# Brewtils Documentation

***Release 3.16.0***

**Logan Asher Jones**

**Apr 14, 2023**

# Contents

CHAPTER 1

# Brewtils

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

## 1.1 Features

Brewtils helps you interact with beer-garden.

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

## 1.2 Installation

To install brewtils, run this command in your terminal:

```
$ pip install brewtils
```

Or add it to your `requirements.txt`

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

## 1.3 Quick Start

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

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

@system
class HelloWorld(object):

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

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

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

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

```
from brewtils.rest.system_client import SystemClient

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

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

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

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

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

```
from brewtils.rest.easy_client import EasyClient

client = EasyClient('localhost', 2337)

systems = client.find_systems()

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

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

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

## 1.4 Documentation

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

CHAPTER 2

# Installation

## 2.1 Stable release

To install Brewtils, run this command in your terminal:

```
$ pip install brewtils
```

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

If you don't have pip installed, this Python installation guide can guide you through the process.

## 2.2 From sources

The sources for Brewtils can be downloaded from the Github repo.

You can either clone the public repository:

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

Or download the tarball:

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

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

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

CHAPTER 3

# brewtils

## 3.1 brewtils package

### 3.1.1 Subpackages

#### brewtils.resolvers package

#### Submodules

#### brewtils.resolvers.bytes module

**class** brewtils.resolvers.bytes.**BytesResolver**(*easy_client*)
Bases: *brewtils.resolvers.ResolverBase*

Resolver that uses the Beergarden file API

**download**(*value*, *definition*)

**should_download**(*value*, *definition*)

**should_upload**(*value*, *definition*)

**upload**(*value*, *definition*)

#### brewtils.resolvers.chunks module

**class** brewtils.resolvers.chunks.**ChunksResolver**(*easy_client*)
Bases: *brewtils.resolvers.ResolverBase*

Resolver that uses the Beergarden chunks API

**download**(*value*, *definition*)

**should_download**(*value*, *definition*)

**should_upload**(*value*, *definition*)
Parameter type must be Base64 and the value must be either:

- String representation of a valid filename.
- An IOBase object

**upload**(*value*, *definition*)

### brewtils.resolvers.identity module

**class** brewtils.resolvers.identity.**IdentityResolver**
Bases: *brewtils.resolvers.ResolverBase*

Resolver that doesn't actually resolve anything

On the upload side this is used to ensure that Resolvables always work when used in a SystemClient. For example, if you're using a SystemClient to execute a command with a Bytes parameter but you already have a Resolvable for that parameter, this makes that work.

On the download side this is used to support autoresolve=False parameters. If a definition specifies "autoresolve": False as part of the type_info dictionary then the parameter WILL NOT be resolved before the command function is invoked. Instead, the Resolvable itself will be passed as that parameter. This might be useful if you wanted to farm out a bytes object to multiple children commands without needing to re-upload the same bytes every time.

**download**(*value*, *definition*)

**should_download**(*value*, *definition*)

**should_upload**(*value*, *definition*)

**upload**(*value*, *definition*)

### brewtils.resolvers.manager module

**class** brewtils.resolvers.manager.**ResolutionManager**(***kwargs*)
Bases: object

Parameter resolution manager

This class is used under-the-hood for various plugin functions. Its purpose is to remove all the various cleanup and housekeeping steps involved in resolving parameters. An example of an unresolved parameter is a dictionary which represents a bytes object. In this case the user wants the open file descriptor, not the random dictionary that they don't know how to process. The parameter resolver helps handle these scenarios.

This is intended for internal use for the plugin class.

**resolve**(*values*, *definitions=None*, *upload=True*)
Iterate through parameters, resolving as necessary

**Parameters**

- **values** – Dictionary of request parameter values
- **definitions** – Parameter definitions
- **upload** – Controls which methods will be called on resolvers

**Returns** The resolved parameter dict

brewtils.resolvers.manager.**build_resolver_map**(*easy_client=None*)
Builds all resolvers

### Module contents

**class** brewtils.resolvers.**ResolverBase**
Bases: object

Base for all Resolver implementations

**download**(*value*, *definition*)

**should_download**(*value*, *definition*)

**should_upload**(*value*, *definition*)

**upload**(*value*, *definition*)

## brewtils.rest package

### Submodules

### brewtils.rest.client module

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

HTTP client for communicating with Beer-garden.

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

**Parameters**

- **bg_host** (*str*) – Beer-garden hostname
- **bg_port** (*int*) – Beer-garden port
- **bg_url_prefix** (*str*) – URL path that will be used as a prefix when communicating with Beer-garden. Useful if Beer-garden is running on a URL other than '/'.
- **ssl_enabled** (*bool*) – Whether to use SSL for Beer-garden communication
- **ca_cert** (*str*) – Path to certificate file containing the certificate of the authority that issued the Beer-garden server certificate
- **ca_verify** (*bool*) – Whether to verify Beer-garden server certificate
- **client_cert** (*str*) – Path to client certificate to use when communicating with Beer-garden
- **api_version** (*int*) – Beer-garden API version to use
- **client_timeout** (*int*) – Max time to wait for Beer-garden server response
- **username** (*str*) – Username for Beer-garden authentication
- **password** (*str*) – Password for Beer-garden authentication
- **access_token** (*str*) – Access token for Beer-garden authentication
- **refresh_token** (*deprecated*) – Refresh token for Beer-garden authentication

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

**LATEST_VERSION = 1**

**can_connect**(***kwargs*)
Determine if a connection to the Beer-garden server is possible

**Parameters** ****kwargs** – Keyword arguments to pass to Requests session call

**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.

**delete_chunked_file**(**args*, ***kwargs*)
Performs a GET on the specific File URL

**Parameters**

- **file_id** – File ID
- ****kwargs** – Query parameters to be used in the GET request

**Returns** Requests Response object

**delete_file**(**args*, ***kwargs*)
Performs a DELETE on the specific File URL

**Parameters** **file_id** – File ID

**Returns** Requests Response object

**delete_garden**(**args*, ***kwargs*)
Performs a DELETE on a Garden URL

**Parameters** **garden_name** – Name of Garden to delete

**Returns** Requests Response object

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

**Parameters** **instance_id** – Instance ID

**Returns** Requests Response object

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

**Parameters** **job_id** – Job ID

**Returns** Requests Response object

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

**Parameters** **queue_name** – Queue name

**Returns** Requests Response object

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

**Returns** Requests Response object

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

**Parameters** **system_id** – System ID

**Returns** Requests Response object

**get_chunked_file**(**args*, ***kwargs*)
Performs a GET on the specific File URL

**Parameters**

- **file_id** – File ID
- ****kwargs** – Query parameters to be used in the GET request

**Returns** Requests Response object

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

**Parameters** **command_id** – Command ID

**Returns** Requests Response object

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

**Returns** Requests Response object

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

**Parameters** ****kwargs** (*deprecated*) – Unused. Accepted for compatibility.

**Returns** Requests Response object

**get_file**(**args*, ***kwargs*)
Performs a GET on the specific File URL

**Parameters**

- **file_id** – File ID
- ****kwargs** – Query parameters to be used in the GET request

**Returns** Requests Response object

**get_garden**(**args*, ***kwargs*)
Performs a GET on the Garden URL

**Parameters**

- **garden_name** – Name of garden to retreive
- ****kwargs** – Query parameters to be used in the GET request

**Returns** Requests Response object

**get_gardens**(**args*, ***kwargs*)
Preform a GET on the Garden URL

This fetches all gardens.

**Returns** Requests Response object

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

**Parameters** **instance_id** – Instance ID

**Returns** Requests Response object

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

**Parameters** **job_id** – Job ID

**Returns** Requests Response object

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

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

**Returns** Requests Response object

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

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

**Returns** Requests Response object

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

**Returns** Requests Response object

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

**Parameters** **request_id** – Request ID

**Returns** Requests Response object

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

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

**Returns** Requests Response object

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

**Parameters**

- **system_id** – System ID
- ****kwargs** – Query parameters to be used in the GET request

**Returns** Requests Response object

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

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

**Returns** Requests Response object

**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** Requests Response object

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

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

**Returns** Requests Response object

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

**Parameters** ****kwargs** (*deprecated*) – Unused. Accepted for compatibility.

**Returns** Requests Response object

**patch_admin**(**args*, ***kwargs*)
Performs a PATCH on the admin URL

**Parameters** **payload** – Serialized PatchOperation

**Returns** Requests Response object

**patch_garden**(**args*, ***kwargs*)
Perform a PATCH on a Garden URL

**Parameters**

- **garden_name** – Garden name
- **payload** – Serialized patch operation

**Returns** Request Response object

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

**Parameters**

- **instance_id** – Instance ID
- **payload** – Serialized PatchOperation

**Returns** Requests Response object

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

**Parameters**

- **job_id** – Job ID
- **payload** – Serialized PatchOperation

**Returns** Requests Response object

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

**Parameters**

- **request_id** – Request ID
- **payload** – Serialized PatchOperation

**Returns** Requests Response object

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

**Parameters**

- **system_id** – System ID
- **payload** – Serialized PatchOperation

**Returns** Requests Response object

**post_chunked_file**(**args*, ***kwargs*)
Performs a POST on the file URL.

**Parameters**

- **fd** – A file descriptor
- **file_params** – Metadata about the file
- **current_position** – The current cursor position for the file object

**Returns** A Requests Response object

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

**Parameters**

- **payload** – Serialized new event definition
- **publishers** – Array of publishers to use

**Returns** Requests Response object

**post_execute_job**(*job_id*, *reset_interval=False*)
Performs a POST on the Job Execute URL

**Parameters**

- **job_id** – The ID of the Job
- **reset_interval** – Sets interval of job to restart now

**Returns** Request Response object

**post_export_jobs**(**args*, ***kwargs*)
Perform a POST on the Job export URL.

**Parameters** **payload** – Serialized list of Jobs

**Returns** Requests Response object

**post_file**(**args*, ***kwargs*)
Performs a PUT on the file URL

**Parameters** **data** – Data bytes

**Returns** A Requests Response object

**post_forward**(**args*, ***kwargs*)
Performs a POST on the Forward URL

**Parameters**

- **payload** – The operation to be executed
- ****kwargs** – Keyword arguments to pass to Requests session call

**Returns** The API response

**post_gardens**(**args*, ***kwargs*)
Performs a POST on the Garden URL

**Parameters** **payload** – New Garden definition

**Returns** Requests Response object

**post_import_jobs**(**args*, ***kwargs*)
Perform a POST on the Job import URL.

**Parameters** **payload** – Serialized list of job definitions

**Returns** Requests Response object

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

**Parameters** **payload** – New Job definition

**Returns** Requests Response object

**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** Requests Response object

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

**Parameters** **payload** – New System definition

**Returns** Requests 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 with specified timeout.

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

### brewtils.rest.easy_client module

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

Client for simplified communication with Beergarden

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

**Parameters**

- **bg_host** (*str*) – Beer-garden hostname
- **bg_port** (*int*) – Beer-garden port
- **bg_url_prefix** (*str*) – URL path that will be used as a prefix when communicating with Beer-garden. Useful if Beer-garden is running on a URL other than '/'.
- **ssl_enabled** (*bool*) – Whether to use SSL for Beer-garden communication
- **ca_cert** (*str*) – Path to certificate file containing the certificate of the authority that issued the Beer-garden server certificate
- **ca_verify** (*bool*) – Whether to verify Beer-garden server certificate
- **client_cert** (*str*) – Path to client certificate to use when communicating with Beer-garden
- **api_version** (*int*) – Beer-garden API version to use
- **client_timeout** (*int*) – Max time to wait for Beer-garden server response
- **username** (*str*) – Username for Beer-garden authentication
- **password** (*str*) – Password for Beer-garden authentication
- **access_token** (*str*) – Access token for Beer-garden authentication
- **refresh_token** (*str*) – Refresh token for Beer-garden authentication

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

**Parameters** ****kwargs** – Keyword arguments passed to the underlying Requests method

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

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

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

**Returns** True if the clear was successful

**Return type** bool

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

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

**Returns** True if the clear was successful

**Return type** bool

**create_garden**(*garden*)
Create a new Garden

**Parameters** **garden** (Garden) – The Garden to create

**Returns** The newly-created Garden

**Return type** *Garden*

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

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

**Returns** The newly-created Job

**Return type** *Job*

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

**Parameters**

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

**Keyword Arguments**

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

**Returns** The newly-created Request

**Return type** *Request*

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

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

**Returns** The newly-created system

**Return type** *System*

**delete_chunked_file**(*file_id*)
Delete a given file on the Beer Garden server.

**Parameters** **file_id** – The beer garden-assigned file id.

**Returns** The API response

**download_bytes**(*file_id*)
Download bytes

**Parameters** **file_id** – Id of bytes to download

**Returns** The bytes data

**download_chunked_file**(*file_id*)
Download a chunked file from the Beer Garden server.

**Parameters** **file_id** – The beer garden-assigned file id.

**Returns** A file object

**download_file**(*file_id*, *path*)
Download a file

**Parameters**

- **file_id** – The File id.

- **path** – Location for downloaded file

**Returns** Path to downloaded file

**execute_job**(*job_id*, *reset_interval=False*)
Execute a Job

**Parameters**

- **job_id** (*str*) – The Job ID
- **reset_interval** (*bool*) – Restarts the job's interval time to now if the job's trigger is an interval

**Returns** The returned request

**Return type** *Request*

**export_jobs**(*job_id_list=None*)
Export jobs from an optional job ID list.

If *job_id_list* is None or empty, definitions for all jobs are returned.

**Parameters** **job_id_list** – A list of job IDS, optional

**Returns** A list of job definitions

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

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

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

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

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

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

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

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

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

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

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

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

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

---

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

---

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

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

**Return type** *Request*, None

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

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

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

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

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

**Return type** *System*, None

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

**forward**(*operation*, ***kwargs*)
Forwards an Operation

**Parameters**

- **operation** – The Operation to be forwarded
- ****kwargs** – Keyword arguments to pass to Requests session call

**Returns** The API response

**get_config**()
Get configuration

**Returns** Configuration dictionary

**Return type** dict

**get_garden**(*garden_name*)
Get a Garden

**Parameters** **garden_name** – Name of garden to retrieve

**Returns** The Garden

**get_gardens**()
Get all Gardens.

**Returns** A list of all the Gardens

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

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

**Returns** The Instance

**get_instance_status**(*instance_id*)
Get an Instance's status

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

**Returns** The Instance's status

**get_logging_config**(*system_name=None*, *local=False*)
Get a logging configuration

Note that the system_name is not relevant and is only provided for backward-compatibility.

**Parameters** **system_name** (*str*) – UNUSED

**Returns** The configuration object

**Return type** dict

**get_queues**()
Retrieve all queue information

**Returns** List of all Queues

**Return type** List[*Queue*]

**get_request**(*request_id*)
Get a Request

**Parameters** **request_id** – The Id

**Returns** The Request

**get_system**(*system_id*)
Get a Garden

**Parameters** **system_id** – The Id

**Returns** The System

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

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

**Returns** The User

**Return type** *Principal*

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

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

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

**Return type** dict

**import_jobs**(*job_list*)
Import job definitions from a list of Jobs.

**Parameters** **job_list** – A list of jobs to import

**Returns** A list of the job IDs created

**initialize_instance**(*instance_id*, *runner_id=None*)
Start an Instance

**Parameters**

- **instance_id** (*str*) – The Instance ID
- **runner_id** (*str*) – The PluginRunner ID, if any

**Returns** The updated Instance

**Return type** *Instance*

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

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

**Returns** True if the heartbeat was successful

**Return type** bool

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

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

**Returns** The updated Job

**Return type** *Job*

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

**Parameters**

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

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

**Returns** True if the publish was successful

**Return type** bool

**remove_garden**(*garden_name*)
Remove a unique Garden

**Parameters** **garden_name** (*String*) – Name of Garden to remove

**Returns** True if removal was successful

**Return type** bool

**Raises** NotFoundError – Couldn't find a Garden matching given name

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

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

**Returns** True if the remove was successful

**Return type** bool

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

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

**Returns** True if removal was successful

**Return type** bool

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

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

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

**Returns** True if removal was successful

**Return type** bool

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

**rescan**()
Rescan local plugin directory

**Returns** True if rescan was successful

**Return type** bool

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

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

**Returns** The updated Job

**Return type** *Job*

**update_garden**(*garden*)

**update_instance**(*instance_id*, ***kwargs*)
Update an Instance status

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

**Keyword Arguments**

- **new_status** (*str*) – The new status
- **metadata** (*dict*) – Will be added to existing instance metadata

**Returns** The updated Instance

**Return type** *Instance*

**update_instance_status**(*instance_id*, *new_status*)
Get an Instance's status

**Parameters**

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

**Returns** The updated Instance

**Return type** *Instance*

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

**Parameters**

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

**Returns** The updated response

**Return type** Response

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

**Parameters**

- **system_id** (*str*) – The System ID

- **new_commands** (*Optional[List[Command]]*) – New System commands

**Keyword Arguments**

- **add_instance** (*Instance*) – An Instance to append
- **metadata** (*dict*) – New System metadata
- **description** (*str*) – New System description
- **display_name** (*str*) – New System display name
- **icon_name** (*str*) – New System icon name
- **template** (*str*) – New System template

**Returns** The updated system

**Return type** *System*

**upload_bytes**(*data*)
Upload a file

**Parameters** **data** – The bytes to upload

**Returns** The bytes Resolvable

**upload_chunked_file**(*file_to_upload*, *desired_filename=None*, *file_params=None*)
Upload a given file to the Beer Garden server.

**Parameters**

- **file_to_upload** – Can either be an open file descriptor or a path.
- **desired_filename** – The desired filename, if none is provided it
- **use the basename of the file_to_upload**(*will*) –
- **file_params** – The metadata surrounding the file. Valid Keys: See brewtils File model

**Returns** A BG file ID.

**upload_file**(*path*)
Upload a file

**Parameters** **path** – Path to file

**Returns** The file Resolvable

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

**Returns** The User

**Return type** *Principal*

brewtils.rest.easy_client.**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.rest.easy_client.**handle_response_failure**(*response*, *default_exc=<class 'brewtils.errors.RestError'>*, *raise_404=True*)

Deal with a response with non-2xx status code

**Parameters**

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

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

**Raises**

- `NotFoundError` – Status code 404 and raise_404 is True
- `WaitExceededError` – Status code 408
- `ConflictError` – Status code 409
- `TooLargeError` – Status code 413
- `ValidationError` – Any other 4xx status codes
- `RestConnectionError` – Status code 503
- `default_exc` – Any other status code

brewtils.rest.easy_client.**wrap_response**(*return_boolean=False*, *parse_method=None*, *parse_many=False*, *default_exc=<class 'brewtils.errors.RestError'>*, *raise_404=True*)

Decorator to consolidate response parsing and error handling

**Parameters**

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

**Returns**

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

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

### brewtils.rest.system_client module

**class** brewtils.rest.system_client.**SystemClient**(**args*, ***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 and a system name:

```
client = SystemClient(
    system_name='example_system',
    system_namespace='default',
    bg_host="host",
    bg_port=2337,
)
```

Note: Passing an empty string as the system_namespace parameter will evalutate to the local garden's default namespace.

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 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 instance for each request.

**always_update:** If True the SystemClient will 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 when making the first request and will only be 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 only 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(
    system_name='example_system',
    system_namespace='default',
    bg_host="localhost",
    bg_port=2337,
    blocking=False,
)

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

(continues on next page)

(continued from previous page)

```
# 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 newer 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 every time the target system is updated.

It's also possible to control what happens when a Request results in an ERROR. If the `raise_on_error` parameter is set to False (the default) then Requests that are not successful simply result in a Request with a status of ERROR, and it is the plugin developer's responsibility to check for this case. However, if `raise_on_error` is set to True then this will result in a `RequestFailedError` being raised. This will happen regardless of the value of the `blocking` flag.

**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.

Request that raises:

```
client = SystemClient(
    system_name="foo",
    system_namespace='default',
    bg_host="localhost",
    bg_port=2337,
)

try:
    client.command_that_errors(_raise_on_error=True)
except RequestFailedError:
    print("I could have just ignored this")
```

**Parameters**

- **system_name** (*str*) – Name of the System to make Requests on
- **system_namespace** (*str*) – Namespace of the System to make Requests on
- **version_constraint** (*str*) – System version to make Requests on. Can be specific ('1.0.0') or 'latest'.
- **default_instance** (*str*) – Name of the Instance to make Requests on

- **always_update** (*bool*) – Whether to check if a newer version of the System exists before making each Request. Only relevant if `version_constraint='latest'`
- **timeout** (*int*) – Seconds to wait for a request to complete. 'None' means wait forever.
- **max_delay** (*int*) – Maximum number of seconds to wait between status checks for a created request
- **blocking** (*bool*) – Flag indicating whether creation will block until the Request is complete or return a Future that will complete when the Request does
- **max_concurrent** (*int*) – Maximum number of concurrent requests allowed. Only has an effect when blocking=False.
- **raise_on_error** (*bool*) – Flag controlling whether created Requests that complete with an ERROR state should raise an exception
- **bg_host** (*str*) – Beer-garden hostname
- **bg_port** (*int*) – Beer-garden port
- **bg_url_prefix** (*str*) – URL path that will be used as a prefix when communicating with Beer-garden. Useful if Beer-garden is running on a URL other than '/'.
- **ssl_enabled** (*bool*) – Whether to use SSL for Beer-garden communication
- **ca_cert** (*str*) – Path to certificate file containing the certificate of the authority that issued the Beer-garden server certificate
- **ca_verify** (*bool*) – Whether to verify Beer-garden server certificate
- **client_cert** (*str*) – Path to client certificate to use when communicating with Beer-garden
- **api_version** (*int*) – Beer-garden API version to use
- **client_timeout** (*int*) – Max time to wait for Beer-garden server response
- **username** (*str*) – Username for Beer-garden authentication
- **password** (*str*) – Password for Beer-garden authentication
- **access_token** (*str*) – Access token for Beer-garden authentication
- **refresh_token** (*str*) – Refresh token for Beer-garden authentication

**bg_default_instance**

**bg_system**

**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)

# Create two callables - one with a parameter and one without
uncreated_requests = [
    client.create_bg_request('command_1', arg_1='Hi!'),
    client.create_bg_request('command_2'),
]
```

(continues on next page)

(continued from previous page)

```
# Calling creates and sends the request
# The result of each is a future because blocking=False on the SystemClient
futures = [req() for req in uncreated_requests]

# Wait for all the futures to complete
concurrent.futures.wait(futures)
```

**Parameters**

- **command_name** (*str*) – Name of the Command to send
- **kwargs** (*dict*) – Will be passed as parameters when creating the Request

**Returns** Partial that will create and execute a Beer-garden request when called

**Raises** `AttributeError` – System does not have a Command with the given command_name

**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.

**Returns** None

**Raises** `FetchError` – Unable to find a matching System

**send_bg_request**(**args*, ***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**

- **args** (*list*) – Unused. Passing positional parameters indicates a bug
- **kwargs** (*dict*) – All necessary request parameters, including Beer-garden internal parameters

**Returns** A completed Request object blocking=False: A future that will be completed when the Request does

**Return type** blocking=True

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

## 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

### brewtils.test package

### Submodules

### brewtils.test.comparable module

Module to simplify model comparisons.

**Warning:** This module was created to simplify testing. As such, it's not recommended for production use.

**Warning:** This module subject to change outside of the normal deprecation cycle.

Seriously, this is a 'use at your own risk' kind of thing.

brewtils.test.comparable.**assert_instance_equal**(*obj1*, *obj2*, *do_raise=False*, ***kwargs*)

Wrapper that will translate AssertionError to a boolean.

This is a safety measure in case these functions are used outside of a testing context. This isn't recommended, but naked asserts are still unacceptable in any packaged code. This method will translate the various comparison functions to a simple boolean return.

Note that in a testing context the AssertionError is re-raised. This is because it's much more helpful to know the specific assertion that failed, as it could be something nested several levels deep.

**Parameters**

- **obj1** – Passed through to _assert_equal
- **obj2** – Passed through to _assert_equal
- **expected_type** – Both objects will be checked (using isinstance) against this type
- **do_raise** – If True, re-raise any raised AssertionError. This helps with nested comparisons.
- ****kwargs** – Passed through to _assert_equal

**Returns**

**True if the comparison was equal. False if**

- The comparison was not equal and
- do_raise is False and
- called from outside of a testing context

**Return type** bool

**Raises** `AssertionError` – The comparison was not equal. Assertion will be translated to a boolean False if do_raise is False and called from outside of a testing context.

`brewtils.test.comparable.`**`assert_choices_equal`**(*obj1*, *obj2*, *do_raise=False*, ***kwargs*)

Wrapper that will translate AssertionError to a boolean.

This is a safety measure in case these functions are used outside of a testing context. This isn't recommended, but naked asserts are still unacceptable in any packaged code. This method will translate the various comparison functions to a simple boolean return.

Note that in a testing context the AssertionError is re-raised. This is because it's much more helpful to know the specific assertion that failed, as it could be something nested several levels deep.

**Parameters**

- **`obj1`** – Passed through to _assert_equal
- **`obj2`** – Passed through to _assert_equal
- **`expected_type`** – Both objects will be checked (using isinstance) against this type
- **`do_raise`** – If True, re-raise any raised AssertionError. This helps with nested comparisons.
- **`**kwargs`** – Passed through to _assert_equal

**Returns**

**True if the comparison was equal. False if**

- The comparison was not equal and
- do_raise is False and
- called from outside of a testing context

**Return type** bool

**Raises** `AssertionError` – The comparison was not equal. Assertion will be translated to a boolean False if do_raise is False and called from outside of a testing context.

`brewtils.test.comparable.`**`assert_patch_equal`**(*obj1*, *obj2*, *do_raise=False*, ***kwargs*)

Wrapper that will translate AssertionError to a boolean.

This is a safety measure in case these functions are used outside of a testing context. This isn't recommended, but naked asserts are still unacceptable in any packaged code. This method will translate the various comparison functions to a simple boolean return.

Note that in a testing context the AssertionError is re-raised. This is because it's much more helpful to know the specific assertion that failed, as it could be something nested several levels deep.

**Parameters**

- **`obj1`** – Passed through to _assert_equal
- **`obj2`** – Passed through to _assert_equal

- **expected_type** – Both objects will be checked (using isinstance) against this type
- **do_raise** – If True, re-raise any raised AssertionError. This helps with nested comparisons.
- ****kwargs** – Passed through to _assert_equal

**Returns**

**True if the comparison was equal. False if**

- The comparison was not equal and
- do_raise is False and
- called from outside of a testing context

**Return type** bool

**Raises** `AssertionError` – The comparison was not equal. Assertion will be translated to a boolean False if do_raise is False and called from outside of a testing context.

`brewtils.test.comparable.`**`assert_logging_config_equal`**(*obj1*, *obj2*, *do_raise=False*, ***kwargs*)

Wrapper that will translate AssertionError to a boolean.

This is a safety measure in case these functions are used outside of a testing context. This isn't recommended, but naked asserts are still unacceptable in any packaged code. This method will translate the various comparison functions to a simple boolean return.

Note that in a testing context the AssertionError is re-raised. This is because it's much more helpful to know the specific assertion that failed, as it could be something nested several levels deep.

**Parameters**

- **obj1** – Passed through to _assert_equal
- **obj2** – Passed through to _assert_equal
- **expected_type** – Both objects will be checked (using isinstance) against this type
- **do_raise** – If True, re-raise any raised AssertionError. This helps with nested comparisons.
- ****kwargs** – Passed through to _assert_equal

**Returns**

**True if the comparison was equal. False if**

- The comparison was not equal and
- do_raise is False and
- called from outside of a testing context

**Return type** bool

**Raises** `AssertionError` – The comparison was not equal. Assertion will be translated to a boolean False if do_raise is False and called from outside of a testing context.

`brewtils.test.comparable.`**`assert_event_equal`**(*obj1*, *obj2*, *do_raise=False*)

`brewtils.test.comparable.`**`assert_queue_equal`**(*obj1*, *obj2*, *do_raise=False*, ***kwargs*)

Wrapper that will translate AssertionError to a boolean.

This is a safety measure in case these functions are used outside of a testing context. This isn't recommended, but naked asserts are still unacceptable in any packaged code. This method will translate the various comparison functions to a simple boolean return.

Note that in a testing context the AssertionError is re-raised. This is because it's much more helpful to know the specific assertion that failed, as it could be something nested several levels deep.

**Parameters**

- **obj1** – Passed through to _assert_equal
- **obj2** – Passed through to _assert_equal
- **expected_type** – Both objects will be checked (using isinstance) against this type
- **do_raise** – If True, re-raise any raised AssertionError. This helps with nested comparisons.
- ****kwargs** – Passed through to _assert_equal

**Returns**

**True if the comparison was equal. False if**

- The comparison was not equal and
- do_raise is False and
- called from outside of a testing context

**Return type** bool

**Raises** `AssertionError` – The comparison was not equal. Assertion will be translated to a boolean False if do_raise is False and called from outside of a testing context.

`brewtils.test.comparable.`**`assert_request_template_equal`**(*obj1*, *obj2*, *do_raise=False*, ***kwargs*)

Wrapper that will translate AssertionError to a boolean.

This is a safety measure in case these functions are used outside of a testing context. This isn't recommended, but naked asserts are still unacceptable in any packaged code. This method will translate the various comparison functions to a simple boolean return.

Note that in a testing context the AssertionError is re-raised. This is because it's much more helpful to know the specific assertion that failed, as it could be something nested several levels deep.

**Parameters**

- **obj1** – Passed through to _assert_equal
- **obj2** – Passed through to _assert_equal
- **expected_type** – Both objects will be checked (using isinstance) against this type
- **do_raise** – If True, re-raise any raised AssertionError. This helps with nested comparisons.
- ****kwargs** – Passed through to _assert_equal

**Returns**

**True if the comparison was equal. False if**

- The comparison was not equal and
- do_raise is False and
- called from outside of a testing context

**Return type** bool

**Raises** `AssertionError` – The comparison was not equal. Assertion will be translated to a boolean False if do_raise is False and called from outside of a testing context.

`brewtils.test.comparable.`**`assert_trigger_equal`**(*obj1*, *obj2*, *do_raise=False*, ***kwargs*)

Wrapper that will translate AssertionError to a boolean.

This is a safety measure in case these functions are used outside of a testing context. This isn't recommended, but naked asserts are still unacceptable in any packaged code. This method will translate the various comparison functions to a simple boolean return.

Note that in a testing context the AssertionError is re-raised. This is because it's much more helpful to know the specific assertion that failed, as it could be something nested several levels deep.

**Parameters**

- **`obj1`** – Passed through to _assert_equal
- **`obj2`** – Passed through to _assert_equal
- **`expected_type`** – Both objects will be checked (using isinstance) against this type
- **`do_raise`** – If True, re-raise any raised AssertionError. This helps with nested comparisons.
- **`**kwargs`** – Passed through to _assert_equal

**Returns**

**True if the comparison was equal. False if**

- The comparison was not equal and
- do_raise is False and
- called from outside of a testing context

**Return type** bool

**Raises** `AssertionError` – The comparison was not equal. Assertion will be translated to a boolean False if do_raise is False and called from outside of a testing context.

`brewtils.test.comparable.`**`assert_command_equal`**(*obj1*, *obj2*, *do_raise=False*)

`brewtils.test.comparable.`**`assert_parameter_equal`**(*obj1*, *obj2*, *do_raise=False*)

`brewtils.test.comparable.`**`assert_principal_equal`**(*obj1*, *obj2*, *do_raise=False*)

`brewtils.test.comparable.`**`assert_request_equal`**(*obj1*, *obj2*, *do_raise=False*)

Assert that two requests are 'equal'.

This is the most complicated due to how we serialize parent and children requests to avoid reference loops.

Parent fields will not serialize their children. That's why compare_parent asserts that the children field is None.

The requests in the children field will not serialize their parents or children. That's why compare_child asserts that both the parent and children fields are None.

`brewtils.test.comparable.`**`assert_role_equal`**(*obj1*, *obj2*, *do_raise=False*)

`brewtils.test.comparable.`**`assert_system_equal`**(*obj1*, *obj2*, *do_raise=False*)

`brewtils.test.comparable.`**`assert_job_equal`**(*obj1*, *obj2*, *do_raise=False*)

`brewtils.test.comparable.`**`assert_request_file_equal`**(*obj1*, *obj2*, *do_raise=False*, ***kwargs*)

Wrapper that will translate AssertionError to a boolean.

This is a safety measure in case these functions are used outside of a testing context. This isn't recommended, but naked asserts are still unacceptable in any packaged code. This method will translate the various comparison functions to a simple boolean return.

Note that in a testing context the AssertionError is re-raised. This is because it's much more helpful to know the specific assertion that failed, as it could be something nested several levels deep.

**Parameters**

- **obj1** – Passed through to _assert_equal
- **obj2** – Passed through to _assert_equal
- **expected_type** – Both objects will be checked (using isinstance) against this type
- **do_raise** – If True, re-raise any raised AssertionError. This helps with nested comparisons.
- ****kwargs** – Passed through to _assert_equal

**Returns**

**True if the comparison was equal. False if**

- The comparison was not equal and
- do_raise is False and
- called from outside of a testing context

**Return type** bool

**Raises** `AssertionError` – The comparison was not equal. Assertion will be translated to a boolean False if do_raise is False and called from outside of a testing context.

`brewtils.test.comparable.`**assert_operation_equal**(*obj1*, *obj2*, *do_raise=False*)

`brewtils.test.comparable.`**assert_runner_equal**(*obj1*, *obj2*, *do_raise=False*, ***kwargs*)

Wrapper that will translate AssertionError to a boolean.

This is a safety measure in case these functions are used outside of a testing context. This isn't recommended, but naked asserts are still unacceptable in any packaged code. This method will translate the various comparison functions to a simple boolean return.

Note that in a testing context the AssertionError is re-raised. This is because it's much more helpful to know the specific assertion that failed, as it could be something nested several levels deep.

**Parameters**

- **obj1** – Passed through to _assert_equal
- **obj2** – Passed through to _assert_equal
- **expected_type** – Both objects will be checked (using isinstance) against this type
- **do_raise** – If True, re-raise any raised AssertionError. This helps with nested comparisons.
- ****kwargs** – Passed through to _assert_equal

**Returns**

**True if the comparison was equal. False if**

- The comparison was not equal and
- do_raise is False and

- called from outside of a testing context

**Return type** bool

**Raises** `AssertionError` – The comparison was not equal. Assertion will be translated to a boolean False if do_raise is False and called from outside of a testing context.

### brewtils.test.fixtures module

`brewtils.test.fixtures.`**`bg_choices`**(**args*, ***kwargs*)

`brewtils.test.fixtures.`**`bg_command`**(**args*, ***kwargs*)
Use the bg_command fixture instead.

`brewtils.test.fixtures.`**`bg_command_2`**(**args*, ***kwargs*)
Use the bg_command fixture instead.

`brewtils.test.fixtures.`**`bg_cron_job`**(**args*, ***kwargs*)
A beer garden cron job

`brewtils.test.fixtures.`**`bg_cron_trigger`**(**args*, ***kwargs*)
A cron trigger as a model.

`brewtils.test.fixtures.`**`bg_date_trigger`**(**args*, ***kwargs*)
A date trigger as a model.

`brewtils.test.fixtures.`**`bg_event`**(**args*, ***kwargs*)
An event as a model.

`brewtils.test.fixtures.`**`bg_garden`**(**args*, ***kwargs*)
An operation as a model.

`brewtils.test.fixtures.`**`bg_instance`**(**args*, ***kwargs*)
An instance as a model.

`brewtils.test.fixtures.`**`bg_interval_job`**(**args*, ***kwargs*)
A beer garden interval job

`brewtils.test.fixtures.`**`bg_interval_trigger`**(**args*, ***kwargs*)
An interval trigger as a model.

`brewtils.test.fixtures.`**`bg_job`**(**args*, ***kwargs*)
A job as a model.

`brewtils.test.fixtures.`**`bg_job_defns_list`**(**args*, ***kwargs*)
A list of job definitions

`brewtils.test.fixtures.`**`bg_job_ids`**(**args*, ***kwargs*)
A list of job IDs

`brewtils.test.fixtures.`**`bg_logging_config`**(**args*, ***kwargs*)
A logging config as a model.

`brewtils.test.fixtures.`**`bg_operation`**(**args*, ***kwargs*)
An operation as a model.

`brewtils.test.fixtures.`**`bg_parameter`**(**args*, ***kwargs*)
Parameter based on the parameter_dict

`brewtils.test.fixtures.`**`bg_patch`**(**args*, ***kwargs*)
A patch as a model.

brewtils.test.fixtures.**bg_patch2**(**args*, ***kwargs*)
A patch as a model.

brewtils.test.fixtures.**bg_principal**(**args*, ***kwargs*)

brewtils.test.fixtures.**bg_queue**(**args*, ***kwargs*)
A queue as a model.

brewtils.test.fixtures.**bg_request**(**args*, ***kwargs*)
A request as a model.

brewtils.test.fixtures.**bg_request_file**(**args*, ***kwargs*)
A request file as a model

brewtils.test.fixtures.**bg_request_template**(**args*, ***kwargs*)
Request template as a bg model.

brewtils.test.fixtures.**bg_resolvable**(**args*, ***kwargs*)

brewtils.test.fixtures.**bg_resolvable_chunk**(**args*, ***kwargs*)

brewtils.test.fixtures.**bg_role**(**args*, ***kwargs*)

brewtils.test.fixtures.**bg_runner**(**args*, ***kwargs*)
A runner as a model.

brewtils.test.fixtures.**bg_system**(**args*, ***kwargs*)
A system as a model.

brewtils.test.fixtures.**bg_system_2**(**args*, ***kwargs*)
A system with a different version.

brewtils.test.fixtures.**child_request**(**args*, ***kwargs*)
A child request as a model.

brewtils.test.fixtures.**child_request_dict**(**args*, ***kwargs*)
A child request represented as a dictionary.

brewtils.test.fixtures.**choices_dict**(**args*, ***kwargs*)
Choices as a dictionary.

brewtils.test.fixtures.**command_dict**(**args*, ***kwargs*)
A command represented as a dictionary.

brewtils.test.fixtures.**command_dict_2**(**args*, ***kwargs*)
A second command represented as a dictionary.

brewtils.test.fixtures.**cron_job_dict**(**args*, ***kwargs*)
A cron job represented as a dictionary.

brewtils.test.fixtures.**cron_trigger_dict**(**args*, ***kwargs*)
A cron trigger as a dictionary.

brewtils.test.fixtures.**date_trigger_dict**(**args*, ***kwargs*)
A cron trigger as a dictionary.

brewtils.test.fixtures.**event_dict**(**args*, ***kwargs*)
An event represented as a dictionary.

brewtils.test.fixtures.**garden_dict**(**args*, ***kwargs*)
A garden as a dictionary.

brewtils.test.fixtures.**instance_dict**(**args*, ***kwargs*)
An instance represented as a dictionary.

brewtils.test.fixtures.**interval_job_dict**(**args*, ***kwargs*)
An interval job represented as a dictionary.

brewtils.test.fixtures.**interval_trigger_dict**(**args*, ***kwargs*)
An interval trigger as a dictionary.

brewtils.test.fixtures.**job_dfn_list_dict**(**args*, ***kwargs*)
A job definition list represented as a dictionary.

brewtils.test.fixtures.**job_dict**(**args*, ***kwargs*)
A date job represented as a dictionary.

brewtils.test.fixtures.**job_dict_for_import**(**args*, ***kwargs*)
A job dict but some keys and values are missing.

brewtils.test.fixtures.**job_id_list_dict**(**args*, ***kwargs*)
A job ID list represented as a dictionary.

brewtils.test.fixtures.**job_ids_dict**(**args*, ***kwargs*)
A list of job IDs represented as a dictionary.

brewtils.test.fixtures.**legacy_role_dict**(**args*, ***kwargs*)

brewtils.test.fixtures.**logging_config_dict**(**args*, ***kwargs*)
A logging config represented as a dictionary.

brewtils.test.fixtures.**nested_parameter_dict**(**args*, ***kwargs*)
Nested Parameter as a dictionary.

brewtils.test.fixtures.**operation_dict**(**args*, ***kwargs*)
An operation as a dictionary.

brewtils.test.fixtures.**parameter_dict**(**args*, ***kwargs*)
Non-nested parameter as a dictionary.

brewtils.test.fixtures.**parent_request**(**args*, ***kwargs*)
A parent request as a model.

brewtils.test.fixtures.**parent_request_dict**(**args*, ***kwargs*)
A parent request represented as a dictionary.

brewtils.test.fixtures.**patch_dict**(**args*, ***kwargs*)
A patch represented as a dictionary.

brewtils.test.fixtures.**patch_dict_no_envelop**(**args*, ***kwargs*)
A patch without an envelope represented as a dictionary.

brewtils.test.fixtures.**patch_dict_no_envelop2**(**args*, ***kwargs*)
A patch without an envelope represented as a dictionary.

brewtils.test.fixtures.**patch_many_dict**(**args*, ***kwargs*)
Multiple patches represented as a dictionary.

brewtils.test.fixtures.**principal_dict**(**args*, ***kwargs*)

brewtils.test.fixtures.**queue_dict**(**args*, ***kwargs*)
A queue represented as a dictionary.

brewtils.test.fixtures.**request_dict**(**args*, ***kwargs*)
A request represented as a dictionary.

brewtils.test.fixtures.**request_file_dict**(**args*, ***kwargs*)
A request file represented as a dictionary.

brewtils.test.fixtures.**request_template_dict**(**args*, ***kwargs*)
Request template as a dictionary.

brewtils.test.fixtures.**resolvable_chunk_dict**(**args*, ***kwargs*)
A resolvable as a dictionary.

brewtils.test.fixtures.**resolvable_dict**(**args*, ***kwargs*)
A resolvable as a dictionary.

brewtils.test.fixtures.**runner_dict**(**args*, ***kwargs*)
A runner as a dictionary.

brewtils.test.fixtures.**system_dict**(**args*, ***kwargs*)
A system represented as a dictionary.

brewtils.test.fixtures.**system_id**(**args*, ***kwargs*)

brewtils.test.fixtures.**ts_2_dt**(**args*, ***kwargs*)
Feb 2, 2017 as a naive datetime.

brewtils.test.fixtures.**ts_2_dt_utc**(**args*, ***kwargs*)
Feb 2, 2017 UTC as timezone-aware datetime.

brewtils.test.fixtures.**ts_2_epoch**(**args*, ***kwargs*)
Feb 2, 2017 UTC as epoch milliseconds.

brewtils.test.fixtures.**ts_dt**(**args*, ***kwargs*)
Jan 1, 2016 as a naive datetime.

brewtils.test.fixtures.**ts_dt_eastern**(**args*, ***kwargs*)
Jan 1, 2016 US/Eastern as timezone-aware datetime.

brewtils.test.fixtures.**ts_dt_utc**(**args*, ***kwargs*)
Jan 1, 2016 UTC as timezone-aware datetime.

brewtils.test.fixtures.**ts_epoch**(**args*, ***kwargs*)
Jan 1, 2016 UTC as epoch milliseconds.

brewtils.test.fixtures.**ts_epoch_eastern**(**args*, ***kwargs*)
Jan 1, 2016 US/Eastern as epoch milliseconds.

#### Module contents

### 3.1.2 Submodules

### 3.1.3 brewtils.choices module

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

**static arg_pair**(*s*)

**static func**(*s*)

**func_args**
alias of __builtin__.list

**static reference**(*s*)

**static url**(*s*)

**url_args**
alias of __builtin__.list

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

**Parameters**

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

**Returns** Dictionary containing the parse results

**Raises** lark.common.ParseError – Unable to find a valid parsing of *input_string*

brewtils.choices.**process_choices**(*choices*)
Process a choices definition into a usable Choices object

**Parameters** **choices** – Raw choices definition, usually from a decorator

**Returns** Dictionary that fully describes a choices specification

**Return type** *Choices*

### 3.1.4 brewtils.config module

brewtils.config.**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** Argument parser with Brewtils arguments loaded

**Return type** ArgumentParser

brewtils.config.**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** Parameters needed to make a connection to Beergarden

**Return type** dict

brewtils.config.**load_config**(*cli_args=True*, *environment=True*, *argument_parser=None*, *bootstrap=False*, ***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. Command line arguments (if `cli_args` argument is not False)
3. **Environment variables using the `BG_` prefix (if `environment` argument** is not False)
4. Default values in the brewtils specification

**Parameters**

- **cli_args** (*Union[bool, list], optional*) – Specifies whether command line should be used as a configuration source - True: Argparse will use the standard sys.argv[1:] - False: Command line arguments will be ignored when loading configuration - List of strings: Will be parsed as CLI args (instead of using sys.argv)
- **environment** (*bool*) – Specifies whether environment variables (with the `BG_` prefix) should be used when loading configuration
- **argument_parser** (*ArgumentParser, optional, deprecated*) – 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

### 3.1.5 brewtils.decorators module

brewtils.decorators.**client**(*_wrapped=None*, *bg_name=None*, *bg_version=None*)
Class decorator that marks a class as a beer-garden Client

Using this decorator is no longer strictly necessary. It was previously required in order to mark a class as being a Beer-garden Client, and contained most of the logic that currently resides in the `parse_client` function. However, that's no longer the case and this currently exists mainly for back-compatibility reasons.

Applying this decorator to a client class does have the nice effect of preventing linters from complaining if any special attributes are used. So that's something.

Those special attributes are below. Note that these are just placeholders until the actual values are populated when the client instance is assigned to a Plugin:

- `_bg_name`: an optional system name
- `_bg_version`: an optional system version
- `_bg_commands`: holds all registered commands
- `_current_request`: Reference to the currently executing request

**Parameters**

- **_wrapped** – The class to decorate. This is handled as a positional argument and shouldn't be explicitly set.
- **bg_name** – Optional plugin name
- **bg_version** – Optional plugin version

**Returns** The decorated class

brewtils.decorators.**command**(*_wrapped=None*, *description=None*, *parameters=None*, *command_type='ACTION'*, *output_type='STRING'*, *schema=None*, *form=None*, *template=None*, *icon_name=None*, *hidden=False*, *metadata=None*)

Decorator for specifying Command details

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.
- **description** – The command description. If not given the first line of the method docstring will be used.
- **parameters** – A list of Command parameters. It's recommended to use @parameter decorators to declare Parameters instead of declaring them here, but it is allowed. Any Parameters given here will be merged with Parameters sourced from decorators and inferred from the method signature.
- **command_type** – The command type. Valid options are Command.COMMAND_TYPES.
- **output_type** – The output type. Valid options are Command.OUTPUT_TYPES.
- **schema** – Deprecated and will be removed in future release. Custom schema definition.
- **form** – Deprecated and will be removed in future release. Custom form definition.
- **template** – Deprecated and will be removed in future release. Custom template definition.
- **icon_name** – The icon name. Should be either a FontAwesome or a Glyphicon name.
- **hidden** – Flag controlling whether the command is visible on the user interface.

- **metadata** – Free-form dictionary

**Returns** The decorated function

brewtils.decorators.**parameter**(*_wrapped=None*, *key=None*, *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*, *type_info=None*, *is_kwarg=None*, *model=None*)

Decorator for specifying Parameter details

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. If the decorated object is a method the key must match an argument name.
- **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.
- **parameters** – Any nested parameters. See also: the 'model' argument.
- **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.
- **form_input_type** – Specify the form input field type (e.g. textarea). Only used for string fields.
- **type_info** – Type-specific information. Mostly reserved for future use.

- **is_kwarg** – Boolean indicating if this parameter is meant to be part of the decorated function's kwargs. Only applies when the decorated object is a method.
- **model** – Class to be used as a model for this parameter. Must be a Python type object, not an instance.

**Returns** The decorated function

brewtils.decorators.**parameters**(**args*, ***kwargs*)

Deprecated since version 3.0: Will be removed in version 4.0. Please use `@command` instead.

Decorator for specifying multiple Parameter definitions at once

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

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

Can become:

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

**Parameters**

- ***args** (*iterable*) – Positional arguments The first (and only) positional argument must be a list containing dictionaries that describe parameters.
- ****kwargs** – Used for bookkeeping. Don't set any of these yourself!

**Returns** The decorated function

**Return type** func

brewtils.decorators.**system**(*_wrapped=None*, *bg_name=None*, *bg_version=None*)

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

Using this decorator is no longer strictly necessary. It was previously required in order to mark a class as being a Beer-garden Client, and contained most of the logic that currently resides in the `parse_client` function. However, that's no longer the case and this currently exists mainly for back-compatibility reasons.

Applying this decorator to a client class does have the nice effect of preventing linters from complaining if any special attributes are used. So that's something.

Those special attributes are below. Note that these are just placeholders until the actual values are populated when the client instance is assigned to a Plugin:

- `_bg_name`: an optional system name
- `_bg_version`: an optional system version
- `_bg_commands`: holds all registered commands
- `_current_request`: Reference to the currently executing request

**Parameters**

- **_wrapped** – The class to decorate. This is handled as a positional argument and shouldn't be explicitly set.
- **bg_name** – Optional plugin name
- **bg_version** – Optional plugin version

**Returns** The decorated class

### 3.1.6 brewtils.display module

brewtils.display.**resolve_form**(*form=None*, *base_dir=None*)
Resolve a form attribute

**Returns** Dictionary that fully describes a form

brewtils.display.**resolve_schema**(*schema=None*, *base_dir=None*)
Resolve a schema attribute

**Returns** Dictionary that fully describes a schema

brewtils.display.**resolve_template**(*template=None*, *base_dir=None*)
Resolve a template attribute

**Returns** Dictionary that fully describes a template

### 3.1.7 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*

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

Special exception that indicates Beer-garden is giving up

This exception is not raised directly, instead it's a special value for a request's error_class attribute. It indicates the request may have information that has not been persisted to the database, but Beer-garden is choosing to abandon further attempts to update it.

Typically indicates a request output is too large or the maximum number of update retry attempts has been reached.

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

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

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

Base exception

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

Error indicating a 409 was raised on the server

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

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

Error Indicating a server Error occurred performing a DELETE

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

Raising an instance will result in a message not being requeued

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

Mixin to log an exception at the CRITICAL level

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

Mixin to log an exception at the DEBUG level

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

Mixin to log an exception at the ERROR level

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

Mixin to log an exception at the INFO level

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

Mixin to log an exception at the WARNING level

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

Error Indicating a server Error occurred performing a GET

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

Base exception for model errors

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

Invalid model

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

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

Error Indicating a 404 was raised on the server

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

Generic error class

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

Error used when plugins have illegal parameters

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

Plugin could not be validated successfully

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

Republish to the end of the message queue

**Parameters**

- **request** – The Request to republish
- **headers** – A dictionary of headers to be used by *brewtils.pika.PikaConsumer*

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

Request returned with a 200, but the status was ERROR

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

Error indicating a 403 was raised on the server

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

Base for exceptions that occur during request processing

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

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

Error while publishing request

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

A status update was an invalid transition

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

Wrapper for all 4XX errors

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

Error indicating a connection error while performing a request

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

Base exception for REST errors

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

Wrapper for all 5XX errors

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

Error Indicating a server Error occurred performing a POST/PUT

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

Mixin that will suppress stacktrace logging

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

Error indicating a timeout occurred waiting for a request to complete

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

Error indicating a 413 was raised on the server

**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.

### 3.1.8 brewtils.log module

Brewtils Logging Utilities

This module streamlines loading logging configuration from Beergarden.

#### Example

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

```
from brewtils import configure_logging, get_connection_info, Plugin

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

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

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

brewtils.log.**configure_logging**(*raw_config*, *namespace=None*, *system_name=None*, *system_version=None*, *instance_name=None*)

Load and enable a logging configuration from Beergarden

WARNING: This method will modify the current logging configuration.

The configuration will be template substituted using the keyword arguments passed to this function. For example, a handler like this:

```
handlers:
    file:
        backupCount: 5
        class: "logging.handlers.RotatingFileHandler"
        encoding: utf8
        formatter: default
        level: INFO
        maxBytes: 10485760
        filename: "$system_name.log"
```

Will result in logging to a file with the same name as the given system_name.

This will also ensure that directories exist for any file-based handlers. Default behavior for the Python logging module is to not create directories that do not already exist, which would dramatically lower the utility of templating.

**Parameters**

- **raw_config** – Configuration to apply
- **namespace** – Used for configuration templating

- **system_name** – Used for configuration templating
- **system_version** – Used for configuration templating
- **instance_name** – Used for configuration templating

**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.**default_config**(*level='INFO'*)
Get a basic logging configuration with the given level

brewtils.log.**find_log_file**()
Find the file name for the first file handler attached to the root logger

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.**read_log_file**(*log_file*, *start_line=None*, *end_line=None*)
Read lines from a log file

**Parameters**

- **log_file** – The file to read from
- **start_line** – Starting line to read
- **end_line** – Ending line to read

**Returns** Lines read from the file

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

### 3.1.9 brewtils.models module

**class** brewtils.models.**BaseModel**

Bases: object

**schema = None**

**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*, *namespace=None*, *local=None*, *template=None*)

Bases: *brewtils.models.BaseModel*

**get_command_by_name**(*command_name*)

Retrieve a particular command from the system

**Parameters** **command_name** (*str*) – The command name

**Returns** The command if it exists, None otherwise

**Return type** *Command*

**get_instance**(*name*)

**get_instance_by_id**(*id*, *raise_missing=False*)

Get an instance that currently exists in the system

**Parameters**

- **id** (*str*) – The instance id
- **raise_missing** (*bool*) – If True, raise an exception if an Instance with the
- **id is not found. If False, will return None in that case.** (*given*) –

**Returns** The instance if it exists, None otherwise

**Return type** *Instance*

**Raises** ModelError – Instance was not found and raise_missing=True

**get_instance_by_name**(*name*, *raise_missing=False*)
Get an instance that currently exists in the system

**Parameters**

- **name** (*str*) – The instance name
- **raise_missing** (*bool*) – If True, raise an exception if an Instance with the
- **name is not found. If False, will return None in that case.** (*given*) –

**Returns** The instance if it exists, None otherwise

**Return type** *Instance*

**Raises** ModelError – Instance was not found and raise_missing=True

**has_different_commands**(*commands*)
Check if a set of commands is different than the current commands

**Parameters** **commands** (*Sequence[Command]*) – Command collection for comparison

**Returns** True if the given Commands differ, False if they are identical

**Return type** bool

**has_instance**(*name*)
Determine if an instance currently exists in the system

**Parameters** **name** (*str*) – The instance name

**Returns** True if an instance with the given name exists, False otherwise

**Return type** bool

**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: *brewtils.models.BaseModel*

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

**schema = 'InstanceSchema'**

**class** brewtils.models.**Command**(*name=None*, *description=None*, *parameters=None*, *command_type=None*, *output_type=None*, *schema=None*, *form=None*, *template=None*, *icon_name=None*, *hidden=False*, *metadata=None*)
Bases: *brewtils.models.BaseModel*

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

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

**get_parameter_by_key**(*key*)
Lookup a Parameter using a given key

**Parameters** **key** (*str*) – The Parameter key to use

**Returns**

A Parameter with the given key

If a Parameter with the given key does not exist None will be returned.

**Return type** *Parameter* (Optional)

**has_different_parameters**(*parameters*)
Determine if parameters differ from the current parameters

**Parameters** **parameters** (*Sequence[Parameter]*) – Parameter collection for comparison

**Returns** True if the given Parameters differ, False if they are identical

**Return type** bool

**parameter_keys**()
Get a list of all Parameter keys

**Returns** A list containing each Parameter's key attribute

**Return type** list[str]

**parameter_keys_by_type**(*desired_type*)
Get a list of all Parameter keys, filtered by Parameter type

**Parameters** **desired_type** (*str*) – Parameter type

**Returns** A list containing matching Parameters' key attribute

**Return type** list[str]

**schema = 'CommandSchema'**

**class** brewtils.models.**Parameter**(*key=None*, *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*, *type_info=None*, *is_kwarg=None*, *model=None*)
Bases: *brewtils.models.BaseModel*

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

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

**is_different**(*other*)

**keys_by_type**(*desired_type*)
Gets all keys by the specified type.

Since parameters can be nested, this method will also return all keys of all nested parameters. The return value is a possibly nested list, where the first value of each list is going to be a string, while the next value is a list.

**Parameters** **desired_type** (*str*) – Desired type

**Returns** An empty list if the type does not exist, otherwise it will be a list containing at least one entry which is a string, each subsequent entry is a nested list with the same structure.

**schema = 'ParameterSchema'**

**class** brewtils.models.**Request**(*system=None*, *system_version=None*, *instance_name=None*, *namespace=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*, *hidden=None*, *updated_at=None*, *status_updated_at=None*, *has_parent=None*, *requester=None*)

Bases: *brewtils.models.RequestTemplate*

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

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

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

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

**classmethod from_template**(*template*, ***kwargs*)

Create a Request instance from a RequestTemplate

**Parameters**

- **template** – The RequestTemplate to use
- ****kwargs** – Optional overrides to use in place of the template's attributes

**Returns** The new Request instance

**is_ephemeral**

**is_json**

**schema = 'RequestSchema'**

**status**

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

Bases: *brewtils.models.BaseModel*

**schema = 'PatchSchema'**

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

Bases: *brewtils.models.BaseModel*

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

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

**schema = 'ChoicesSchema'**

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

Bases: *brewtils.models.BaseModel*

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

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

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

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

**formatter_names**

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

Get a specific plugin logging configuration.

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

**Keyword Arguments** **information for a system** (*Identifying*) –

**Returns** The logging configuration for this system

**handler_names**

**schema = 'LoggingConfigSchema'**

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

Bases: *brewtils.models.BaseModel*

**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**

**COMMAND_PUBLISHING_BLOCKLIST_REMOVE = 49**

**COMMAND_PUBLISHING_BLOCKLIST_SYNC = 48**

**COMMAND_PUBLISHING_BLOCKLIST_UPDATE = 50**

**DB_CREATE = 16**

**DB_DELETE = 18**

**DB_UPDATE = 17**

**ENTRY_STARTED = 31**

**ENTRY_STOPPED = 32**

**FILE_CREATED = 24**

**GARDEN_CREATED = 19**

**GARDEN_ERROR = 28**

**GARDEN_NOT_CONFIGURED = 29**

**GARDEN_REMOVED = 21**

**GARDEN_STARTED = 25**

**GARDEN_STOPPED = 26**

**GARDEN_SYNC = 30**

**GARDEN_UNREACHABLE = 27**

**GARDEN_UPDATED = 20**

**INSTANCE_INITIALIZED = 8**

**INSTANCE_STARTED = 9**

**INSTANCE_STOPPED = 10**

**INSTANCE_UPDATED = 23**

**JOB_CREATED = 33**

**JOB_DELETED = 34**

**JOB_EXECUTED = 43**

**JOB_PAUSED = 35**

**JOB_RESUMED = 36**

**JOB_UPDATED = 41**

**PLUGIN_LOGGER_FILE_CHANGE = 37**

**QUEUE_CLEARED = 14**

**REQUEST_CANCELED = 42**

**REQUEST_COMPLETED = 7**

**REQUEST_CREATED = 5**

**REQUEST_STARTED = 6**

**REQUEST_UPDATED = 22**

**ROLES_IMPORTED = 47**

**ROLE_UPDATED = 46**

**RUNNER_REMOVED = 40**

**RUNNER_STARTED = 38**

**RUNNER_STOPPED = 39**

**SYSTEM_CREATED = 11**

**SYSTEM_REMOVED = 13**

**SYSTEM_UPDATED = 12**

**USERS_IMPORTED = 45**

**USER_UPDATED = 44**

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

Bases: *brewtils.models.BaseModel*

**schema = 'QueueSchema'**

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

Bases: *brewtils.models.BaseModel*

**schema = 'PrincipalSchema'**

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

Bases: *brewtils.models.BaseModel*

**schema = 'LegacyRoleSchema'**

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

**schema = 'RefreshTokenSchema'**

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

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

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

**schema = 'JobSchema'**

**class** brewtils.models.**RequestFile**(*storage_type=None*, *filename=None*, *id=None*)
Bases: *brewtils.models.BaseModel*

**schema = 'RequestFileSchema'**

**class** brewtils.models.**File**(*id=None*, *owner_id=None*, *owner_type=None*, *updated_at=None*, *file_name=None*, *file_size=None*, *chunks=None*, *chunk_size=None*, *owner=None*, *job=None*, *request=None*)
Bases: *brewtils.models.BaseModel*

**schema = 'FileSchema'**

**class** brewtils.models.**FileChunk**(*id=None*, *file_id=None*, *offset=None*, *data=None*, *owner=None*)
Bases: *brewtils.models.BaseModel*

**schema = 'FileChunkSchema'**

**class** brewtils.models.**FileStatus**(*owner_id=None*, *owner_type=None*, *updated_at=None*, *file_name=None*, *file_size=None*, *chunks=None*, *chunk_size=None*, *chunk_id=None*, *file_id=None*, *offset=None*, *data=None*, *valid=None*, *missing_chunks=None*, *expected_max_size=None*, *size_ok=None*, *expected_number_of_chunks=None*, *number_of_chunks=None*, *chunks_ok=None*, *operation_complete=None*, *message=None*)
Bases: *brewtils.models.BaseModel*

**schema = 'FileStatusSchema'**

**class** brewtils.models.**RequestTemplate**(*system=None*, *system_version=None*, *instance_name=None*, *namespace=None*, *command=None*, *command_type=None*, *parameters=None*, *comment=None*, *metadata=None*, *output_type=None*)
Bases: *brewtils.models.BaseModel*

**TEMPLATE_FIELDS = ['system', 'system_version', 'instance_name', 'namespace', 'command'**

**schema = 'RequestTemplateSchema'**

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

**scheduler_attributes**

**scheduler_kwargs**

**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: *brewtils.models.BaseModel*

**scheduler_attributes**

**scheduler_kwargs**

**schema = 'CronTriggerSchema'**

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

Bases: *brewtils.models.BaseModel*

**scheduler_attributes**

**scheduler_kwargs**

**schema = 'IntervalTriggerSchema'**

**class** brewtils.models.**FileTrigger**(*pattern=None*, *path=None*, *recursive=None*, *callbacks=None*)

Bases: *brewtils.models.BaseModel*

**scheduler_attributes**

**scheduler_kwargs**

**schema = 'FileTriggerSchema'**

**class** brewtils.models.**Garden**(*id=None*, *name=None*, *status=None*, *status_info=None*, *namespaces=None*, *systems=None*, *connection_type=None*, *connection_params=None*)

Bases: *brewtils.models.BaseModel*

**GARDEN_STATUSES = set(['BLOCKED', 'ERROR', 'INITIALIZING', 'NOT_CONFIGURED', 'RUNNING'**

**schema = 'GardenSchema'**

**class** brewtils.models.**Operation**(*model=None*, *model_type=None*, *args=None*, *kwargs=None*, *target_garden_name=None*, *source_garden_name=None*, *operation_type=None*)

Bases: *brewtils.models.BaseModel*

**schema = 'OperationSchema'**

**class** brewtils.models.**Resolvable**(*id=None*, *type=None*, *storage=None*, *details=None*)

Bases: *brewtils.models.BaseModel*

**TYPES = ('Base64', 'Bytes')**

**schema = 'ResolvableSchema'**

### 3.1.10 brewtils.pika module

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

Bases: object

Base class for connecting to RabbitMQ using Pika

**Parameters**

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

**connection_parameters**(***kwargs*)

Get ConnectionParameters associated with this client

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

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

**Returns** ConnectionParameters object

**Return type** pika.ConnectionParameters

**connection_url**

Connection URL for this client's connection information

**Type** str

**class** brewtils.pika.**PikaConsumer**(*amqp_url=None*, *queue_name=None*, *panic_event=None*, *logger=None*, *thread_name=None*, ***kwargs*)

Bases: *brewtils.request_handling.RequestConsumer*

Pika message consumer

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

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

**Parameters**

- **amqp_url** – (str) The AMQP url to connect to
- **queue_name** – (str) The name of the queue to connect to
- **on_message_callback** (*func*) – function called to invoke message
- **Must return a Future.** (*processing.*) –
- **panic_event** (*threading.Event*) – Event to be set on a catastrophic failure
- **logger** (*logging.Logger*) – A configured Logger
- **thread_name** (*str*) – Name to use for this thread
- **max_concurrent** – (int) Maximum requests to process concurrently
- **max_reconnect_attempts** (*int*) – Number of times to attempt reconnection to message queue before giving up (default -1 aka never)
- **max_reconnect_timeout** (*int*) – Maximum time to wait before reconnect attempt
- **starting_reconnect_timeout** (*int*) – Time to wait before first reconnect attempt

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

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

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

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

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

**Parameters**

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

**Returns** None

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

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

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

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

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

**Parameters**

- **channel** – The channel
- **args** – Tuple of arguments describing why the channel closed For pika < 1: reply_code (Numeric code indicating close reason), reply_text (String describing close reason). For pika >= 1 exc (Exception describing close).

**Returns** None

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

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

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

**Returns** None

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

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

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

**Parameters**

- **connection** – The connection
- **args** – Tuple of arguments describing why the connection closed For pika < 1: reply_code (Numeric code indicating close reason), reply_text (String describing close reason). For pika >= 1 exc (Exception describing close).

**Returns** None

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

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

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

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

**Returns** None

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

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

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

**Returns** None

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

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

**Parameters**

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

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

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

**Parameters**

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

**Returns** None

**open_channel**()
Open a channel

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

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

**Returns** The SelectConnection object

**run**()
Run the consumer

This method creates a connection to RabbitMQ and starts the IOLoop. The IOLoop will block and allow the SelectConnection to operate. This means that to stop the PikaConsumer we just need to stop the IOLoop.

If the connection closed unexpectedly (the shutdown event is not set) then this will wait a certain amount of time and before attempting to restart it.

Finally, if the maximum number of reconnect attempts have been reached the panic event will be set, which will end the PikaConsumer as well as the Plugin.

**Returns** None

**start_consuming**()
Begin consuming messages

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

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

**Returns** None

**stop**()
Cleanly shutdown

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

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

**Returns** None

**stop_consuming**()
Stop consuming messages

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

**Returns** None

**class** brewtils.pika.**TransientPikaClient**(***kwargs*)
Bases: *brewtils.pika.PikaClient*

Client implementation that creates new connection and channel for each action

**declare_exchange**()

**is_alive**()

**publish**(*message*, ***kwargs*)
Publish a message

**Parameters**

- **message** – Message to publish
- **kwargs** – Additional message properties

**Keyword Arguments**

- **routing_key --** (*) – Routing key to use when publishing
- **headers --** (*) – Headers to be included as part of the message properties
- **expiration --** (*) – Expiration to be included as part of the message properties
- **confirm --** (*) – Flag indicating whether to operate in publisher-acknowledgements mode
- **mandatory --** (*) – Raise if the message can not be routed to any queues
- **priority --** (*) – Message priority

**setup_queue**(*queue_name*, *queue_args*, *routing_keys*)
Create a new queue with queue_args and bind it to routing_keys

### 3.1.11 brewtils.plugin module

**class** brewtils.plugin.**Plugin**(*client=None*, *system=None*, *logger=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.

A Plugin needs certain pieces of information in order to function correctly. These can be grouped into two high-level categories: identifying information and connection information.

Identifying information is how Beer-garden differentiates this Plugin from all other Plugins. If you already have fully-defined System model you can pass that directly to the Plugin (`system=my_system`). However, normally it's simpler to pass the pieces directly:

- `name` (required)
- `version` (required)
- `instance_name` (required, but defaults to "default")
- `namespace`
- `description`
- `icon_name`
- `metadata`
- `display_name`

Connection information tells the Plugin how to communicate with Beer-garden. The most important of these is the `bg_host` (to tell the plugin where to find the Beer-garden you want to connect to):

- `bg_host`
- `bg_port`
- `bg_url_prefix`
- `ssl_enabled`
- `ca_cert`
- `ca_verify`
- `client_cert`

An example plugin might look like this:

```
Plugin(
    name="Test",
    version="1.0.0",
    instance_name="default",
    namespace="test plugins",
    description="A Test",
    bg_host="localhost",
)
```

Plugins use Yapconf for configuration loading, which means that values can be discovered from sources other than direct argument passing. Config can be passed as command line arguments:

```
python my_plugin.py --bg-host localhost
```

Values can also be specified as environment variables with a "BG_" prefix:

```
BG_HOST=localhost python my_plugin.py
```

Plugins service requests using a `concurrent.futures.ThreadPoolExecutor`. The maximum number of threads available is controlled by the `max_concurrent` argument.

**Warning:** Normally the processing of each Request occurs in a distinct thread context. If you need to access shared state please be careful to use appropriate concurrency mechanisms.

**Warning:** The default value for `max_concurrent` is 5, but setting it to 1 is allowed. This means that a Plugin will essentially be single-threaded, but realize this means that if the Plugin invokes a Command on itself in the course of processing a Request then the Plugin **will** deadlock!

**Parameters**

- **client** – Instance of a class annotated with `@system`.
- **bg_host** (*str*) – Beer-garden hostname
- **bg_port** (*int*) – Beer-garden port
- **bg_url_prefix** (*str*) – URL path that will be used as a prefix when communicating with Beer-garden. Useful if Beer-garden is running on a URL other than '/'.
- **ssl_enabled** (*bool*) – Whether to use SSL for Beer-garden communication
- **ca_cert** (*str*) – Path to certificate file containing the certificate of the authority that issued the Beer-garden server certificate
- **ca_verify** (*bool*) – Whether to verify Beer-garden server certificate
- **client_cert** (*str*) – Path to client certificate to use when communicating with Beer-garden. NOTE: This is required to be a cert / key bundle if SSL/TLS is enabled for rabbitmq in your environment.
- **client_key** (*str*) – Path to client key. Not necessary if client_cert is a bundle.
- **api_version** (*int*) – Beer-garden API version to use
- **client_timeout** (*int*) – Max time to wait for Beer-garden server response
- **username** (*str*) – Username for Beer-garden authentication
- **password** (*str*) – Password for Beer-garden authentication
- **access_token** (*str*) – Access token for Beer-garden authentication
- **refresh_token** (*str*) – Refresh token for Beer-garden authentication
- **system** (*brewtils.models.System*) – A Beer-garden System definition. Incompatible with name, version, description, display_name, icon_name, max_instances, and metadata parameters.
- **name** (*str*) – System name
- **version** (*str*) – System version
- **description** (*str*) – System description
- **display_name** (*str*) – System display name
- **icon_name** (*str*) – System icon name

- **max_instances** (*int*) – System maximum instances
- **metadata** (*dict*) – System metadata
- **instance_name** (*str*) – Instance name
- **namespace** (*str*) – Namespace name
- **logger** (logging.Logger) – Logger that will be used by the Plugin. Passing a logger will prevent the Plugin from preforming any additional logging configuration.
- **worker_shutdown_timeout** (*int*) – Time to wait during shutdown to finish processing
- **max_concurrent** (*int*) – Maximum number of requests to process concurrently
- **max_attempts** (*int*) – Number of times to attempt updating of a Request before giving up. Negative numbers are interpreted as no maximum.
- **max_timeout** (*int*) – Maximum amount of time to wait between Request update attempts. Negative numbers are interpreted as no maximum.
- **starting_timeout** (*int*) – Initial time to wait between Request update attempts. Will double on subsequent attempts until reaching max_timeout.
- **mq_max_attempts** (*int*) – Number of times to attempt reconnection to message queue before giving up. Negative numbers are interpreted as no maximum.
- **mq_max_timeout** (*int*) – Maximum amount of time to wait between message queue reconnect attempts. Negative numbers are interpreted as no maximum.
- **mq_starting_timeout** (*int*) – Initial time to wait between message queue reconnect attempts. Will double on subsequent attempts until reaching mq_max_timeout.
- **working_directory** (*str*) – Path to a preferred working directory. Only used when working with bytes parameters.

**bg_host**
Deprecated since version 3.0: bg_host is now in _config (plugin._config.bg_host)

Provided for backward-comptibility

**bg_port**
Deprecated since version 3.0: bg_port is now in _config (plugin._config.bg_port)

Provided for backward-comptibility

**bg_url_prefix**
Deprecated since version 3.0: bg_url_prefix is now in _config (plugin._config.bg_url_prefix)

Provided for backward-comptibility

**bm_client**
Deprecated since version 3.0: bm_client attribute has been renamed to _ez_client.

Provided for backward-comptibility

**ca_cert**
Deprecated since version 3.0: ca_cert is now in _config (plugin._config.ca_cert)

Provided for backward-comptibility

**ca_verify**
Deprecated since version 3.0: ca_verify is now in `_config` (`plugin._config.ca_verify`)

Provided for backward-comptibility

**client**

**client_cert**
Deprecated since version 3.0: client_cert is now in `_config` (`plugin._config.client_cert`)

Provided for backward-comptibility

**connection_parameters**
Deprecated since version 3.0: connection_parameters has been removed. Please use `_config`

Provided for backward-comptibility

**instance**

**instance_name**
Deprecated since version 3.0: instance_name is now in `_config` (`plugin._config.instance_name`)

Provided for backward-comptibility

**logger**
Deprecated since version 3.0: logger attribute has been renamed to `_logger`.

Provided for backward-comptibility

**max_attempts**
Deprecated since version 3.0: max_attempts is now in `_config` (`plugin._config.max_attempts`)

Provided for backward-comptibility

**max_concurrent**
Deprecated since version 3.0: max_concurrent is now in `_config` (`plugin._config.max_concurrent`)

Provided for backward-comptibility

**max_timeout**
Deprecated since version 3.0: max_timeout is now in `_config` (`plugin._config.max_timeout`)

Provided for backward-comptibility

**metadata**
Deprecated since version 3.0: metadata is now part of the `system` attribute (`plugin.system.metadata`)

Provided for backward-comptibility

**run**()

**shutdown_event**
Deprecated since version 3.0: shutdown_event attribute has been renamed to `_shutdown_event`.

Provided for backward-comptibility

**ssl_enabled**
Deprecated since version 3.0: ssl_enabled is now in `_config` (`plugin._config.ssl_enabled`)

Provided for backward-comptibility

**starting_timeout**
Deprecated since version 3.0: starting_timeout is now in _config (plugin._config.starting_timeout)

Provided for backward-comptibility

**system**

**unique_name**

**class** brewtils.plugin.**PluginBase**(**args*, ***kwargs*)
Bases: *brewtils.plugin.Plugin*

Deprecated since version 3.0: Will be removed in version 4.0. Please use Plugin instead.

Plugin alias Provided for backward-comptibility

**class** brewtils.plugin.**RemotePlugin**(**args*, ***kwargs*)
Bases: *brewtils.plugin.Plugin*

Deprecated since version 3.0: Will be removed in version 4.0. Please use Plugin instead.

Plugin alias Provided for backward-comptibility

### 3.1.12 brewtils.queues module

This module currently exists to maintain backwards compatibility.

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

Base class for connecting to RabbitMQ using Pika

**Parameters**

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

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

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

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

**Returns** ConnectionParameters object

**Return type** `pika.ConnectionParameters`

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

**Type** str

### 3.1.13 brewtils.request_handling module

**class** brewtils.request_handling.**AdminProcessor**(*target*, *updater*, *consumer*, *validation_funcs=None*, *logger=None*, *plugin_name=None*, *max_workers=None*, *resolver=None*, *system=None*)
Bases: *brewtils.request_handling.RequestProcessor*

RequestProcessor with slightly modified process method

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

Will invoke the command and set the final status / output / error_class.

Will NOT set the status to IN_PROGRESS or set the request context.

**Parameters**

- **target** – The object to invoke received commands on
- **request** – The parsed Request
- **headers** – Dictionary of headers from the *PikaConsumer*

**Returns** None

**class** brewtils.request_handling.**HTTPRequestUpdater**(*ez_client*, *shutdown_event*, ***kwargs*)
Bases: *brewtils.request_handling.RequestUpdater*

RequestUpdater implementation based around an EasyClient.

**Parameters**

- **ez_client** – EasyClient to use for communication
- **shutdown_event** – *threading.Event* to allow for timely shutdowns
- **logger** – A logger

**Keyword Arguments**

- **logger** – A logger
- **max_attempts** – Max number of unsuccessful updates before discarding the message
- **max_timeout** – Maximum amount of time (seconds) to wait between update attempts

- **starting_timeout** – Starting time to wait (seconds) between update attempts

**shutdown**()

**update_request**(*request*, *headers*)
Sends a Request update to beer-garden

Ephemeral requests do not get updated, so we simply skip them.

**If beergarden appears to be down, it will wait for beergarden to come back** up before updating.

If this is the final attempt to update, we will attempt a known, good request to give some information to the user. If this attempt fails then we simply discard the message.

**Parameters**

- **request** – The request to update
- **headers** – A dictionary of headers from the *PikaConsumer*

**Returns** None

**Raises** `RepublishMessageException` – The Request update failed (any reason)

**class** brewtils.request_handling.**NoopUpdater**(**args*, ***kwargs*)
Bases: *brewtils.request_handling.RequestUpdater*

RequestUpdater implementation that explicitly does not update.

**shutdown**()

**update_request**(*request*, *headers*)

**class** brewtils.request_handling.**RequestConsumer**(**args*, ***kwargs*)
Bases: `threading.Thread`

Base class for consumers

Classes deriving from this are expected to provide a concrete implementation for a specific queue type.

After the consumer is created it will be passed to a `RequestProcessor`. The processor will then set the `on_message_callback` property of the consumer to the correct method.

This means when the consumer receives a message it should invoke its own `_on_message_callback` method with the message body and headers as parameters:

```
self._on_message_callback(body, properties.headers)
```

**static create**(*connection_type=None*, ***kwargs*)
Factory method for consumer creation

Currently the only supported connection_type is "rabbitmq", which will return an instance of `brewtils.pika.PikaConsumer`.

**Parameters**

- **connection_type** (*str*) – String describing connection type
- **kwargs** – Keyword arguments to be passed to the Consumer initializer

**Returns** Concrete instance of RequestConsumer

**Raises** `ValueError` – The specified connection_type does not map to a consumer class

**on_message_callback**

**stop**()

**stop_consuming**()

**class** brewtils.request_handling.**RequestProcessor**(*target*, *updater*, *consumer*, *validation_funcs=None*, *logger=None*, *plugin_name=None*, *max_workers=None*, *resolver=None*, *system=None*)

Bases: object

Class responsible for coordinating Request processing

The RequestProcessor is responsible for the following: - Defining on_message_received callback that will be invoked by the PikaConsumer - Parsing the request - Invoking the command on the target - Formatting the output - Reporting request updates to Beergarden (using a RequestUpdater)

**Parameters**

- **target** – Incoming requests will be invoked on this object
- **updater** – RequestUpdater that will be used for updating requests
- **validation_funcs** – List of functions that will called before invoking a command
- **logger** – A logger
- **plugin_name** – The Plugin's unique name
- **max_workers** – Max number of threads to use in the executor pool

**on_message_received**(*message*, *headers*)

Callback function that will be invoked for received messages

This will attempt to parse the message and then run the parsed Request through all validation functions that this RequestProcessor knows about.

If the request parses cleanly and passes validation it will be submitted to this RequestProcessor's ThreadPoolExecutor for processing.

**Parameters**

- **message** – The message string
- **headers** – The header dictionary

**Returns** A future that will complete when processing finishes

**Raises**

- DiscardMessageException – The request failed to parse correctly
- RequestProcessException – Validation failures should raise a subclass of this

**process_message**(*target*, *request*, *headers*)

Process a message. Intended to be run on an Executor.

Will set the status to IN_PROGRESS, invoke the command, and set the final status / output / error_class.

**Parameters**

- **target** – The object to invoke received commands on
- **request** – The parsed Request
- **headers** – Dictionary of headers from the *PikaConsumer*

**Returns** None

**shutdown**()
Stop the RequestProcessor

**startup**()
Start the RequestProcessor

**class** brewtils.request_handling.**RequestUpdater**
Bases: object

**shutdown**()

**update_request**(*request*, *headers*)

### 3.1.14 brewtils.schema_parser module

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

Serialize and deserialize Brewtils models

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

**classmethod parse**(*data*, *model_class*, *from_string=False*, ***kwargs*)
Convert a JSON string or dictionary into a model object

**Parameters**

- **data** – The raw input
- **model_class** – Class object of the desired model type
- **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 model 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_file**(*file*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a file model object

**Parameters**

- **file** – 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 File object

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

**Parameters**

- **garden** – 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 Garden 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_job_ids**(*job_id_json*, *from_string=False*, ***kwargs*)
Convert raw JSON string containing a list of strings to a list of job ids.

Passes a list of strings through unaltered if from_string is False.

**Parameters**

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

**Returns** A dictionary containing a list of job ids

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

**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_operation**(*operation*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a garden model object

**Parameters**

- **operation** – 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 Operation 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_request_file**(*request_file*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a request file model object

**Parameters**

- **request_file** – 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 RequestFile object

**classmethod parse_resolvable**(*resolvable*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a runner model object

**Parameters**

- **resolvable** – 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 Resolvable 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_runner**(*runner*, *from_string=False*, ***kwargs*)
Convert raw JSON string or dictionary to a runner model object

**Parameters**

- **runner** – 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 Runner 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**(*model*, *to_string=False*, *schema_name=None*, ***kwargs*)
Convert a model object or list of models into a dictionary or JSON string.

This is potentially recursive - here's how this should work:

- Determine the correct schema to use for serializing. This can be explicitly passed as an argument, or it can be determined by inspecting the model to serialize.
- **Determine if the model to serialize is a collection or a single object.**
  - If it's a single object, serialize it and return that.
  - If it's a collection, construct a list by calling this method for each individual item in the collection. Then serialize **that** and return it.

**Parameters**

- **model** – The model or model list
- **to_string** – True to generate a JSON string, False to generate a dictionary
- **schema_name** – Name of schema to use for serializing. If None, will be
- **by inspecting model** (*determined*) –
- ****kwargs** – Additional parameters to be passed to the Schema. Note that the 'many' parameter will be set correctly automatically.

**Returns** A serialized model representation

**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_garden**(*garden*, *to_string=True*, ***kwargs*)
Convert an garden model into serialized form

**Parameters**

- **garden** – The garden 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 garden

**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_job_for_import**(*job*, *to_string=True*, ***kwargs*)
Convert a Job object into serialized form expected by the import endpoint.

The fields that an existing Job would have that a new Job should not (e.g. 'id') are removed by the schema.

**Parameters**

- **job** – The Job 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 the Job

**classmethod serialize_job_ids**(*job_id_list*, *to_string=True*, ***kwargs*)
Convert a list of IDS into serialized form expected by the export endpoint.

**Parameters**

- **job_id_list** – The list of Job id(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 the job IDs

**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_operation**(*operation*, *to_string=True*, ***kwargs*)
Convert an operation model into serialized form

**Parameters**

- **operation** – The operation 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 operation

**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_request_file**(*request_file*, *to_string=True*, ***kwargs*)
Convert a request file model into serialized form

**Parameters**

- **request_file** – The request file 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 file

**classmethod serialize_resolvable**(*resolvable*, *to_string=True*, ***kwargs*)
Convert a resolvable model into serialized form

**Parameters**

- **resolvable** – The resolvable 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 runner

**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_runner**(*runner*, *to_string=True*, ***kwargs*)
Convert a runner model into serialized form

**Parameters**

- **runner** – The runner 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 runner

**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

### 3.1.15 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.**RequestFileSchema**(*strict=True*, ***kwargs*)
Bases: brewtils.schemas.BaseSchema

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

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

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

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

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

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

**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*)
Helper function for parsing the different patch formats.

This exists because previously multiple patches serialized like:

```
{
    "operations": [
        {"operation": "replace", ...},
        {"operation": "replace", ...}
        ...
    ]
}
```

But we also wanted to be able to handle a simple list:

```
[
    {"operation": "replace", ...},
    {"operation": "replace", ...}
    ...
]
```

Patches are now (as of v3) serialized as the latter. Prior to v3 they were serialized as the former.

**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.**LegacyRoleSchema**(*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.**JobExportSchema**(**args*, ***kwargs*)
Bases: *brewtils.schemas.JobSchema*

**make_object**(*data*)

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

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

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

**class** brewtils.schemas.**JobExportListSchema**(*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>**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### 3.1.16 brewtils.specification module

### 3.1.17 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

### 3.1.18 Module contents

brewtils.**client**(*_wrapped=None*, *bg_name=None*, *bg_version=None*)

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

Using this decorator is no longer strictly necessary. It was previously required in order to mark a class as being a Beer-garden Client, and contained most of the logic that currently resides in the `parse_client` function. However, that's no longer the case and this currently exists mainly for back-compatibility reasons.

Applying this decorator to a client class does have the nice effect of preventing linters from complaining if any special attributes are used. So that's something.

Those special attributes are below. Note that these are just placeholders until the actual values are populated when the client instance is assigned to a Plugin:

- `_bg_name`: an optional system name
- `_bg_version`: an optional system version
- `_bg_commands`: holds all registered commands
- `_current_request`: Reference to the currently executing request

**Parameters**

- **_wrapped** – The class to decorate. This is handled as a positional argument and shouldn't be explicitly set.
- **bg_name** – Optional plugin name
- **bg_version** – Optional plugin version

**Returns** The decorated class

brewtils.**command**(*_wrapped=None*, *description=None*, *parameters=None*, *command_type='ACTION'*, *output_type='STRING'*, *schema=None*, *form=None*, *template=None*, *icon_name=None*, *hidden=False*, *metadata=None*)

Decorator for specifying Command details

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.
- **description** – The command description. If not given the first line of the method docstring will be used.
- **parameters** – A list of Command parameters. It's recommended to use @parameter decorators to declare Parameters instead of declaring them here, but it is allowed. Any Parameters given here will be merged with Parameters sourced from decorators and inferred from the method signature.
- **command_type** – The command type. Valid options are Command.COMMAND_TYPES.
- **output_type** – The output type. Valid options are Command.OUTPUT_TYPES.

- **schema** – Deprecated and will be removed in future release. Custom schema definition.
- **form** – Deprecated and will be removed in future release. Custom form definition.
- **template** – Deprecated and will be removed in future release. Custom template definition.
- **icon_name** – The icon name. Should be either a FontAwesome or a Glyphicon name.
- **hidden** – Flag controlling whether the command is visible on the user interface.
- **metadata** – Free-form dictionary

**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*, *parameters=None*, *nullable=None*, *maximum=None*, *minimum=None*, *regex=None*, *form_input_type=None*, *type_info=None*, *is_kwarg=None*, *model=None*)

Decorator for specifying Parameter details

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. If the decorated object is a method the key must match an argument name.
- **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.
- **parameters** – Any nested parameters. See also: the 'model' argument.
- **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.
- **form_input_type** – Specify the form input field type (e.g. textarea). Only used for string fields.
- **type_info** – Type-specific information. Mostly reserved for future use.
- **is_kwarg** – Boolean indicating if this parameter is meant to be part of the decorated function's kwargs. Only applies when the decorated object is a method.
- **model** – Class to be used as a model for this parameter. Must be a Python type object, not an instance.

**Returns** The decorated function

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

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

Using this decorator is no longer strictly necessary. It was previously required in order to mark a class as being a Beer-garden Client, and contained most of the logic that currently resides in the `parse_client` function. However, that's no longer the case and this currently exists mainly for back-compatibility reasons.

Applying this decorator to a client class does have the nice effect of preventing linters from complaining if any special attributes are used. So that's something.

Those special attributes are below. Note that these are just placeholders until the actual values are populated when the client instance is assigned to a Plugin:

- `_bg_name`: an optional system name
- `_bg_version`: an optional system version
- `_bg_commands`: holds all registered commands
- `_current_request`: Reference to the currently executing request

**Parameters**

- **_wrapped** – The class to decorate. This is handled as a positional argument and shouldn't be explicitly set.
- **bg_name** – Optional plugin name
- **bg_version** – Optional plugin version

**Returns** The decorated class

**class** brewtils.**Plugin**(*client=None*, *system=None*, *logger=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.

A Plugin needs certain pieces of information in order to function correctly. These can be grouped into two high-level categories: identifying information and connection information.

Identifying information is how Beer-garden differentiates this Plugin from all other Plugins. If you already have fully-defined System model you can pass that directly to the Plugin (`system=my_system`). However, normally it's simpler to pass the pieces directly:

- `name` (required)
- `version` (required)
- `instance_name` (required, but defaults to "default")
- `namespace`
- `description`
- `icon_name`
- `metadata`
- `display_name`

Connection information tells the Plugin how to communicate with Beer-garden. The most important of these is the `bg_host` (to tell the plugin where to find the Beer-garden you want to connect to):

- `bg_host`
- `bg_port`
- `bg_url_prefix`
- `ssl_enabled`
- `ca_cert`
- `ca_verify`
- `client_cert`

An example plugin might look like this:

```
Plugin(
    name="Test",
    version="1.0.0",
    instance_name="default",
    namespace="test plugins",
    description="A Test",
    bg_host="localhost",
)
```

Plugins use Yapconf for configuration loading, which means that values can be discovered from sources other than direct argument passing. Config can be passed as command line arguments:

```
python my_plugin.py --bg-host localhost
```

Values can also be specified as environment variables with a "BG_" prefix:

```
BG_HOST=localhost python my_plugin.py
```

Plugins service requests using a `concurrent.futures.ThreadPoolExecutor`. The maximum number of threads available is controlled by the `max_concurrent` argument.

**Warning:** Normally the processing of each Request occurs in a distinct thread context. If you need to access shared state please be careful to use appropriate concurrency mechanisms.

**Warning:** The default value for `max_concurrent` is 5, but setting it to 1 is allowed. This means that a Plugin will essentially be single-threaded, but realize this means that if the Plugin invokes a Command on itself in the course of processing a Request then the Plugin **will** deadlock!

**Parameters**

- **client** – Instance of a class annotated with `@system`.
- **bg_host** (*str*) – Beer-garden hostname
- **bg_port** (*int*) – Beer-garden port
- **bg_url_prefix** (*str*) – URL path that will be used as a prefix when communicating with Beer-garden. Useful if Beer-garden is running on a URL other than '/'.
- **ssl_enabled** (*bool*) – Whether to use SSL for Beer-garden communication
- **ca_cert** (*str*) – Path to certificate file containing the certificate of the authority that issued the Beer-garden server certificate
- **ca_verify** (*bool*) – Whether to verify Beer-garden server certificate
- **client_cert** (*str*) – Path to client certificate to use when communicating with Beer-garden. NOTE: This is required to be a cert / key bundle if SSL/TLS is enabled for rabbitmq in your environment.
- **client_key** (*str*) – Path to client key. Not necessary if client_cert is a bundle.
- **api_version** (*int*) – Beer-garden API version to use
- **client_timeout** (*int*) – Max time to wait for Beer-garden server response
- **username** (*str*) – Username for Beer-garden authentication
- **password** (*str*) – Password for Beer-garden authentication
- **access_token** (*str*) – Access token for Beer-garden authentication
- **refresh_token** (*str*) – Refresh token for Beer-garden authentication
- **system** (*brewtils.models.System*) – A Beer-garden System definition. Incompatible with name, version, description, display_name, icon_name, max_instances, and metadata parameters.
- **name** (*str*) – System name
- **version** (*str*) – System version
- **description** (*str*) – System description
- **display_name** (*str*) – System display name
- **icon_name** (*str*) – System icon name
- **max_instances** (*int*) – System maximum instances
- **metadata** (*dict*) – System metadata
- **instance_name** (*str*) – Instance name
- **namespace** (*str*) – Namespace name
- **logger** (`logging.Logger`) – Logger that will be used by the Plugin. Passing a logger will prevent the Plugin from preforming any additional logging configuration.

- **worker_shutdown_timeout** (*int*) – Time to wait during shutdown to finish processing
- **max_concurrent** (*int*) – Maximum number of requests to process concurrently
- **max_attempts** (*int*) – Number of times to attempt updating of a Request before giving up. Negative numbers are interpreted as no maximum.
- **max_timeout** (*int*) – Maximum amount of time to wait between Request update attempts. Negative numbers are interpreted as no maximum.
- **starting_timeout** (*int*) – Initial time to wait between Request update attempts. Will double on subsequent attempts until reaching max_timeout.
- **mq_max_attempts** (*int*) – Number of times to attempt reconnection to message queue before giving up. Negative numbers are interpreted as no maximum.
- **mq_max_timeout** (*int*) – Maximum amount of time to wait between message queue reconnect attempts. Negative numbers are interpreted as no maximum.
- **mq_starting_timeout** (*int*) – Initial time to wait between message queue reconnect attempts. Will double on subsequent attempts until reaching mq_max_timeout.
- **working_directory** (*str*) – Path to a preferred working directory. Only used when working with bytes parameters.

**bg_host**
Deprecated since version 3.0: bg_host is now in `_config` (`plugin._config.bg_host`)

Provided for backward-comptibility

**bg_port**
Deprecated since version 3.0: bg_port is now in _config (`plugin._config.bg_port`)

Provided for backward-comptibility

**bg_url_prefix**
Deprecated since version 3.0: bg_url_prefix is now in `_config` (`plugin._config.bg_url_prefix`)

Provided for backward-comptibility

**bm_client**
Deprecated since version 3.0: bm_client attribute has been renamed to `_ez_client`.

Provided for backward-comptibility

**ca_cert**
Deprecated since version 3.0: ca_cert is now in `_config` (`plugin._config.ca_cert`)

Provided for backward-comptibility

**ca_verify**
Deprecated since version 3.0: ca_verify is now in `_config` (`plugin._config.ca_verify`)

Provided for backward-comptibility

**client**

**client_cert**
Deprecated since version 3.0: client_cert is now in `_config` (`plugin._config.client_cert`)

Provided for backward-comptibility

**connection_parameters**
Deprecated since version 3.0: connection_parameters has been removed. Please use `_config`

Provided for backward-comptibility

**instance**

**instance_name**
Deprecated since version 3.0: instance_name is now in `_config` (`plugin._config.instance_name`)

Provided for backward-comptibility

**logger**
Deprecated since version 3.0: logger attribute has been renamed to `_logger`.

Provided for backward-comptibility

**max_attempts**
Deprecated since version 3.0: max_attempts is now in `_config` (`plugin._config.max_attempts`)

Provided for backward-comptibility

**max_concurrent**
Deprecated since version 3.0: max_concurrent is now in `_config` (`plugin._config.max_concurrent`)

Provided for backward-comptibility

**max_timeout**
Deprecated since version 3.0: max_timeout is now in `_config` (`plugin._config.max_timeout`)

Provided for backward-comptibility

**metadata**
Deprecated since version 3.0: metadata is now part of the `system` attribute (`plugin.system.metadata`)

Provided for backward-comptibility

**run**()

**shutdown_event**
Deprecated since version 3.0: shutdown_event attribute has been renamed to `_shutdown_event`.

Provided for backward-comptibility

**ssl_enabled**
Deprecated since version 3.0: ssl_enabled is now in `_config` (`plugin._config.ssl_enabled`)

Provided for backward-comptibility

**starting_timeout**
Deprecated since version 3.0: starting_timeout is now in `_config` (`plugin._config.starting_timeout`)

Provided for backward-comptibility

**system**

**unique_name**

**class** brewtils.**EasyClient**(**args*, ***kwargs*)
Bases: `object`

Client for simplified communication with Beergarden

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

**Parameters**

- **bg_host** (*str*) – Beer-garden hostname
- **bg_port** (*int*) – Beer-garden port
- **bg_url_prefix** (*str*) – URL path that will be used as a prefix when communicating with Beer-garden. Useful if Beer-garden is running on a URL other than '/'.
- **ssl_enabled** (*bool*) – Whether to use SSL for Beer-garden communication
- **ca_cert** (*str*) – Path to certificate file containing the certificate of the authority that issued the Beer-garden server certificate
- **ca_verify** (*bool*) – Whether to verify Beer-garden server certificate
- **client_cert** (*str*) – Path to client certificate to use when communicating with Beer-garden
- **api_version** (*int*) – Beer-garden API version to use
- **client_timeout** (*int*) – Max time to wait for Beer-garden server response
- **username** (*str*) – Username for Beer-garden authentication
- **password** (*str*) – Password for Beer-garden authentication
- **access_token** (*str*) – Access token for Beer-garden authentication
- **refresh_token** (*str*) – Refresh token for Beer-garden authentication

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

Determine if the Beergarden server is responding.

**Parameters** ****kwargs** – Keyword arguments passed to the underlying Requests method

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

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

**clear_all_queues**()

Cancel and remove all Requests in all queues

**Returns** True if the clear was successful

**Return type** bool

**clear_queue**(*queue_name*)

Cancel and remove all Requests from a message queue

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

**Returns** True if the clear was successful

**Return type** bool

**create_garden**(*garden*)

Create a new Garden

**Parameters** **garden** (Garden) – The Garden to create

**Returns** The newly-created Garden

**Return type** *Garden*

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

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

**Returns** The newly-created Job

**Return type** *Job*

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

**Parameters**

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

**Keyword Arguments**

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

**Returns** The newly-created Request

**Return type** *Request*

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

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

**Returns** The newly-created system

**Return type** *System*

**delete_chunked_file**(*file_id*)
Delete a given file on the Beer Garden server.

**Parameters** **file_id** – The beer garden-assigned file id.

**Returns** The API response

**download_bytes**(*file_id*)
Download bytes

**Parameters** **file_id** – Id of bytes to download

**Returns** The bytes data

**download_chunked_file**(*file_id*)
Download a chunked file from the Beer Garden server.

**Parameters** **file_id** – The beer garden-assigned file id.

**Returns** A file object

**download_file**(*file_id*, *path*)
Download a file

**Parameters**

- **file_id** – The File id.
- **path** – Location for downloaded file

**Returns** Path to downloaded file

**execute_job**(*job_id*, *reset_interval=False*)
Execute a Job

**Parameters**

- **job_id** (*str*) – The Job ID
- **reset_interval** (*bool*) – Restarts the job's interval time to now if the job's trigger is an interval

**Returns** The returned request

**Return type** *Request*

**export_jobs**(*job_id_list=None*)
Export jobs from an optional job ID list.

If *job_id_list* is None or empty, definitions for all jobs are returned.

**Parameters** **job_id_list** – A list of job IDS, optional

**Returns** A list of job definitions

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

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

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

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

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

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

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

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

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

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

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

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

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

---

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

---

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

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

**Return type** *Request*, None

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

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

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

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

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

**Return type** *System*, None

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

**forward**(*operation*, ***kwargs*)
Forwards an Operation

**Parameters**

- **operation** – The Operation to be forwarded
- ****kwargs** – Keyword arguments to pass to Requests session call

**Returns** The API response

**get_config**()
Get configuration

**Returns** Configuration dictionary

**Return type** dict

**get_garden**(*garden_name*)
Get a Garden

**Parameters** **garden_name** – Name of garden to retrieve

**Returns** The Garden

**get_gardens**()
Get all Gardens.

**Returns** A list of all the Gardens

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

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

**Returns** The Instance

**get_instance_status**(*instance_id*)
Get an Instance's status

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

**Returns** The Instance's status

**get_logging_config**(*system_name=None*, *local=False*)
Get a logging configuration

Note that the system_name is not relevant and is only provided for backward-compatibility.

**Parameters** **system_name** (*str*) – UNUSED

**Returns** The configuration object

**Return type** dict

**get_queues**()
Retrieve all queue information

**Returns** List of all Queues

**Return type** List[*Queue*]

**get_request**(*request_id*)
Get a Request

**Parameters** **request_id** – The Id

**Returns** The Request

**get_system**(*system_id*)
Get a Garden

**Parameters** **system_id** – The Id

**Returns** The System

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

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

**Returns** The User

**Return type** *Principal*

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

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

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

**Return type** dict

**import_jobs**(*job_list*)
Import job definitions from a list of Jobs.

**Parameters** **job_list** – A list of jobs to import

**Returns** A list of the job IDs created

**initialize_instance**(*instance_id*, *runner_id=None*)
Start an Instance

**Parameters**

- **instance_id** (*str*) – The Instance ID
- **runner_id** (*str*) – The PluginRunner ID, if any

**Returns** The updated Instance

**Return type** *Instance*

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

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

**Returns** True if the heartbeat was successful

**Return type** bool

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

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

**Returns** The updated Job

**Return type** *Job*

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

**Parameters**

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

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

**Returns** True if the publish was successful

**Return type** bool

**remove_garden**(*garden_name*)
Remove a unique Garden

**Parameters** **garden_name** (*String*) – Name of Garden to remove

**Returns** True if removal was successful

**Return type** bool

**Raises** NotFoundError – Couldn't find a Garden matching given name

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

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

**Returns** True if the remove was successful

**Return type** bool

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

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

**Returns** True if removal was successful

**Return type** bool

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

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

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

**Returns** True if removal was successful

**Return type** bool

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

**rescan**()
Rescan local plugin directory

**Returns** True if rescan was successful

**Return type** bool

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

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

**Returns** The updated Job

**Return type** *Job*

**update_garden**(*garden*)

**update_instance**(*instance_id*, ***kwargs*)
Update an Instance status

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

**Keyword Arguments**

- **new_status** (*str*) – The new status
- **metadata** (*dict*) – Will be added to existing instance metadata

**Returns** The updated Instance

**Return type** *Instance*

**update_instance_status**(*instance_id*, *new_status*)
Get an Instance's status

**Parameters**

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

**Returns** The updated Instance

**Return type** *Instance*

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

**Parameters**

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

**Returns** The updated response

**Return type** Response

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

**Parameters**

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

**Keyword Arguments**

- **add_instance** (Instance) – An Instance to append
- **metadata** (*dict*) – New System metadata
- **description** (*str*) – New System description
- **display_name** (*str*) – New System display name
- **icon_name** (*str*) – New System icon name
- **template** (*str*) – New System template

**Returns** The updated system

**Return type** *System*

**upload_bytes**(*data*)
Upload a file

**Parameters** **data** – The bytes to upload

**Returns** The bytes Resolvable

**upload_chunked_file**(*file_to_upload*, *desired_filename=None*, *file_params=None*)
Upload a given file to the Beer Garden server.

**Parameters**

- **file_to_upload** – Can either be an open file descriptor or a path.
- **desired_filename** – The desired filename, if none is provided it
- **use the basename of the file_to_upload** (*will*) –
- **file_params** – The metadata surrounding the file. Valid Keys: See brewtils File model

**Returns** A BG file ID.

**upload_file**(*path*)
Upload a file

**Parameters** **path** – Path to file

**Returns** The file Resolvable

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

**Returns** The User

**Return type** *Principal*

**class** brewtils.**SystemClient**(**args*, ***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 and a system name:

```
client = SystemClient(
    system_name='example_system',
    system_namespace='default',
    bg_host="host",
    bg_port=2337,
)
```

Note: Passing an empty string as the system_namespace parameter will evalutate to the local garden's default namespace.

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 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 instance for each request.

**always_update:** If True the SystemClient will 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 when making the first request and will only be 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 only 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(
    system_name='example_system',
    system_namespace='default',
    bg_host="localhost",
    bg_port=2337,
    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 newer 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 every time the target system is updated.

It's also possible to control what happens when a Request results in an ERROR. If the `raise_on_error` parameter is set to False (the default) then Requests that are not successful simply result in a Request with a status of ERROR, and it is the plugin developer's responsibility to check for this case. However, if `raise_on_error` is set to True then this will result in a `RequestFailedError` being raised. This will happen regardless of the value of the `blocking` flag.

**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.

Request that raises:

```
client = SystemClient(
    system_name="foo",
    system_namespace='default',
    bg_host="localhost",
    bg_port=2337,
)

try:
    client.command_that_errors(_raise_on_error=True)
except RequestFailedError:
    print("I could have just ignored this")
```

**Parameters**

- **system_name** (*str*) – Name of the System to make Requests on
- **system_namespace** (*str*) – Namespace of the System to make Requests on
- **version_constraint** (*str*) – System version to make Requests on. Can be specific ('1.0.0') or 'latest'.
- **default_instance** (*str*) – Name of the Instance to make Requests on
- **always_update** (*bool*) – Whether to check if a newer version of the System exists before making each Request. Only relevant if `version_constraint='latest'`
- **timeout** (*int*) – Seconds to wait for a request to complete. 'None' means wait forever.

- **max_delay** (*int*) – Maximum number of seconds to wait between status checks for a created request
- **blocking** (*bool*) – Flag indicating whether creation will block until the Request is complete or return a Future that will complete when the Request does
- **max_concurrent** (*int*) – Maximum number of concurrent requests allowed. Only has an effect when blocking=False.
- **raise_on_error** (*bool*) – Flag controlling whether created Requests that complete with an ERROR state should raise an exception
- **bg_host** (*str*) – Beer-garden hostname
- **bg_port** (*int*) – Beer-garden port
- **bg_url_prefix** (*str*) – URL path that will be used as a prefix when communicating with Beer-garden. Useful if Beer-garden is running on a URL other than '/'.
- **ssl_enabled** (*bool*) – Whether to use SSL for Beer-garden communication
- **ca_cert** (*str*) – Path to certificate file containing the certificate of the authority that issued the Beer-garden server certificate
- **ca_verify** (*bool*) – Whether to verify Beer-garden server certificate
- **client_cert** (*str*) – Path to client certificate to use when communicating with Beer-garden
- **api_version** (*int*) – Beer-garden API version to use
- **client_timeout** (*int*) – Max time to wait for Beer-garden server response
- **username** (*str*) – Username for Beer-garden authentication
- **password** (*str*) – Password for Beer-garden authentication
- **access_token** (*str*) – Access token for Beer-garden authentication
- **refresh_token** (*str*) – Refresh token for Beer-garden authentication

**bg_default_instance**

**bg_system**

**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)

# Create two callables - one with a parameter and one without
uncreated_requests = [
    client.create_bg_request('command_1', arg_1='Hi!'),
    client.create_bg_request('command_2'),
]

# Calling creates and sends the request
# The result of each is a future because blocking=False on the SystemClient
futures = [req() for req in uncreated_requests]
```

(continues on next page)

(continued from previous page)

```
# Wait for all the futures to complete
concurrent.futures.wait(futures)
```

**Parameters**

- **command_name** (*str*) – Name of the Command to send
- **kwargs** (*dict*) – Will be passed as parameters when creating the Request

**Returns** Partial that will create and execute a Beer-garden request when called

**Raises** AttributeError – System does not have a Command with the given command_name

**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.

**Returns** None

**Raises** FetchError – Unable to find a matching System

**send_bg_request**(**args*, ***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**

- **args** (*list*) – Unused. Passing positional parameters indicates a bug
- **kwargs** (*dict*) – All necessary request parameters, including Beer-garden internal parameters

**Returns** A completed Request object blocking=False: A future that will be completed when the Request does

**Return type** blocking=True

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

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** Argument parser with Brewtils arguments loaded

**Return type** ArgumentParser

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** Parameters needed to make a connection to Beergarden

**Return type** dict

brewtils.**load_config**(*cli_args=True*, *environment=True*, *argument_parser=None*, *bootstrap=False*, ***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. Command line arguments (if `cli_args` argument is not False)

3. **Environment variables using the `BG_` prefix (if `environment` argument** is not False)
4. Default values in the brewtils specification

**Parameters**

- **`cli_args`** (*`Union[bool, list]`, `optional`*) – Specifies whether command line should be used as a configuration source - True: Argparse will use the standard sys.argv[1:] - False: Command line arguments will be ignored when loading configuration - List of strings: Will be parsed as CLI args (instead of using sys.argv)
- **`environment`** (*`bool`*) – Specifies whether environment variables (with the `BG_` prefix) should be used when loading configuration
- **`argument_parser`** (*`ArgumentParser`, `optional`, `deprecated`*) – 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.`**`configure_logging`**(*raw_config*, *namespace=None*, *system_name=None*, *system_version=None*, *instance_name=None*)

Load and enable a logging configuration from Beergarden

WARNING: This method will modify the current logging configuration.

The configuration will be template substituted using the keyword arguments passed to this function. For example, a handler like this:

```
handlers:
    file:
        backupCount: 5
        class: "logging.handlers.RotatingFileHandler"
        encoding: utf8
        formatter: default
        level: INFO
        maxBytes: 10485760
        filename: "$system_name.log"
```

Will result in logging to a file with the same name as the given system_name.

This will also ensure that directories exist for any file-based handlers. Default behavior for the Python logging module is to not create directories that do not already exist, which would dramatically lower the utility of templating.

**Parameters**

- **`raw_config`** – Configuration to apply
- **`namespace`** – Used for configuration templating
- **`system_name`** – Used for configuration templating
- **`system_version`** – Used for configuration templating
- **`instance_name`** – Used for configuration templating

**Returns** None

brewtils.**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

CHAPTER 4

# Contributing

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

You can contribute in many ways:

## 4.1 Types of Contributions

### 4.1.1 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.

### 4.1.2 Fix Bugs

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

### 4.1.3 Implement Features

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

### 4.1.4 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.

### 4.1.5 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 :)

## 4.2 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:

```
$ flake8 brewtils test
$ pytest test
```

   To get flake8 and pytest, 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.

## 4.3 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 and 3.6+. Check https://github.com/beer-garden/brewtils/pulls and make sure that the tests pass for all supported Python versions.

## 4.4 Tips

To run a subset of tests:

```
$ pytest test/models_test.py::TestSystem::test_instance_names
```

CHAPTER 5

# Credits

## 5.1 Development Leads

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

## 5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

# Brewtils Changelog

## 6.1 3.16.0

4/14/2023

### 6.1.1 Other Changes

- Removed version pinning on the packaging and wrapt dependencies
- Support for python 3.11

## 6.2 3.15.0

8/31/2022

### 6.2.1 Other Changes

- Removed internal references to beer garden v2 naming conventions

## 6.3 3.14.0

6/2/2022

### 6.3.1 Deprecations / Removals

- The ability to customize rendering in the Beer Garden UI by specifying the schema, form, and template parameters in the @command decorator is now deprecated. Future releases of Beer Garden will no longer support this type of customization, so these options should no longer be used in brewtils.

### 6.3.2 Other Changes

- Removed pyjwt dependency
- Added various internal event types

## 6.4 3.13.0

4/12/2022

**NOTE:** This release fixes an issue where client certificates would not be sent to rabbitmq, even if a Plugin was configured to do so. Connecting to rabbitmq with certificates currently requires that the provided certificate be a key and certificate bundle. Please be aware that in certain configurations where the certificate is already set and is not a bundle, your connection to rabbitmq may fail under this release. To fix this, switch your certificate to be a bundle that also includes the key.

### 6.4.1 Bug Fixes

- Plugins will now properly use client certificates when connecting to rabbitmq if provided.
- Fixed an issue that was preventing brewtils from working properly in python 3.10.

## 6.5 3.12.0

3/21/2022

### 6.5.1 Other Changes

- Added new internal event types: `USER_UPDATED` and `USERS_IMPORTED`.

## 6.6 3.11.0

2/9/2022

### 6.6.1 New Features

- `get_gardens` (list of all Gardens) and `update_garden` (apply a new definition to an existing Garden) added to easy client

### 6.6.2 Other Changes

- Permission field added to `UserSchema`.

## 6.7 3.10.0

1/4/2022

### 6.7.1 Bug Fixes

- `Bytes` and `Base64` parameter types can now be defined as optional.
- `RestClient` no longer requires `username` and `password` when using certificates.

## 6.8 3.9.0

12/8/21

### 6.8.1 New Features

- EasyClient `execute_job` method now supports resetting the run interval for jobs with an interval trigger.

## 6.9 3.8.0

11/18/21

### 6.9.1 New Features

- EasyClient now has an `execute_job` method for doing ad-hoc executions of a scheduled job.
- Request now has a `status_updated_at` field representing when the last status changed occured.

### 6.9.2 Other Changes

- Misc additions related to future support of authentication / authorization in Beer Garden.

## 6.10 3.7.1

10/15/21

### 6.10.1 Bug Fixes

- Pinned troublesome dependency `wrapt` to version that's known to not be a problem

### 6.10.2 Other Changes

- Misc additions related to future support of authentication / authorization in Beer Garden.

## 6.11 3.6.0

9/22/21

### 6.11.1 Bug Fixes

- Fixed issues related to interacting with beer-garden urls containing unicode characters (Issue #339 / PR #344)

### 6.11.2 New Features

- Added `export_jobs` and `import_jobs` to EasyClient (Issue #353 / PR #337)
- Added `create_garden` and `remove_garden` to EasyClient (Issue #348 / PR #350)

### 6.11.3 Other Changes

- Added schemas for use in future authorization related features (Issue #345 / PR #347)

## 6.12 3.5.0

8/18/21

### 6.12.1 New Features

- Can now specify proxy parameters when creating RestClients

## 6.13 3.4.0

6/24/21

### 6.13.1 Bug Fixes

- Changed duplicate event enum value (Issue #932 / PR #330)
- Better handling of non-json error responses (Issue #1033 / PR #324)
- No longer ignoring `max_attempts`, `max_timeout`, and `starting_timeout` values (Issue #1028 / PR #323)
- A plugin Client instance can now be reused (Issue #1014 / PR #321)
- Charset in content-type header no longer breaks URL-based display resource loading (Issue #1010 / PR #319)
- URL-based template resolution respects connection configuration (Issue #1009 / PR #318)

- System attributes (like description) can now be cleared (Issue #1002 / PR #317)

### 6.13.2 New Features

- Jobs now have a timeout field (Issue #1046 / PR #329)
- Added `bg_system` and `bg_default_instance` properties to SystemClient (Issue #279 / PR #273)
- Forwarding REST calls now support `blocking` and `timeout` parameters (Issue #895 / PR #325)
- Added support for lambdas as a Choices source (Issue #1004 / PR #322)
- Bytes-type parameters are now supported (Issue #991 / PR #316)
- Systems can now have UI templates (Issue #997 / PR #315)
- Commands now have a metadata field (Issue #358 / PR #314)

### 6.13.3 Other Changes

- Removed support for pika versions below 1.0 (Issue #651 / PR #328)
- SystemClient now has a `__str__` method (Issue #76 / PR #327)
- Dropped official support for Python 3.5 (Issue #1043 / PR #326)
- Added INVALID Request status (PR #325)

## 6.14 3.3.0

4/23/21

### 6.14.1 Bug Fixes

- Better error messages for incorrect parameter definitions (Issue #986 / PR #309)
- Fixed a case where reusing a parameter model could break (Issue #987 / PR #310)

### 6.14.2 New Features

- Support for scheduled job modification (Issue #294 / PR #308)

## 6.15 3.2.1

4/16/21

### 6.15.1 Bug Fixes

- Nullable multi parameters with a model no longer set a problematic default (Issue #769, #983 / PR #305)
- End date is now set correctly for cron-type jobs (Issue #963 / PR #306)
- Order of parameters in the UI now matches the order of decorators (Issue #267, #981 / PR #304)

### 6.15.2 Other Changes

- More type hints for SystemClient and EasyClient methods (Issue #957 / PR #303)

## 6.16 3.2.0

4/1/21

### 6.16.1 New Features

- SystemClient with no parameters will default to the current plugin (Issue #780 / PR #293)
- Added methods to RestClient and EasyClient for using the /api/v1/forward API (PR #301)
- New and improved decorators module (Issue #777 / PR #290)

### 6.16.2 Other Changes

- The @system decorator has been renamed to @client (Issue #927 / PR #297)
- @parameters (plural, with an "s") is now deprecated (Issue #924, PR #299)
- Easier to specify logger name when creating a StoppableThread (Issue #874 / PR #291)

## 6.17 3.1.0

2/5/21

### 6.17.1 Bug Fixes

- SystemClient parameter resolution no longer always fails if no system is assigned (Issue #859 / PR #289)
- Added positional arguments back-compatibility for EasyClient and SystemClient creation (Issue #836 / PR #286)
- Fixed regression relating to old decorator deprecations (Issue #835 / PR #285)

### 6.17.2 Other Changes

- Added 'hidden' field to Request ile model (Issue #414 / PR #288)
- Added 'job' and 'request' fields to File model (Issue #833 / PR #284)

## 6.18 3.0.2

Date: 1/11/21

### 6.18.1 Bug Fixes

- SystemClient no longer disallows creating a Request for a System without a namespace (Issue #827 / PR #281)
- Logs are now written correctly when a Plugin encounters an uncaught exception after initialization (Issue #787 / PR #276)
- Plugin registration will now behave as expected when the list of plugin Commands is empty (Issue #806 / PR #277)

### 6.18.2 New Features

- Added a Rescan method to the EasyClient (Issue #815 / PR #278)

### 6.18.3 Other Changes

- The decorators `command_registrar`, `register`, and `plugin_param` are officially deprecated (Issue #825 / PR #280)

## 6.19 3.0.1

Date: 12/15/20

### 6.19.1 New Features

- Added `client_key` parameter to support separate key and cert files (beer-garden#785)
- Better `SystemClient` error message if a positional parameter is used (beer-garden#775)
- Plugins will now work when connected to a v2 Beer Garden (beer-garden#751)
- Support for file-type parameters (beer-garden#368)

### 6.19.2 Bug Fixes

- Using nested models when defining Parameters now works correctly (beer-garden#354)

### 6.19.3 Other Changes

- Plugins now register a SIGTERM handler for shutdown consistency (beer-garden/#745)

## 6.20 3.0.0

Date: 11/10/20

Note: This is a major upgrade with several breaking changes. Please see the Upgrade Guide for all changes.

### 6.20.1 New Features

- Plugins now automatically load configuration from CLI and environment variables
- Logging configuration is loaded automatically when Plugins are created
- No longer need to pass connection information to System/Easy/Rest Clients
- Parameter choices definition can be a non-list iterable (beer-garden/#512)
- It's now easier to specify an alternate parent when making a request (beer-garden/#336)
- SchemaParser can now directly serialize dicts and Boxes (#239)

### 6.20.2 Bug Fixes

- EasyClient.get_instance_status is deprecated but now actually returns the instance status

### 6.20.3 Other Changes

- Plugins are now multi-threaded by default (#47)
- Better error messages when using SystemClient with raise_on_error=True (beer-garden/#689)
- Various deprecated names have been removed
- Can now defer setting a Plugin client
- EasyClient.get_version returns actual version information instead of Response object
- Using a pika version <1 is deprecated

## 6.21 2.4.15

Date: 10/13/20

### 6.21.1 Bug Fixes

- Fixing command invocation error when request has no parameters (beer-garden/#351)

## 6.22 2.4.14

Date: 1/30/20

### 6.22.1 Bug Fixes

- Better error handling if a request exceeds 16MB size limit (beer-garden/#308)

## 6.23 2.4.13

Date: 1/13/20

### 6.23.1 Bug Fixes

- Requests republished to rabbit are now persistent (beer-garden/#397)

## 6.24 2.4.12

Date: 1/10/20

### 6.24.1 Other Changes

- Reverting a log message level that was incorrectly set to INFO

## 6.25 2.4.11

Date: 12/9/19

### 6.25.1 Other Changes

- Plugins always attempt to notify Beer-garden when terminating (beer-garden/#376)

## 6.26 2.4.10

Date: 11/12/19

### 6.26.1 Bug Fixes

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

## 6.27 2.4.9

Date: 10/30/19

### 6.27.1 Bug Fixes

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

## 6.28 2.4.8

Date: 9/5/19

### 6.28.1 New Features

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

### 6.28.2 Bug Fixes

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

## 6.29 2.4.7

Date: 6/27/19

### 6.29.1 New Features

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

### 6.29.2 Bug Fixes

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

### 6.29.3 Other Changes

- Added compatability with Pika v1 (#130)

## 6.30 2.4.6

Date: 4/19/19

### 6.30.1 Bug Fixes

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

### 6.30.2 Other Changes

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

## 6.31 2.4.5

Date: 2/14/19

### 6.31.1 Bug Fixes

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

### 6.31.2 Other Changes

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

## 6.32 2.4.4

Date: 1/7/19

### 6.32.1 Bug Fixes

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

## 6.33 2.4.3

Date: 11/16/18

### 6.33.1 New Features

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

### 6.33.2 Bug Fixes

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

## 6.34 2.4.2

Date: 10/7/18

### 6.34.1 New Features

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

### 6.34.2 Bug Fixes

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

## 6.35 2.4.1

Date: 09/11/18

### 6.35.1 Other Changes

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

## 6.36 2.4.0

Date: 09/5/18

### 6.36.1 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

### 6.36.2 Other Changes

- Renamed PluginBase to Plugin (old name is aliased)

## 6.37 2.3.7

Date: 07/11/18

### 6.37.1 New Features

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

### 6.37.2 Bug Fixes

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

## 6.38 2.3.6

Date: 06/06/18

### 6.38.1 Other Changes

- Added *has_parent* to request model

## 6.39 2.3.5

Date: 4/17/18

### 6.39.1 Bug Fixes

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

## 6.40 2.3.4

Date: 4/5/18

### 6.40.1 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)

### 6.40.2 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

## 6.41 2.3.3

Date: 3/20/18

### 6.41.1 Bug Fixes

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

## 6.42 2.3.2

Date: 3/7/18

### 6.42.1 Bug Fixes

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

## 6.43 2.3.1

Date: 2/22/18

### 6.43.1 New Features

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

## 6.44 2.3.0

Date: 1/26/18

### 6.44.1 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

### 6.44.2 Bug Fixes

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

### 6.44.3 Other Changes

- Better data integrity by only allowing certain Request status transitions

## 6.45 2.2.1

Date: 1/11/18

### 6.45.1 Bug Fixes

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

## 6.46 2.2.0

Date: 10/23/17

### 6.46.1 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

### 6.46.2 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

### 6.46.3 Other changes

- Added license file

## 6.47 2.1.1

Date: 8/25/17

### 6.47.1 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`

### 6.47.2 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`

### 6.47.3 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

### 6.47.4 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

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## b

# Index

## A



## B

## C

## D

## E



## F



## G

## H

## I



## J



## K



## L

## P

## Q



## R

## S

## T



## U



## V



## W