schema (*brewtils.models.System attribute*), 40
SchemaParser (*class in brewtils.schema_parser*), 51
send() (*brewtils.rest.client.TimeoutAdapter method*), 19
send_bg_request() (*brewtils.rest.system_client.SystemClient method*), 29
send_bg_request() (*brewtils.SystemClient method*), 71
serialize_command() (*brewtils.schema_parser.SchemaParser class method*), 54
serialize_event() (*brewtils.schema_parser.SchemaParser class method*), 54
serialize_instance() (*brewtils.schema_parser.SchemaParser class method*), 54
serialize_job() (*brewtils.schema_parser.SchemaParser class method*), 54
serialize_logging_config() (*brewtils.schema_parser.SchemaParser class method*), 54
serialize_parameter() (*brewtils.schema_parser.SchemaParser class method*), 55
serialize_patch() (*brewtils.schema_parser.SchemaParser class method*), 55
serialize_principal() (*brewtils.schema_parser.SchemaParser class method*), 55
serialize_queue() (*brewtils.schema_parser.SchemaParser class method*), 55
serialize_refresh_token() (*brewtils.schema_parser.SchemaParser class method*), 55
serialize_request() (*brewtils.schema_parser.SchemaParser class method*), 56
serialize_role() (*brewtils.schema_parser.SchemaParser class method*), 56
serialize_system() (*brewtils.schema_parser.SchemaParser class method*), 56
setup_logger() (*in module brewtils.log*), 39
start_consuming() (*brewtils.request_consumer.RequestConsumer method*), 51
status (*brewtils.models.Request attribute*), 41
STATUS_LIST (*brewtils.models.Request attribute*), 41
STATUS_TYPES (*brewtils.models.Job attribute*), 43
stop() (*brewtils.request_consumer.RequestConsumer method*), 51
stop() (*brewtils.stoppable_thread.StoppableThread method*), 58
stop_consuming() (*brewtils.request_consumer.RequestConsumer*

*method*), 51
StoppableThread (*class in brewtils.stoppable_thread*), 58
stopped() (*brewtils.stoppable_thread.StoppableThread method*), 58
SUPPORTED_HANDLERS (*brewtils.models.LoggingConfig attribute*), 42
SuppressStacktrace, 37
System (*class in brewtils.models*), 40
system() (*in module brewtils*), 59
system() (*in module brewtils.decorators*), 30
SYSTEM_CREATED (*brewtils.models.Events attribute*), 43
SYSTEM_REMOVED (*brewtils.models.Events attribute*), 43
SYSTEM_UPDATED (*brewtils.models.Events attribute*), 43
SystemClient (*class in brewtils*), 68
SystemClient (*class in brewtils.rest.system_client*), 26
SystemSchema (*class in brewtils.schemas*), 56

## T

TimeoutAdapter (*class in brewtils.rest.client*), 19
TimeoutExceededError, 37
TRIGGER_TYPES (*brewtils.models.Job attribute*), 43
TYPES (*brewtils.models.Choices attribute*), 42
TYPES (*brewtils.models.Parameter attribute*), 41

## U

unwrap_envelope() (*brewtils.schemas.PatchSchema method*), 57
update_instance_status() (*brewtils.EasyClient method*), 67
update_instance_status() (*brewtils.rest.easy_client.EasyClient method*), 24
update_request() (*brewtils.EasyClient method*), 67
update_request() (*brewtils.rest.easy_client.EasyClient method*), 24
update_system() (*brewtils.EasyClient method*), 67
update_system() (*brewtils.rest.easy_client.EasyClient method*), 25
url() (*brewtils.choices.FunctionTransformer static method*), 30
url_args (*brewtils.choices.FunctionTransformer attribute*), 30

## V

ValidationError, 37

## W

wait() (*brewtils.stoppable_thread.StoppableThread method*), 58
WaitExceededError (*in module brewtils.errors*), 38
who_am_i() (*brewtils.EasyClient method*), 68
who_am_i() (*brewtils.rest.easy_client.EasyClient method*), 25
wrap_envelope() (*brewtils.schemas.PatchSchema method*), 57
wrap_response() (*in module brewtils.rest.easy_client*), 25