
Brewtils Documentation

Release 2.4.3

Logan Asher Jones

Nov 16, 2018

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	63
6.1	Types of Contributions	63
6.2	Get Started!	64
6.3	Pull Request Guidelines	65
6.4	Tips	65
7	Credits	67
7.1	Development Leads	67
7.2	Contributors	67
8	Brewtils Changelog	69
8.1	2.4.3	69
8.2	2.4.2	69
8.3	2.4.1	70
8.4	2.4.0	70
8.5	2.3.7	70

8.6	2.3.6	71
8.7	2.3.5	71
8.8	2.3.4	71
8.9	2.3.3	71
8.10	2.3.2	72
8.11	2.3.1	72
8.12	2.3.0	72
8.13	2.2.1	73
8.14	2.2.0	73
8.15	2.1.1	73
Python Module Index		75

Contents:

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

1.1 Features

Brewtils helps you interact with beer-garden.

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

1.2 Installation

To install brewtils, run this command in your terminal:

```
$ pip install brewtils
```

Or add it to your `requirements.txt`

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

1.3 Quick Start

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

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

@system
class HelloWorld(object):

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

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

    plugin.run()
```

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

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

```
from brewtils.rest.system_client import SystemClient

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

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

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

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

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

```
from brewtils.rest.easy_client import EasyClient

client = EasyClient('localhost', 2337)

systems = client.find_systems()

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

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

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

1.4 Documentation

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

2.1 Stable release

To install Brewtils, run this command in your terminal:

```
$ pip install brewtils
```

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

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

2.2 From sources

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

You can either clone the public repository:

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

Or download the [tarball](#):

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

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

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


CHAPTER 3

Usage

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

4.1 Features

Brewtils helps you interact with beer-garden.

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

4.2 Installation

To install brewtils, run this command in your terminal:

```
$ pip install brewtils
```

Or add it to your `requirements.txt`

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

4.3 Quick Start

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

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

@system
class HelloWorld(object):

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

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

    plugin.run()
```

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

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

```
from brewtils.rest.system_client import SystemClient

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

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

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

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

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

```
from brewtils.rest.easy_client import EasyClient

client = EasyClient('localhost', 2337)

systems = client.find_systems()

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

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

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

4.4 Documentation

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

5.1 brewtils package

5.1.1 Subpackages

brewtils.log package

Module contents

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

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.

This 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 communicating with beer-garden

This class provides nice wrappers around the functionality provided by a `brewtils.rest.client.RestClient`

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.
- **ca_cert** – beer-garden REST API server CA certificate.
- **client_cert** – The client certificate to use when making requests.
- **parser** – The parser to use. If None will default to an instance of SchemaParser.
- **logger** – The logger to use. If None one will be created.
- **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

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 clear all messages from all queues

Returns The response

clear_queue (*queue_name*)

Cancel and clear all messages from a queue

Returns The response

create_job (*job*)

Create a new job by POSTing

Parameters **job** – The job to create

Returns The job creation response.

create_request (*request*, ****kwargs**)

Create a new request by POSTing

Parameters

- **request** – 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

create_system (*system*)

Create a new system by POSTing

Parameters **system** – The system to create

Returns The system creation response

find_jobs (****kwargs**)

Find jobs using keyword arguments as search parameters

Parameters ****kwargs** – Search parameters

Returns List of jobs.

find_requests (****kwargs**)

Find requests using keyword arguments as search parameters

Parameters **kwargs** – Search parameters

Returns A list of request instances satisfying the given search parameters

find_systems (****kwargs**)

Find systems using keyword arguments as search parameters

Parameters **kwargs** – Search parameters

Returns A list of system instances satisfying the given search parameters

find_unique_request (****kwargs**)

Find a unique request using keyword arguments as search parameters

Note: If 'id' is present in kwargs then all other parameters will be ignored.

Parameters **kwargs** – Search parameters

Returns One request instance

find_unique_system (****kwargs**)

Find a unique system using keyword arguments as search parameters

Parameters **kwargs** – Search parameters

Returns One system instance

get_instance_status (*instance_id*)

Get an instance's status

Parameters **instance_id** – The Id

Returns The status

get_logging_config (*system_name*)

Get the logging configuration for a particular system.

Parameters *system_name* – Name of system

Returns LoggingConfig object

get_queues ()

Retrieve all queue information

Returns The response

get_user (*user_identifier*)

Find a specific user using username or ID

Parameters *user_identifier* – ID or username of User

Returns A User

get_version (***kwargs*)

initialize_instance (*instance_id*)

Start an instance by PATCHing

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

Returns The start response

instance_heartbeat (*instance_id*)

Send heartbeat for health and status

Parameters *instance_id* – The ID of the instance

Returns The response

pause_job (*job_id*)

Pause a Job by ID.

Parameters *job_id* – The ID of the job to pause.

Returns A copy of the job.

publish_event (**args, **kwargs*)

Publish a new event by POSTing

Parameters

- **args** – The Event to create
- **_publishers** – Optional list of specific publishers. If None all publishers will be used.
- **kwargs** – If no Event is given in the **args*, on will be constructed from the *kwargs*

Returns The response

remove_instance (*instance_id*)

Remove an instance

Parameters *instance_id* – The ID of the instance

Returns The response

remove_job (*job_id*)

Remove a job by ID.

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

Returns True if successful, raises an error otherwise.

remove_system (***kwargs*)

Remove a specific system by DELETEing, using keyword arguments as search parameters

Parameters **kwargs** – Search parameters

Returns The response

resume_job (*job_id*)

Resume a job by ID.

Parameters **job_id** – The ID of the job to resume.

Returns A copy of the job.

update_instance_status (*instance_id, new_status*)

Update an instance by PATCHing

Parameters

- **instance_id** – The ID of the instance to start
- **new_status** – The updated status

Returns The start response

update_request (*request_id, status=None, output=None, error_class=None*)

Set various fields on a request by PATCHing

Parameters

- **request_id** – The ID of the request to update
- **status** – The new status
- **output** – The new output
- **error_class** – The new error class

Returns The response

update_system (*system_id, new_commands=None, **kwargs*)

Update a system by PATCHing

Parameters

- **system_id** – The ID of the system to update
- **new_commands** – The new commands

Keyword Arguments

- **metadata** (*dict*) The updated metadata for the system
- **description** (*str*) The updated description for the system
- **display_name** (*str*) The updated display_name for the system
- **icon_name** (*str*) The updated icon_name for the system

Returns The response

who_am_i ()

Find the user represented by the current set of credentials

Returns The current user

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,        ver-
                                                sion_constraint='latest',  de-
                                                fault_instance='default', al-
                                                ways_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.

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 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)`

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

Creates a `_commands` property on the class that holds all registered commands.

Parameters `cls` – The class to decorated

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, ver-
                      sion=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:

```
PluginBase(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")
PluginBase(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 not specified)
- `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.
- **max_instances** (*int*) – Maximum 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, in-
                             stance_name=None, logger=None, parser=None, multi-
                             threaded=None, metadata=None, max_concurrent=None,
                             bg_url_prefix=None, **kwargs)
```

Bases: *brewtils.plugin.Plugin*

```
class brewtils.SystemClient(bg_host=None, bg_port=None, system_name=None, ver-
                             sion_constraint='latest', default_instance='default', al-
                             ways_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
    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

5.1.3 Submodules

5.1.4 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*

5.1.5 brewtils.decorators module

`brewtils.decorators.system(cls)`

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

Creates a `_commands` property on the class that holds all registered commands.

Parameters `cls` – The class to decorated

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 (list)* – 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)

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

Creates a `_commands` property on the class that holds all registered commands.

Parameters **cls** – The class to decorated

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', out-  
                             put_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.

5.1.6 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 queued

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.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.

5.1.7 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(['UNKNOWN', 'DEAD', 'PAUSED', 'RUNNING', 'STOPPED', 'INITIALIZING'])
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', 'DateTime')
    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': 'stdout'}
    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)
    Bases: object

class brewtils.models.Role(id=None, name=None, description=None, roles=None, permissions=None)
    Bases: object

class brewtils.models.RefreshToken(id=None, issued=None, expires=None, payload=None)
    Bases: object

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)
    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)
```

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:

```
PluginBase(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")
PluginBase(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 not specified)
- `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.
- **max_instances** (*int*) – Maximum 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.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 connection with
- **queue_name** (*str*) – The name of the queue to connect to
- **on_message_callback** (*func*) – The function called to invoke message processing. Must return a Future.
- **panic_event** (*event*) – An event to be set in the event of a catastrophic failure
- **logger** (*logging.Logger*) – A configured logger
- **thread_name** (*str*) – The name to use for this thread
- **max_connect_retries** (*int*) – Number of connection retry attempts before failure. Default is -1 (retry forever).
- **max_connect_backoff** (*int*) – Maximum amount of time to wait between connection retry attempts. Default 30.
- **max_concurrent** (*int*) – Maximum requests to process concurrently

close_channel()

Cleanly close the channel

close_connection()

This method closes the connection to RabbitMQ

on_cancelok (*unused_frame*)

Invoked when RabbitMq acknowledges consumer cancellation

This method is invoked when RabbitMQ acknowledges the cancellation of a consumer. It is unused except for logging purposes.

Parameters **unused_frame** (*pika.frame.Method*) – The Basic.CancelOK frame

on_channel_closed (*channel, reply_code, reply_text*)

Invoked when the connection is closed

Invoked by pika when RabbitMQ unexpectedly closes the channel. 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 close the connection to reset.

Parameters

- **channel** (*pika.channel.Channel*) – The closed channel

- **reply_code** (*int*) – The numeric reason the channel was closed
- **reply_text** (*str*) – The text reason the channel was closed

on_channel_open (*channel*)

Invoked when the channel has been opened

Immediately start consuming since the queue bindings are not the consumer's responsibility.

Parameters **channel** (*pika.channel.Channel*) – The channel object

on_connection_closed (*connection, reply_code, reply_text*)

Invoked when the connection is closed

This method is invoked by pika when the connection to RabbitMQ is closed unexpectedly. This method will attempt to reconnect.

Parameters

- **connection** (*pika.connection.Connection*) – the closed connection
- **reply_code** (*int*) – The server provided reply_code if given
- **reply_text** (*basestring*) – The server provided reply_text if given

on_connection_open (*unused_connection*)

Invoked when the connection has been established

This method is called by pika once the connection to RabbitMQ has been established. It passes the handle to the connection object in case we need it, but in this case, we'll just mark it unused.

on_consumer_cancelled (*method_frame*)

Invoked when the consumer is canceled by the broker

This method will simply close the channel if it exists.

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

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*) – properties
- **body** (*bytes*) – The message body

on_message_callback_complete (*basic_deliver, future*)

Invoked when the future returned by `_on_message_callback` completes.

Parameters

- **basic_deliver** (*pika.Spec.Basic.Deliver*) – basic_deliver method
- **future** (*concurrent.futures.Future*) – Completed future

Returns None

open_channel()

Open a channel using the connection

When RabbitMQ responds that the channel is open, the `on_channel_open` callback will be invoked.

open_connection()

Opens a connection to RabbitMQ

This method connects to RabbitMQ, returning the connection handle. The `on_connection_open` method will be invoked when the connection opens.

Return type `pika.SelectConnection`

reconnect()

Will be invoked by the IOLoop timer if the connection is closed

run()

Run the consumer

Creates a connection to RabbitMQ and starts the IOLoop. The IOLoop will block and allow the Select-Connection to operate.

Returns**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.

stop()

Cleanly shutdown the connection

Assumes the `stop_consuming` method has already been called. When the queueing service acknowledges the closure, the connection is closed which will end the RequestConsumer.

Returns**stop_consuming()**

Stop consuming messages

5.1.10 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 schema (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

5.1.11 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>
```

5.1.12 brewtils.specification module

5.1.13 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
```

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
```


7.1 Development Leads

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

7.2 Contributors

None yet. Why not be the first?

8.1 2.4.3

Date: 11/16/18

8.1.1 New Features

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

8.1.2 Bug Fixes

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

8.2 2.4.2

Date: 10/7/18

8.2.1 New Features

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

8.2.2 Bug Fixes

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

8.3 2.4.1

Date: 09/11/18

8.3.1 Other Changes

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

8.4 2.4.0

Date: 09/5/18

8.4.1 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

8.4.2 Other Changes

- Renamed PluginBase to Plugin (old name is aliased)

8.5 2.3.7

Date: 07/11/18

8.5.1 New Features

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

8.5.2 Bug Fixes

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

8.6 2.3.6

Date: 06/06/18

8.6.1 Other Changes

- Added *has_parent* to request model

8.7 2.3.5

Date: 4/17/18

8.7.1 Bug Fixes

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

8.8 2.3.4

Date: 4/5/18

8.8.1 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)

8.8.2 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

8.9 2.3.3

Date: 3/20/18

8.9.1 Bug Fixes

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

8.10 2.3.2

Date: 3/7/18

8.10.1 Bug Fixes

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

8.11 2.3.1

Date: 2/22/18

8.11.1 New Features

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

8.12 2.3.0

Date: 1/26/18

8.12.1 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

8.12.2 Bug Fixes

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

8.12.3 Other Changes

- Better data integrity by only allowing certain Request status transitions

8.13 2.2.1

Date: 1/11/18

8.13.1 Bug Fixes

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

8.14 2.2.0

Date: 10/23/17

8.14.1 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

8.14.2 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

8.14.3 Other changes

- Added license file

8.15 2.1.1

Date: 8/25/17

8.15.1 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

8.15.2 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`

8.15.3 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

8.15.4 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

b

- `brewtils`, [29](#)
- `brewtils.choices`, [38](#)
- `brewtils.decorators`, [39](#)
- `brewtils.errors`, [43](#)
- `brewtils.log`, [15](#)
- `brewtils.models`, [46](#)
- `brewtils.plugin`, [50](#)
- `brewtils.request_consumer`, [53](#)
- `brewtils.rest`, [29](#)
 - `brewtils.rest.client`, [17](#)
 - `brewtils.rest.easy_client`, [21](#)
 - `brewtils.rest.system_client`, [26](#)
- `brewtils.schema_parser`, [55](#)
- `brewtils.schemas`, [60](#)
- `brewtils.specification`, [62](#)
- `brewtils.stoppable_thread`, [62](#)

A

AckAndContinueException, 43
 AckAndDieException, 43
 ALL_QUEUES_CLEARED (brewtils.models.Events attribute), 48
 arg_pair() (brewtils.choices.FunctionTransformer static method), 38
 AuthorizationRequired, 43

B

BARTENDER_STARTED (brewtils.models.Events attribute), 48
 BARTENDER_STOPPED (brewtils.models.Events attribute), 48
 BGConflictError (in module brewtils.errors), 43
 BGNotFoundError (in module brewtils.errors), 43
 BGRequestFailedError (in module brewtils.errors), 43
 BrewmasterConnectionError (in module brewtils.errors), 43
 BrewmasterDeleteError (in module brewtils.errors), 43
 BrewmasterEasyClient (class in brewtils.rest.easy_client), 21
 BrewmasterFetchError (in module brewtils.errors), 43
 BrewmasterModelError (in module brewtils.errors), 43
 BrewmasterModelValidationError (in module brewtils.errors), 43
 BrewmasterRestClient (class in brewtils.rest.client), 17
 BrewmasterRestClientError (in module brewtils.errors), 43
 BrewmasterRestError (in module brewtils.errors), 43
 BrewmasterRestServerError (in module brewtils.errors), 43
 BrewmasterSaveError (in module brewtils.errors), 43
 BrewmasterSchemaParser (class in brewtils.schema_parser), 55
 BrewmasterSystemClient (class in brewtils.rest.system_client), 26
 BrewmasterTimeoutError (in module brewtils.errors), 43
 BrewmasterValidationError (in module brewtils.errors),

43

brewtils (module), 29
 brewtils.choices (module), 38
 brewtils.decorators (module), 39
 brewtils.errors (module), 43
 brewtils.log (module), 15
 brewtils.models (module), 46
 brewtils.plugin (module), 50
 brewtils.request_consumer (module), 53
 brewtils.rest (module), 29
 brewtils.rest.client (module), 17
 brewtils.rest.easy_client (module), 21
 brewtils.rest.system_client (module), 26
 brewtils.schema_parser (module), 55
 brewtils.schemas (module), 60
 brewtils.specification (module), 62
 brewtils.stoppable_thread (module), 62
 BrewtilsException, 43
 BREWVIEW_STARTED (brewtils.models.Events attribute), 48
 BREWVIEW_STOPPED (brewtils.models.Events attribute), 48

C

can_connect() (brewtils.rest.easy_client.EasyClient method), 22
 Choices (class in brewtils.models), 48
 clear_all_queues() (brewtils.rest.easy_client.EasyClient method), 22
 clear_queue() (brewtils.rest.easy_client.EasyClient method), 22
 close_channel() (brewtils.request_consumer.RequestConsumer method), 53
 close_connection() (brewtils.request_consumer.RequestConsumer method), 53
 Command (class in brewtils.models), 47
 command() (in module brewtils), 29
 command() (in module brewtils.decorators), 40
 command_registrar() (in module brewtils.decorators), 41

COMMAND_TYPES (brewtils.models.Command attribute), 47

COMMAND_TYPES (brewtils.models.Request attribute), 47

CommandSchema (class in brewtils.schemas), 60

COMPLETED_STATUSES (brewtils.models.Request attribute), 47

configure_logging() (in module brewtils), 38

configure_logging() (in module brewtils.log), 15

ConflictError, 43

ConnectionTimeoutError (in module brewtils.errors), 43

convert_logging_config() (in module brewtils.log), 16

create_bg_request() (brewtils.rest.system_client.SystemClient method), 28

create_bg_request() (brewtils.SystemClient method), 35

create_job() (brewtils.rest.easy_client.EasyClient method), 22

create_request() (brewtils.rest.easy_client.EasyClient method), 22

create_system() (brewtils.rest.easy_client.EasyClient method), 23

CronTrigger (class in brewtils.models), 49

CronTriggerSchema (class in brewtils.schemas), 62

D

DateTrigger (class in brewtils.models), 49

DateTriggerSchema (class in brewtils.schemas), 61

DEFAULT_FORMAT (brewtils.models.LoggingConfig attribute), 48

DEFAULT_HANDLER (brewtils.models.LoggingConfig attribute), 48

delete_instance() (brewtils.rest.client.RestClient method), 18

delete_job() (brewtils.rest.client.RestClient method), 18

delete_queue() (brewtils.rest.client.RestClient method), 18

delete_queues() (brewtils.rest.client.RestClient method), 18

delete_system() (brewtils.rest.client.RestClient method), 18

DeleteError, 44

DiscardMessageException, 44

DISPLAYS (brewtils.models.Choices attribute), 48

E

EasyClient (class in brewtils.rest.easy_client), 21

enable_auth() (in module brewtils.rest.client), 21

Event (class in brewtils.models), 48

Events (class in brewtils.models), 48

EventSchema (class in brewtils.schemas), 61

F

FetchError, 44

find_jobs() (brewtils.rest.easy_client.EasyClient method), 23

find_requests() (brewtils.rest.easy_client.EasyClient method), 23

find_systems() (brewtils.rest.easy_client.EasyClient method), 23

find_unique_request() (brewtils.rest.easy_client.EasyClient method), 23

find_unique_system() (brewtils.rest.easy_client.EasyClient method), 23

FORM_INPUT_TYPES (brewtils.models.Parameter attribute), 47

formatter_names (brewtils.models.LoggingConfig attribute), 48

func() (brewtils.choices.FunctionTransformer static method), 38

func_args (brewtils.choices.FunctionTransformer attribute), 38

FunctionTransformer (class in brewtils.choices), 38

G

get_argument_parser() (in module brewtils), 37

get_bg_connection_parameters() (in module brewtils), 38

get_command() (brewtils.rest.client.RestClient method), 18

get_command_by_name() (brewtils.models.System method), 46

get_commands() (brewtils.rest.client.RestClient method), 18

get_config() (brewtils.rest.client.RestClient method), 18

get_connection_info() (in module brewtils), 37

get_easy_client() (in module brewtils), 36

get_instance() (brewtils.models.System method), 46

get_instance() (brewtils.rest.client.RestClient method), 18

get_instance_status() (brewtils.rest.easy_client.EasyClient method), 23

get_job() (brewtils.rest.client.RestClient method), 18

get_jobs() (brewtils.rest.client.RestClient method), 18

get_logging_config() (brewtils.rest.client.RestClient method), 18

get_logging_config() (brewtils.rest.easy_client.EasyClient method), 24

get_logging_config() (in module brewtils.log), 16

get_parameter_by_key() (brewtils.models.Command method), 47

get_plugin_log_config() (brewtils.models.LoggingConfig method), 48

get_python_logging_config() (in module brewtils.log), 16

get_queues() (brewtils.rest.client.RestClient method), 19

get_queues() (brewtils.rest.easy_client.EasyClient method), 24

get_request() (brewtils.rest.client.RestClient method), 19

[get_requests\(\)](#) (brewtils.rest.client.RestClient method), [19](#)
[get_system\(\)](#) (brewtils.rest.client.RestClient method), [19](#)
[get_systems\(\)](#) (brewtils.rest.client.RestClient method), [19](#)
[get_tokens\(\)](#) (brewtils.rest.client.RestClient method), [19](#)
[get_user\(\)](#) (brewtils.rest.client.RestClient method), [19](#)
[get_user\(\)](#) (brewtils.rest.easy_client.EasyClient method), [24](#)
[get_version\(\)](#) (brewtils.rest.client.RestClient method), [19](#)
[get_version\(\)](#) (brewtils.rest.easy_client.EasyClient method), [24](#)

H

[handler_names](#) (brewtils.models.LoggingConfig attribute), [48](#)
[has_different_commands\(\)](#) (brewtils.models.System method), [46](#)
[has_different_parameters\(\)](#) (brewtils.models.Command method), [47](#)
[has_instance\(\)](#) (brewtils.models.System method), [46](#)

I

[initialize_instance\(\)](#) (brewtils.rest.easy_client.EasyClient method), [24](#)
[Instance](#) (class in brewtils.models), [46](#)
[instance_heartbeat\(\)](#) (brewtils.rest.easy_client.EasyClient method), [24](#)
[INSTANCE_INITIALIZED](#) (brewtils.models.Events attribute), [49](#)
[instance_names](#) (brewtils.models.System attribute), [46](#)
[INSTANCE_STARTED](#) (brewtils.models.Events attribute), [49](#)
[INSTANCE_STATUSES](#) (brewtils.models.Instance attribute), [46](#)
[INSTANCE_STOPPED](#) (brewtils.models.Events attribute), [49](#)
[InstanceSchema](#) (class in brewtils.schemas), [60](#)
[IntervalTrigger](#) (class in brewtils.models), [50](#)
[IntervalTriggerSchema](#) (class in brewtils.schemas), [61](#)
[is_different\(\)](#) (brewtils.models.Parameter method), [47](#)
[is_ephemeral](#) (brewtils.models.Request attribute), [47](#)
[is_json](#) (brewtils.models.Request attribute), [48](#)

J

[Job](#) (class in brewtils.models), [49](#)
[JobSchema](#) (class in brewtils.schemas), [61](#)
[JSON_HEADERS](#) (brewtils.rest.client.RestClient attribute), [17](#)

L

[LATEST_VERSION](#) (brewtils.rest.client.RestClient attribute), [17](#)
[LEVELS](#) (brewtils.models.LoggingConfig attribute), [48](#)

[load_bg_system\(\)](#) (brewtils.rest.system_client.SystemClient method), [28](#)
[load_bg_system\(\)](#) (brewtils.SystemClient method), [36](#)
[load_config\(\)](#) (in module brewtils), [37](#)
[logger](#) (brewtils.schema_parser.SchemaParser attribute), [55](#)
[LoggingConfig](#) (class in brewtils.models), [48](#)
[LoggingConfigSchema](#) (class in brewtils.schemas), [61](#)

M

[ModelError](#), [44](#)
[ModelValidationError](#), [44](#)

N

[NoAckAndDieException](#), [44](#)
[normalize_url_prefix\(\)](#) (in module brewtils.rest), [29](#)
[NotFoundError](#), [44](#)

O

[on_cancelok\(\)](#) (brewtils.request_consumer.RequestConsumer method), [53](#)
[on_channel_closed\(\)](#) (brewtils.request_consumer.RequestConsumer method), [53](#)
[on_channel_open\(\)](#) (brewtils.request_consumer.RequestConsumer method), [54](#)
[on_connection_closed\(\)](#) (brewtils.request_consumer.RequestConsumer method), [54](#)
[on_connection_open\(\)](#) (brewtils.request_consumer.RequestConsumer method), [54](#)
[on_consumer_cancelled\(\)](#) (brewtils.request_consumer.RequestConsumer method), [54](#)
[on_message\(\)](#) (brewtils.request_consumer.RequestConsumer method), [54](#)
[on_message_callback_complete\(\)](#) (brewtils.request_consumer.RequestConsumer method), [54](#)
[open_channel\(\)](#) (brewtils.request_consumer.RequestConsumer method), [54](#)
[open_connection\(\)](#) (brewtils.request_consumer.RequestConsumer method), [55](#)
[opts](#) (brewtils.schemas.CommandSchema attribute), [61](#)
[opts](#) (brewtils.schemas.CronTriggerSchema attribute), [62](#)
[opts](#) (brewtils.schemas.DateTriggerSchema attribute), [61](#)
[opts](#) (brewtils.schemas.EventSchema attribute), [61](#)
[opts](#) (brewtils.schemas.InstanceSchema attribute), [60](#)
[opts](#) (brewtils.schemas.IntervalTriggerSchema attribute), [61](#)
[opts](#) (brewtils.schemas.JobSchema attribute), [61](#)
[opts](#) (brewtils.schemas.LoggingConfigSchema attribute), [61](#)
[opts](#) (brewtils.schemas.ParameterSchema attribute), [61](#)
[opts](#) (brewtils.schemas.PatchSchema attribute), [61](#)
[opts](#) (brewtils.schemas.PrincipalSchema attribute), [61](#)

opts (brewtils.schemas.QueueSchema attribute), 61
opts (brewtils.schemas.RefreshTokenSchema attribute), 61
opts (brewtils.schemas.RequestSchema attribute), 61
opts (brewtils.schemas.RoleSchema attribute), 61
opts (brewtils.schemas.SystemSchema attribute), 60
OUTPUT_TYPES (brewtils.models.Command attribute), 47
OUTPUT_TYPES (brewtils.models.Request attribute), 47

P

Parameter (class in brewtils.models), 47
parameter() (in module brewtils), 30
parameter() (in module brewtils.decorators), 39
parameter_keys() (brewtils.models.Command method), 47
parameters() (in module brewtils.decorators), 40
ParameterSchema (class in brewtils.schemas), 61
parse() (in module brewtils.choices), 39
parse_command() (brewtils.schema_parser.SchemaParser class method), 55
parse_event() (brewtils.schema_parser.SchemaParser class method), 56
parse_exception_as_json() (in module brewtils.errors), 45
parse_instance() (brewtils.schema_parser.SchemaParser class method), 56
parse_job() (brewtils.schema_parser.SchemaParser class method), 56
parse_logging_config() (brewtils.schema_parser.SchemaParser class method), 56
parse_parameter() (brewtils.schema_parser.SchemaParser class method), 56
parse_patch() (brewtils.schema_parser.SchemaParser class method), 56
parse_principal() (brewtils.schema_parser.SchemaParser class method), 57
parse_queue() (brewtils.schema_parser.SchemaParser class method), 57
parse_refresh_token() (brewtils.schema_parser.SchemaParser class method), 57
parse_request() (brewtils.schema_parser.SchemaParser class method), 57
parse_role() (brewtils.schema_parser.SchemaParser class method), 57
parse_system() (brewtils.schema_parser.SchemaParser class method), 58
patch_instance() (brewtils.rest.client.RestClient method), 19
patch_job() (brewtils.rest.client.RestClient method), 20
patch_request() (brewtils.rest.client.RestClient method), 20
patch_system() (brewtils.rest.client.RestClient method), 20

PatchOperation (class in brewtils.models), 48
PatchSchema (class in brewtils.schemas), 61
pause_job() (brewtils.rest.easy_client.EasyClient method), 24
Plugin (class in brewtils), 31
Plugin (class in brewtils.plugin), 50
plugin_param() (in module brewtils.decorators), 41
PluginBase (in module brewtils.plugin), 52
PluginError, 44
PluginParamError, 44
PluginValidationError, 44
post_event() (brewtils.rest.client.RestClient method), 20
post_jobs() (brewtils.rest.client.RestClient method), 20
post_requests() (brewtils.rest.client.RestClient method), 20
post_systems() (brewtils.rest.client.RestClient method), 21
Principal (class in brewtils.models), 49
PrincipalSchema (class in brewtils.schemas), 61
process_admin_message() (brewtils.Plugin method), 33
process_admin_message() (brewtils.plugin.Plugin method), 52
process_message() (brewtils.Plugin method), 33
process_message() (brewtils.plugin.Plugin method), 52
process_request_message() (brewtils.Plugin method), 33
process_request_message() (brewtils.plugin.Plugin method), 52
publish_event() (brewtils.rest.easy_client.EasyClient method), 24

Q

Queue (class in brewtils.models), 49
QUEUE_CLEARED (brewtils.models.Events attribute), 49
QueueSchema (class in brewtils.schemas), 61

R

reconnect() (brewtils.request_consumer.RequestConsumer method), 55
reference() (brewtils.choices.FunctionTransformer static method), 38
refresh() (brewtils.rest.client.RestClient method), 21
RefreshToken (class in brewtils.models), 49
RefreshTokenSchema (class in brewtils.schemas), 61
register() (in module brewtils.decorators), 42
RemotePlugin (class in brewtils), 33
RemotePlugin (class in brewtils.plugin), 52
remove_instance() (brewtils.rest.easy_client.EasyClient method), 24
remove_job() (brewtils.rest.easy_client.EasyClient method), 24
remove_system() (brewtils.rest.easy_client.EasyClient method), 25
RepublishRequestException, 44

- Request (class in brewtils.models), 47
- REQUEST_COMPLETED (brewtils.models.Events attribute), 49
- REQUEST_CREATED (brewtils.models.Events attribute), 49
- REQUEST_STARTED (brewtils.models.Events attribute), 49
- RequestConsumer (class in brewtils.request_consumer), 53
- RequestFailedError, 44
- RequestForbidden, 44
- RequestProcessException, 45
- RequestProcessingError, 45
- RequestPublishException, 45
- RequestSchema (class in brewtils.schemas), 61
- RequestStatusTransitionError, 45
- RequestTemplate (class in brewtils.models), 49
- RestClient (class in brewtils.rest.client), 17
- RestClientError, 45
- RestConnectionError, 45
- RestError, 45
- RestServerError, 45
- resume_job() (brewtils.rest.easy_client.EasyClient method), 25
- Role (class in brewtils.models), 49
- RoleSchema (class in brewtils.schemas), 61
- run() (brewtils.Plugin method), 33
- run() (brewtils.plugin.Plugin method), 52
- run() (brewtils.request_consumer.RequestConsumer method), 55
- ## S
- SaveError, 45
- schema (brewtils.models.Choices attribute), 48
- schema (brewtils.models.Command attribute), 47
- schema (brewtils.models.CronTrigger attribute), 50
- schema (brewtils.models.DateTrigger attribute), 49
- schema (brewtils.models.Event attribute), 48
- schema (brewtils.models.Instance attribute), 47
- schema (brewtils.models.IntervalTrigger attribute), 50
- schema (brewtils.models.Job attribute), 49
- schema (brewtils.models.LoggingConfig attribute), 48
- schema (brewtils.models.Parameter attribute), 47
- schema (brewtils.models.PatchOperation attribute), 48
- schema (brewtils.models.Queue attribute), 49
- schema (brewtils.models.Request attribute), 48
- schema (brewtils.models.RequestTemplate attribute), 49
- schema (brewtils.models.System attribute), 46
- SchemaParser (class in brewtils.schema_parser), 55
- send() (brewtils.rest.client.TimeoutAdapter method), 21
- send_bg_request() (brewtils.rest.system_client.SystemClient method), 28
- send_bg_request() (brewtils.SystemClient method), 36
- serialize_command() (brewtils.schema_parser.SchemaParser class method), 58
- serialize_event() (brewtils.schema_parser.SchemaParser class method), 58
- serialize_instance() (brewtils.schema_parser.SchemaParser class method), 58
- serialize_job() (brewtils.schema_parser.SchemaParser class method), 58
- serialize_logging_config() (brewtils.schema_parser.SchemaParser class method), 59
- serialize_parameter() (brewtils.schema_parser.SchemaParser class method), 59
- serialize_patch() (brewtils.schema_parser.SchemaParser class method), 59
- serialize_principal() (brewtils.schema_parser.SchemaParser class method), 59
- serialize_queue() (brewtils.schema_parser.SchemaParser class method), 59
- serialize_refresh_token() (brewtils.schema_parser.SchemaParser class method), 59
- serialize_request() (brewtils.schema_parser.SchemaParser class method), 60
- serialize_role() (brewtils.schema_parser.SchemaParser class method), 60
- serialize_system() (brewtils.schema_parser.SchemaParser class method), 60
- setup_logger() (in module brewtils.log), 16
- start_consuming() (brewtils.request_consumer.RequestConsumer method), 55
- status (brewtils.models.Request attribute), 48
- STATUS_LIST (brewtils.models.Request attribute), 47
- STATUS_TYPES (brewtils.models.Job attribute), 49
- stop() (brewtils.request_consumer.RequestConsumer method), 55
- stop() (brewtils.stoppable_thread.StoppableThread method), 62
- stop_consuming() (brewtils.request_consumer.RequestConsumer method), 55
- StoppableThread (class in brewtils.stoppable_thread), 62
- stopped() (brewtils.stoppable_thread.StoppableThread method), 62
- SUPPORTED_HANDLERS (brewtils.models.LoggingConfig attribute), 48
- System (class in brewtils.models), 46
- system() (in module brewtils), 31
- system() (in module brewtils.decorators), 39
- SYSTEM_CREATED (brewtils.models.Events attribute), 49
- SYSTEM_REMOVED (brewtils.models.Events attribute), 49
- SYSTEM_UPDATED (brewtils.models.Events attribute),

49

SystemClient (class in brewtils), 33

SystemClient (class in brewtils.rest.system_client), 26

SystemSchema (class in brewtils.schemas), 60

T

TimeoutAdapter (class in brewtils.rest.client), 21

TimeoutExceededError, 45

TRIGGER_TYPES (brewtils.models.Job attribute), 49

TYPES (brewtils.models.Choices attribute), 48

TYPES (brewtils.models.Parameter attribute), 47

U

unwrap_envelope() (brewtils.schemas.PatchSchema method), 61

update_instance_status() (brewtils.rest.easy_client.EasyClient method), 25

update_request() (brewtils.rest.easy_client.EasyClient method), 25

update_system() (brewtils.rest.easy_client.EasyClient method), 25

url() (brewtils.choices.FunctionTransformer static method), 39

url_args (brewtils.choices.FunctionTransformer attribute), 39

V

ValidationError, 45

W

wait() (brewtils.stoppable_thread.StoppableThread method), 62

WaitExceededError (in module brewtils.errors), 45

who_am_i() (brewtils.rest.easy_client.EasyClient method), 25

wrap_envelope() (brewtils.schemas.PatchSchema method), 61