1 About 1

2 Installation 3

3 Usage 5

4 Development 7

5 API Documentation 9
   5.1 CloudMan API 9
   5.2 Galaxy API 19
   5.3 Toolshed API 88

6 Configuration 97
   6.1 Configuration documents for BioBlend 97

7 Testing 99

8 Getting help 101

9 Related documentation 103

10 Indices and tables 105

Python Module Index 107

Index 109
BioBlend is a Python library for interacting with CloudMan and Galaxy’s API.

BioBlend is supported and tested on:

- Python 2.7, 3.4, 3.5, 3.6 and 3.7
- Galaxy release_14.10 and later.

Conceptually, it makes it possible to script and automate the process of cloud infrastructure provisioning and scaling via CloudMan, and running of analyses via Galaxy. In reality, it makes it possible to do things like this:

- Create a CloudMan compute cluster, via an API and directly from your local machine:

  ```python
  from bioblend.cloudman import CloudManConfig
  from bioblend.cloudman import CloudManInstance
  cfg = CloudManConfig('your cloud access key', 'your cloud secret key', 'My CloudMan', 'ami-ID', 'm1.small', 'password')
  cmi = CloudManInstance.launch_instance(cfg)
  cmi.get_status()
  ```

- Reconnect to an existing CloudMan instance and manipulate it:

  ```python
  from bioblend.cloudman import CloudManInstance
  cmi = CloudManInstance('instance IP', 'password')
  cmi.add_nodes(3)
  cluster_status = cmi.get_status()
  cmi.remove_nodes(2)
  ```

- Interact with Galaxy via a straightforward API:

  ```python
  from bioblend.galaxy import GalaxyInstance
  gi = GalaxyInstance('Galaxy IP', key='your API key')
  libs = gi.libraries.get_libraries()
  gi.workflows.show_workflow('workflow ID')
  gi.workflows.run_workflow('workflow ID', input_dataset_map)
  ```

- Interact with Galaxy via an object-oriented API:
from bioblend.galaxy.objects import GalaxyInstance
gi = GalaxyInstance("URL", "API_KEY")
wf = gi.workflows.list()[0]
hist = gi.histories.list()[0]
inputs = hist.get_datasets()[:2]
input_map = dict(zip(wf.input_labels, inputs))
params = {"Paste1": {"delimiter": "U"}}
wf.run(input_map, "wf_output", params=params)

Note: Although this library allows you to blend these two services into a cohesive unit, the library itself can be used with either service irrespective of the other. For example, you can use it to just manipulate CloudMan clusters or to script the interactions with an instance of Galaxy running on your laptop.
Stable releases of BioBlend are best installed via `pip` or `easy_install` from PyPI using something like:

```
$ pip install bioblend
```

Alternatively, you may install the most current source code from our Git repository, or fork the project on Github. To install from source, do the following:

```
# Clone the repository to a local directory
$ git clone https://github.com/galaxyproject/bioblend.git
# Install the library
$ cd bioblend
$ python setup.py install
```

After installing the library, you will be able to simply import it into your Python environment with `import bioblend`. For details on the available functionality, see the `API documentation`.

BioBlend requires a number of Python libraries. These libraries are installed automatically when BioBlend itself is installed, regardless whether it is installed via PyPi or by running `python setup.py install` command. The current list of required libraries is always available from `setup.py` in the source code repository.

If you also want to run tests locally, some extra libraries are required. To install them, run:

```
$ python setup.py test
```
CHAPTER 3

Usage

To get started using BioBlend, install the library as described above. Once the library becomes available on the given system, it can be developed against. The developed scripts do not need to reside in any particular location on the system.

It is probably best to take a look at the example scripts in docs/examples source directory and browse the API documentation. Beyond that, it’s up to your creativity :-).
Anyone interested in contributing or tweaking the library is more than welcome to do so. To start, simply fork the Git repository on Github and start playing with it. Then, issue pull requests.
BioBlend’s API focuses around and matches the services it wraps. Thus, there are two top-level sets of APIs, each corresponding to a separate service and a corresponding step in the automation process. Note that each of the service APIs can be used completely independently of one another.

Effort has been made to keep the structure and naming of those API’s consistent across the library but because they do bridge different services, some discrepancies may exist. Feel free to point those out and/or provide fixes.

For Galaxy, an alternative object-oriented API is also available. This API provides an explicit modeling of server-side Galaxy instances and their relationships, providing higher-level methods to perform operations such as retrieving all datasets for a given history, etc. Note that, at the moment, the oo API is still incomplete, providing access to a more restricted set of Galaxy modules with respect to the standard one.

## 5.1 CloudMan API

API used to manipulate the instantiated infrastructure. For example, scale the size of the compute cluster, get infrastructure status, get service status.

### 5.1.1 API documentation for interacting with CloudMan

**CloudManLauncher**

```python
class bioblend.cloudman.launch.CloudManLauncher(access_key, secret_key, cloud=None)
```

Define the environment in which this instance of CloudMan will be launched.

Besides providing the credentials, optionally provide the `cloud` object. This object must define the properties required to establish a `boto` connection to that cloud. See this method’s implementation for an example of the required fields. Note that as long the as provided object defines the required fields, it can really by implemented as anything (e.g., a Bunch, a database object, a custom class). If no value for the `cloud` argument is provided, the default is to use the Amazon cloud.

```python
assign_floating_ip(ec2_conn, instance)
```
connect_ec2(a_key, s_key, cloud=None)
Create and return an EC2-compatible connection object for the given cloud.
See _get_cloud_info method for more details on the requirements for the cloud parameter. If no value is provided, the class field is used.

connect_s3(a_key, s_key, cloud=None)
Create and return an S3-compatible connection object for the given cloud.
See _get_cloud_info method for more details on the requirements for the cloud parameter. If no value is provided, the class field is used.

connect_vpc(a_key, s_key, cloud=None)
Establish a connection to the VPC service.
TODO: Make this work with non-default clouds as well.

create_cm_security_group(sg_name='CloudMan', vpc_id=None)
Create a security group with all authorizations required to run CloudMan.
If the group already exists, check its rules and add the missing ones.

Parameters
• sg_name (str) – A name for the security group to be created.
• vpc_id (str) – VPC ID under which to create the security group.

Return type dict
Returns A dictionary containing keys name (with the value being the name of the security group that was created), error (with the value being the error message if there was an error or None if no error was encountered), and ports (containing the list of tuples with port ranges that were opened or attempted to be opened).

Changed in version 0.6.1: The return value changed from a string to a dict

create_key_pair(key_name='cloudman_key_pair')
If a key pair with the provided key_name does not exist, create it.

Parameters sg_name (str) – A name for the key pair to be created.

Return type dict
Returns A dictionary containing keys name (with the value being the name of the key pair that was created), error (with the value being the error message if there was an error or None if no error was encountered), and material (containing the unencrypted PEM encoded RSA private key if the key was created or None if the key already existed).

Changed in version 0.6.1: The return value changed from a tuple to a dict

find_placements(ec2_conn, instance_type, cloud_type, cluster_name=None)
Find a list of placement zones that support the specified instance type.

If cluster_name is given and a cluster with the given name exist, return a list with only one entry where the given cluster lives.

Searching for available zones for a given instance type is done by checking the spot prices in the potential availability zones for support before deciding on a region: http://blog.piefox.com/2011/07/ec2-availability-zones-and-instance.html

Note that, currently, instance-type based zone selection applies only to AWS. For other clouds, all the available zones are returned (unless a cluster is being recreated, in which case the cluster’s placement zone is returned as stored in its persistent data.
Return type  dict

Returns  A dictionary with zones and error keywords.

Changed in version 0.3: Changed method name from _find_placements to find_placements. Also added cluster_name parameter.

Changed in version 0.7.0: The return value changed from a list to a dictionary.

get_cluster_pd(cluster_name)

Return persistent data (as a dict) associated with a cluster with the given cluster_name. If a cluster with the given name is not found, return an empty dict.

New in version 0.3.

get_clusters_pd(include_placement=True)

Return persistent data of all existing clusters for this account.

Parameters include_placement (bool) – Whether or not to include region placement for the clusters. Setting this option will lead to a longer function runtime.

Return type  dict

Returns  A dictionary containing keys clusters and error. The value of clusters will be a dictionary with the following keys cluster_name, persistent_data, bucket_name and optionally placement or an empty list if no clusters were found or an error was encountered. persistent_data key value is yet another dictionary containing given cluster’s persistent data. The value for the error key will contain a string with the error message.

New in version 0.3.

Changed in version 0.7.0: The return value changed from a list to a dictionary.

get_status(instance_id)

Check on the status of an instance. instance_id needs to be a boto-library compatible instance ID (e.g., i-8fehrdss). If instance_id is not provided, the ID obtained when launching the most recent instance is used. Note that this assumes the instance being checked on was launched using this class. Also note that the same class may be used to launch multiple instances but only the most recent instance_id is kept while any others will to be explicitly specified.

This method also allows the required ec2_conn connection object to be provided at invocation time. If the object is not provided, credentials defined for the class are used (ability to specify a custom ec2_conn helps in case of stateless method invocations).

Return a state dict containing the following keys: instance_state, public_ip, placement, and error, which capture CloudMan’s current state. For instance_state, expected values are: pending, booting, running, or error and represent the state of the underlying instance. Other keys will return an empty value until the instance_state enters running state.

launch (cluster_name, image_id, instance_type, password, kernel_id=None, ramdisk_id=None,
key_name='cloudman_key_pair', security_groups=['CloudMan'], placement='', subnet_id=None, ebs_optimized=False, **kwargs)

Check all the prerequisites (key pair and security groups) for launching a CloudMan instance, compose the user data based on the parameters specified in the arguments and the cloud properties as defined in the object’s cloud field.

For the current list of user data fields that can be provided via kwargs, see https://galaxyproject.org/cloudman/userdata/.

Return a dict containing the properties and info with which an instance was launched, namely: sg_names containing the names of the security groups, kp_name containing the name of the key pair, kp_material containing the private portion of the key pair (note that this portion of the key is available

5.1. CloudMan API
and can be retrieved only at the time the key is created, which will happen only if no key with the name provided in the key_name argument exists), rs containing the boto ResultSet object, instance_id containing the ID of a started instance, and error containing an error message if there was one.

**rule_exists** *(rules, from_port, to_port, ip_protocol='tcp', cidr_ip='0.0.0.0/0')*

A convenience method to check if an authorization rule in a security group already exists.

---

**CloudManInstance**

API for interacting with a CloudMan instance.

**class** `bioblend.cloudman.CloudManConfig` *(access_key=None, secret_key=None, cluster_name=None, image_id=None, instance_type='m1.medium', password=None, cloud_metadata=None, cluster_type=None, galaxy_data_option='', initial_storage_size=10, key_name='cloudman_key_pair', security_groups=['CloudMan'], placement='', kernel_id=None, ramdisk_id=None, block_until_ready=False, **kwargs)*

Initializes a CloudMan launch configuration object.

**Parameters**

- **access_key**(str) – Access credentials.
- **secret_key**(str) – Access credentials.
- **cluster_name**(str) – Name used to identify this CloudMan cluster.
- **image_id**(str) – Machine image ID to use when launching this CloudMan instance.
- **instance_type**(str) – The type of the machine instance, as understood by the chosen cloud provider. (e.g., m1.medium)
- **password**(str) – The administrative password for this CloudMan instance.
- **cloud_metadata**(Bunch) – This object must define the properties required to establish a boto connection to that cloud. See this method’s implementation for an example of the required fields. Note that as long as the as provided object defines the required fields, it can really be implemented as anything (e.g., a Bunch, a database object, a custom class). If no value for the cloud argument is provided, the default is to use the Amazon cloud.
- **kernel_id**(str) – The ID of the kernel with which to launch the instances
- **ramdisk_id**(str) – The ID of the RAM disk with which to launch the instances
- **key_name**(str) – The name of the key pair with which to launch instances
- **security_groups**(list of str) – The IDs of the security groups with which to associate instances
- **placement**(str) – The availability zone in which to launch the instances
- **cluster_type**(str) – The type, either ‘Galaxy’, ‘Data’, or ‘Test’, defines the type of cluster platform to initialize.
- **galaxy_data_option**(str) – The storage type to use for this instance. May be ‘transient’, ‘custom_size’ or ’’. The default is ’’, which will result in ignoring the bioblend specified initial_storage_size. ‘custom_size’ must be used for initial_storage_size to come into effect.
• **initial_storage_size** (*int*) – The initial storage to allocate for the instance. This only applies if `cluster_type` is set to either Galaxy or Data and `galaxy_data_option` is set to `custom_size`.

• **block_until_ready** (*bool*) – Specifies whether the launch method will block until the instance is ready and only return once all initialization is complete. The default is False. If False, the launch method will return immediately without blocking. However, any subsequent calls made will automatically block if the instance is not ready and initialized. The blocking timeout and polling interval can be configured by providing extra parameters to the `CloudManInstance.launch_instance` method.

### static `CustomTypeDecoder` (*dct*)

### class `CustomTypeEncoder` (*skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, encoding='utf-8', default=None*)

Constructor for JSONEncoder, with sensible defaults.

If `skipkeys` is false, then it is a TypeError to attempt encoding of keys that are not str, int, long, float or None. If `skipkeys` is True, such items are simply skipped.

If `ensure_ascii` is true (the default), all non-ASCII characters in the output are escaped with uXXXX sequences, and the results are str instances consisting of ASCII characters only. If `ensure_ascii` is False, a result may be a unicode instance. This usually happens if the input contains unicode strings or the `encoding` parameter is used.

If `check_circular` is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an OverflowError). Otherwise, no such check takes place.

If `allow_nan` is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If `sort_keys` is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation. Since the default item separator is ‘,’ ‘’, the output might include trailing whitespace when indent is specified. You can use separators=('‘’, ‘:’) to avoid this.

If specified, separators should be a `(item_separator, key_separator)` tuple. The default is (‘‘, ‘:’). To get the most compact JSON representation you should specify (‘‘, ‘:’) to eliminate whitespace.

If specified, default is a function that gets called for objects that can’t otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

If `encoding` is not None, then all input strings will be transformed into unicode using that encoding prior to JSON-encoding. The default is UTF-8.

### default (*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```python
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
```
pass
else:
    return list(iterable)
# Let the base class default method raise the TypeError
return JSONEncoder.default(self, o)

static load_config(fp)

save_config(fp)

set_connection_parameters(access_key, secret_key, cloud_metadata=None)

set_extra_parameters(**kwargs)

set_post_launch_parameters(cluster_type=None, galaxy_data_option="", initial_storage_size=10)

set_pre_launch_parameters(cluster_name, image_id, instance_type, password, kernel_id=None, ramdisk_id=None, key_name='cloudman_key_pair', security_groups=['CloudMan'], initial_storage_size=10)

validate()

class bioblend.cloudman.CloudManInstance(url, password, **kwargs)
Create an instance of the CloudMan API class, which is to be used when manipulating that given CloudMan instance.

The url is a string defining the address of CloudMan, for example "http://115.146.92.174". The password is CloudMan’s password, as defined in the user data sent to CloudMan on instance creation.

add_nodes(**kwargs)
Add a number of worker nodes to the cluster, optionally specifying the type for new instances. If instance_type is not specified, instance(s) of the same type as the master instance will be started. Note that the instance_type must match the type of instance available on the given cloud.

spot_price applies only to AWS and, if set, defines the maximum price for Spot instances, thus turning this request for more instances into a Spot request.

adjust_autoscaling(**kwargs)
Adjust the autoscaling configuration parameters.

The number of worker nodes in the cluster is bounded by the optional minimum_nodes and maximum_nodes parameters. If a parameter is not provided then its configuration value does not change.

autoscaling_enabled(**kwargs)
Returns a boolean indicating whether autoscaling is enabled.

cloudman_url
Returns the URL for accessing this instance of CloudMan.

disable_autoscaling(**kwargs)
Disable autoscaling, meaning that worker nodes will need to be manually added and removed.

enable_autoscaling(**kwargs)
Enable cluster autoscaling, allowing the cluster to automatically add, or remove, worker nodes, as needed.

The number of worker nodes in the cluster is bounded by the minimum_nodes (default is 0) and maximum_nodes (default is 19) parameters.
**galaxy_url**

Returns the base URL for this instance, which by default happens to be the URL for Galaxy application.

**get_cloudman_version(**kwargs)**

Returns the cloudman version from the server. Versions prior to Cloudman 2 does not support this call, and therefore, the default is to return 1

**get_cluster_size(**kwargs)**

Get the size of the cluster in terms of the number of nodes; this count includes the master node.

**get_cluster_type(**kwargs)**

Get the cluster type for this CloudMan instance. See the CloudMan docs about the available types. Returns a dictionary, for example: `{u'cluster_type': u'Test'}`.

**get_galaxy_state(**kwargs)**

Get the current status of Galaxy running on the cluster.

**get_master_id(**kwargs)**

Returns the instance ID of the master node in this CloudMan cluster

**get_master_ip(**kwargs)**

Returns the public IP of the master node in this CloudMan cluster

**get_nodes(**kwargs)**

Get a list of nodes currently running in this CloudMan cluster.

**get_static_state(**kwargs)**

Get static information on this CloudMan instance. i.e. state that doesn’t change over the lifetime of the cluster

**get_status(**kwargs)**

Get status information on this CloudMan instance.

**initialize(**kwargs)**

Initialize CloudMan platform. This needs to be done before the cluster can be used.

The `cluster_type`, either ‘Galaxy’, ‘Data’, or ‘Test’, defines the type of cluster platform to initialize.

**is_master_execution_host(**kwargs)**

Checks whether the master node has job execution enabled.

**static_launch_instance**(cfg, **kwargs)**

Launches a new instance of CloudMan on the specified cloud infrastructure.

Parameters

- **cfg** (CloudManConfig) – A CloudManConfig object containing the initial parameters for this launch.

**reboot_node(**kwargs)**

Reboot a specific worker node.

The `instance_id` parameter defines the ID, as a string, of a worker node to reboot.

**remove_node(**kwargs)**

Remove a specific worker node from the cluster.

The `instance_id` parameter defines the ID, as a string, of a worker node to remove from the cluster. The `force` parameter (defaulting to False), is a boolean indicating whether the node should be forcibly removed rather than gracefully removed.

**remove_nodes(**kwargs)**

Remove worker nodes from the cluster.
The `num_nodes` parameter defines the number of worker nodes to remove. The `force` parameter (defaulting to False), is a boolean indicating whether the nodes should be forcibly removed rather than gracefully removed.

**set_master_as_execution_host(**kwargs)\n---
Enables/disables master as execution host.

**terminate(**kwargs)\n---
Terminates this CloudMan cluster. There is an option to also terminate the master instance (all worker instances will be terminated in the process of cluster termination), and delete the whole cluster.

**Warning:** Deleting a cluster is irreversible - all of the data will be permanently deleted.

**update()**\n---
Updates the local object’s fields to be in sync with the actual state of the CloudMan instance the object points to. This method should be called periodically to ensure you are looking at the current data.

New in version 0.2.2.

**class** bioblend.cloudman.GenericVMInstance(launcher, launch_result)\n---
Create an instance of the CloudMan API class, which is to be used when manipulating that given CloudMan instance.

The `url` is a string defining the address of CloudMan, for example “http://115.146.92.174”. The `password` is CloudMan’s password, as defined in the user data sent to CloudMan on instance creation.

**get_machine_status()**\n---
Check on the underlying VM status of an instance. This can be used to determine whether the VM has finished booting up and if CloudMan is up and running.

Return a `state dict` with the current `instance_state`, `public_ip`, `placement`, and `error` keys, which capture the current state (the values for those keys default to empty string if no data is available from the cloud).

**instance_id**\n---
Returns the ID of this instance (e.g., i-87ey32dd) if launch was successful or `None` otherwise.

**key_pair_material**\n---
Returns the private portion of the generated key pair. It does so only if the instance was properly launched and key pair generated; `None` otherwise.

**key_pair_name**\n---
Returns the name of the key pair used by this instance. If instance was not launched properly, returns `None`.

**wait_until_instance_ready**\n---
Wait until the VM state changes to ready/error or timeout elapses. Updates the host name once ready.

**exception** bioblend.cloudman.VMLaunchException(value)\n---

**bioblend.cloudman.block_until_vm_ready**\n---
This decorator exists to make sure that a launched VM is ready and has received a public IP before allowing the wrapped function call to continue. If the VM is not ready, the function will block until the VM is ready. If the VM does not become ready until the `vm_ready_timeout` elapses or the VM status returns an error, a VMLaunchException will be thrown.

This decorator relies on the `wait_until_instance_ready` method defined in class GenericVMInstance. All methods to which this decorator is applied must be members of a class which inherit from GenericVMInstance.

The following two optional keyword arguments are recognized by this decorator:
Parameters

- **vm_ready_timeout** \((\text{int})\) – Maximum length of time to block before timing out. Once the timeout is reached, a VMLaunchException will be thrown.

- **vm_ready_check_interval** \((\text{int})\) – The number of seconds to pause between consecutive calls when polling the VM’s ready status.

### 5.1.2 Usage documentation

This page describes some sample use cases for CloudMan API and provides examples for these API calls. In addition to this page, there are functional examples of complete scripts in `docs/examples` directory of the BioBlend source code repository.

#### Setting up custom cloud properties

CloudMan supports Amazon, OpenStack, OpenNebula, and Eucalyptus based clouds and BioBlend can be used to programatically manipulate CloudMan on any of those clouds. Once launched, the API calls to CloudMan are the same irrespective of the cloud. In order to launch an instance on a given cloud, cloud properties need to be provided to `CloudManLauncher`. If cloud properties are not specified, `CloudManLauncher` will default to Amazon cloud properties.

If we want to use a different cloud provider, we need to specify additional cloud properties when creating an instance of the `CloudManLauncher` class. For example, if we wanted to create a connection to NeCTAR, Australia’s national research cloud, we would use the following properties:

```python
from bioblend.util import Bunch
nectar = Bunch(
    name='NeCTAR',
    cloud_type='openstack',
    bucket_default='cloudman-os',
    region_name='NeCTAR',
    region_endpoint='nova.rc.nectar.org.au',
    ec2_port=8773,
    ec2_conn_path='/services/Cloud',
    cidr_range='115.146.92.0/22',
    is_secure=True,
    s3_host='swift.rc.nectar.org.au',
    s3_port=8888,
    s3_conn_path='/')
```

**Note:** These properties are cloud-specific and need to be obtained from a given cloud provider.

#### Launching a new cluster instance

In order to launch a CloudMan cluster on a chosen cloud, we do the following (continuing from the previous example):

```python
from bioblend.cloudman import CloudManConfig
from bioblend.cloudman import CloudManInstance
cmc = CloudManConfig(‘<your AWS access key’, ‘your AWS secret key’, ‘Cluster name’, ‘ami-<ID>’, ‘m1.medium’, ‘choose_a_password_here’, nectar)
cmi = CloudManInstance.launch_instance(cmc)
```
Note: If you already have an existing instance of CloudMan, just create an instance of the `CloudManInstance` object directly by calling its constructor and connecting to it (the password you provide must match the password you provided as part of user data when launching this instance). For example:

```python
cmi = CloudManInstance('http://115.146.92.174', 'your_UD_password')
```

We now have a `CloudManInstance` object that allows us to manage created CloudMan instance via the API. Once launched, it will take a few minutes for the instance to boot and CloudMan start. To check on the status of the machine, (repeatedly) run the following command:

```python
>>> cmi.get_machine_status()
{'error': '',
 'instance_state': u'pending',
 'placement': '',
 'public_ip': ''}
>>> cmi.get_machine_status()
{'error': '',
 'instance_state': u'running',
 'placement': u'melbourne-qh2',
 'public_ip': u'115.146.86.29'}
```

Once the instance is ready, although it may still take a few moments for CloudMan to start, it is possible to start interacting with the application.

Note: The `CloudManInstance` object (e.g., `cmi`) is a local representation of the actual CloudMan instance. As a result, the local object can get out of sync with the remote instance. To update the state of the local object, call the `update` method on the `cmi` object:

```python
>>> cmi.update()
```

**Manipulating an existing cluster**

Having a reference to a `CloudManInstance` object, we can manage it via the available `CloudManInstance` API:

```python
>>> cmi.initialized
False
>>> cmi.initialize('SGE')
>>> cmi.get_status()
{'all_fs': [],
 'app_status': u'yellow',
 'autoscaling': {u'as_max': u'N/A',
 u'as_min': u'N/A',
 u'use_autoscaling': False},
 'cluster_status': u'STARTING',
 'data_status': u'green',
 'disk_usage': {u'pct': u'0%', u'total': u'0', u'used': u'0'},
 u'dns': u'#',
 'instance_status': {u'available': u'0', u'idle': u'0', u'requested': u'0'},
 'snapshot': {u'progress': u'None', u'status': u'None'}
>>> cmi.get_cluster_size()
1
>>> cmi.get_nodes()
```
5.2 Galaxy API

API used to manipulate genomic analyses within Galaxy, including data management and workflow execution.

5.2.1 API documentation for interacting with Galaxy

**GalaxyInstance**

class bioblend.galaxy.GalaxyInstance(url, key=None, email=None, password=None, verify=True)

A base representation of a connection to a Galaxy instance, identified by the server URL and user credentials. 

After you have created a GalaxyInstance object, access various modules via the class fields. For example, to work with histories and get a list of all the user’s histories, the following should be done:

```python
from bioblend import galaxy

gi = galaxy.GalaxyInstance(url='http://127.0.0.1:8000', key='your_api_key')
hl = gi.histories.get_histories()
```

**Parameters**

- **url (str)** – A FQDN or IP for a given instance of Galaxy. For example: http://127.0.0.1:8080. If a Galaxy instance is served under a prefix (e.g., http://127.0.0.1:8080/galaxy/), supply the entire URL including the prefix (note that the prefix must end with a slash).
- **key (str)** – User’s API key for the given instance of Galaxy, obtained from the user preferences. If a key is not supplied, an email address and password must be and the key will automatically be created for the user.
- **email (str)** – Galaxy e-mail address corresponding to the user. Ignored if key is supplied directly.
• **password** *(str)* – Password of Galaxy account corresponding to the above e-mail address. Ignored if key is supplied directly.

• **verify** *(bool)* – Whether to verify the server’s TLS certificate

```python
__init__(url, key=None, email=None, password=None, verify=True)
```

A base representation of a connection to a Galaxy instance, identified by the server URL and user credentials.

After you have created a `GalaxyInstance` object, access various modules via the class fields. For example, to work with histories and get a list of all the user’s histories, the following should be done:

```python
from bioblend import galaxy

gi = galaxy.GalaxyInstance(url='http://127.0.0.1:8000', key='your_api_key')

hl = gi.histories.get_histories()
```

**Parameters**

• **url** *(str)* – A FQDN or IP for a given instance of Galaxy. For example: `http://127.0.0.1:8080`. If a Galaxy instance is served under a prefix (e.g., `http://127.0.0.1:8080/galaxy/`), supply the entire URL including the prefix (note that the prefix must end with a slash).

• **key** *(str)* – User’s API key for the given instance of Galaxy, obtained from the user preferences. If a key is not supplied, an email address and password must be and the key will automatically be created for the user.

• **email** *(str)* – Galaxy e-mail address corresponding to the user. Ignored if key is supplied directly.

• **password** *(str)* – Password of Galaxy account corresponding to the above e-mail address. Ignored if key is supplied directly.

• **verify** *(bool)* – Whether to verify the server’s TLS certificate

```python
get_retry_delay
max_get_attempts
```

**Config**

Contains possible interaction dealing with Galaxy configuration.

```python
class bioblend.galaxy.config.ConfigClient(galaxy_instance)
```

```python
get_config()

Get a list of attributes about the Galaxy instance. More attributes will be present if the user is an admin.

Return type  list

Returns

A list of attributes. For example:
get_version()

Get the current version of the Galaxy instance. This functionality is available since Galaxy release_15.03.

Return type dict

Returns Version of the Galaxy instance

For example:

```python
{'extra': {}, 'version_major': '17.01'}
```

Datasets

Contains possible interactions with the Galaxy Datasets

class bioblend.galaxy.datasets.DatasetClient(galaxy_instance)

download_dataset (dataset_id, file_path=None, use_default_filename=True, maxwait=12000)

Download a dataset to file or in memory. If the dataset state is not ‘ok’, a DatasetStateException will be thrown.

Parameters

- **dataset_id** (str) – Encoded dataset ID
- **file_path** (str) – If this argument is provided, the dataset will be streamed to disk at that path (should be a directory if use_default_filename=True). If the file_path argument is not provided, the dataset content is loaded into memory and returned by the method (Memory consumption may be heavy as the entire file will be in memory).
- **use_default_filename** (bool) – If True, the exported file will be saved as file_path/%s, where %s is the dataset name. If False, file_path is assumed to contain the full file path including the filename.
- **maxwait** (float) – Total time (in seconds) to wait for the dataset state to become terminal. If the dataset state is not terminal within this time, a DatasetTimeoutException will be thrown.

Return type dict

Returns If a file_path argument is not provided, returns a dict containing the file content. Otherwise returns nothing.
show_dataset (dataset_id, deleted=False, hda_ldda='hda')
Get details about a given dataset. This can be a history or a library dataset.

Parameters

- **dataset_id** (str) – Encoded dataset ID
- **deleted** (bool) – Whether to return results for a deleted dataset
- **hda_ldda** (str) – Whether to show a history dataset ('hda' - the default) or library dataset ('ldda').

Return type  dict

Returns  Information about the HDA or LDDA

show_stderr (dataset_id)
Get the stderr output of a dataset.

Deprecated since version 0.9.0: Use show_job() with full_details=True instead.

Parameters  **dataset_id** (str) – Encoded dataset ID

show_stdout (dataset_id)
Get the stdout output of a dataset.

Deprecated since version 0.9.0: Use show_job() with full_details=True instead.

Parameters  **dataset_id** (str) – Encoded dataset ID

exception  bioblend.galaxy.datasets.DatasetStateException
exception  bioblend.galaxy.datasets.DatasetTimeoutException

Datatypes
Contains possible interactions with the Galaxy Datatype

class  bioblend.galaxy.datatypes.DatatypesClient (galaxy_instance)

going_datatypes (extension_only=False, upload_only=False)
Get the list of all installed datatypes.

Parameters

- **extension_only** (bool) – Return only the extension rather than the datatype name
- **upload_only** (bool) – Whether to return only datatypes which can be uploaded

Return type  list

Returns  A list of datatype names. For example:

```
[u'snpmatrix',
 u'snptest',
 u'tabular',
 u'taxonomy',
 u'twobit',
 u'txt',
]
```

(continues on next page)
get_sniffers()
    Get the list of all installed sniffers.

    Return type  list

    Returns
    A list of sniffer names. For example:

    [u'galaxy.datatypes.tabular:Vcf',
     u'galaxy.datatypes.binary:TwoBit',
     u'galaxy.datatypes.binary:Bam',
     u'galaxy.datatypes.binary:Sff',
     u'galaxy.datatypes.xml:Phyloxml',
     u'galaxy.datatypes.xml:GenericXml',
     u'galaxy.datatypes.sequence:Maf',
     u'galaxy.datatypes.sequence:Lav',
     u'galaxy.datatypes.sequence:csFasta']

Folders
Contains possible interactions with the Galaxy library folders

class bioblend.galaxy.folders.FoldersClient(galaxy_instance)

create_folder(parent_folder_id, name, description=None)
    Create a folder.

    Parameters
    • parent_folder_id (str) – Folder’s description
    • name (str) – name of the new folder
    • description (str) – folder’s description

    Return type  dict

    Returns  details of the updated folder

delete_folder(folder_id, undelete=False)
    Marks the folder with the given id as deleted (or removes the deleted mark if the undelete param is True).

    Parameters
    • folder_id (str) – the folder’s encoded id, prefixed by ‘F’
    • undelete (bool) – If set to True, the folder will be undeleted (i.e. the deleted mark will be removed)

    Returns  detailed folder information

    Return type  dict
get_permissions (folder_id, scope)
Get the permissions of a folder.

Parameters

- folder_id (str) – the folder's encoded id, prefixed by 'F'
- scope (str) – scope of permissions, either 'current' or 'available'

Return type: dict

Returns: dictionary including details of the folder

set_permissions (folder_id, action='set_permissions', add_ids=None, manage_ids=None, modify_ids=None)
Set the permissions of a folder.

Parameters

- folder_id (str) – the folder's encoded id, prefixed by 'F'
- action (str) – action to execute, only "set_permissions" is supported.
- add_ids (list of str) – list of role IDs which can add datasets to the folder
- manage_ids (list of str) – list of role IDs which can manage datasets in the folder
- modify_ids (list of str) – list of role IDs which can modify datasets in the folder

Return type: dict

Returns: dictionary including details of the folder

show_folder (folder_id, contents=False)
Display information about a folder.

Parameters

- folder_id (str) – the folder's encoded id, prefixed by 'F'
- contents (bool) – True to get the contents of the folder, rather than just the folder details.

Return type: dict

Returns: dictionary including details of the folder

update_folder (folder_id, name, description=None)
Update folder information.

Parameters

- folder_id (str) – the folder's encoded id, prefixed by 'F'
- name (str) – name of the new folder
- description (str) – folder's description

Return type: dict

Returns: details of the updated folder
Forms

Contains possible interactions with the Galaxy Forms

class bioblend.galaxy.forms.FormsClient(galaxy_instance)

create_form(form_xml_text)
Create a new form.

Parameters form_xml_text (str) – Form xml to create a form on galaxy instance

Return type str

Returns Unique URL of newly created form with encoded id

get_forms()
Get the list of all forms.

Return type list

Returns
Displays a collection (list) of forms. For example:

```
[{u'id': u'f2db41e1fa331b3e',
  u'model_class': u'FormDefinition',
  u'name': u'First form',
  u'url': u'/api/forms/f2db41e1fa331b3e'},
 {u'id': u'ebfb8f50c6abde6d',
  u'model_class': u'FormDefinition',
  u'name': u'second form',
  u'url': u'/api/forms/ebfb8f50c6abde6d'}]
```

show_form(form_id)
Get details of a given form.

Parameters form_id (str) – Encoded form ID

Return type dict

Returns
A description of the given form. For example:

```
{u'desc': u'here it is ',
 u'fields': [],
 u'form_definition_current_id': u'f2db41e1fa331b3e',
 u'id': u'f2db41e1fa331b3e',
 u'layout': [],
 u'model_class': u'FormDefinition',
 u'name': u'First form',
 u'url': u'/api/forms/f2db41e1fa331b3e'}
```

FTP files

Contains possible interactions with the Galaxy FTP Files

class bioblend.galaxy.ftpfiles.FTPFilesClient(galaxy_instance)
**get_ftp_files** *(deleted=False)*
Get a list of local files.

**Parameters**
- **deleted** *(bool)* – Whether to include deleted files

**Return type** list

**Returns** A list of dicts with details on individual files on FTP

---

### Genomes

Contains possible interactions with the Galaxy Histories

**class** **bioblend.galaxy.genomes.GenomeClient** *(galaxy_instance)*

**get_genomes** *
Returns a list of installed genomes

**Return type** list

**Returns** List of installed genomes

**install_genome** *(func='download', source=None, dbkey=None, ncbi_name=None, ensembl_dbkey=None, url_dbkey=None, indexers=None)*
Download and/or index a genome.

**Parameters**
- **dbkey** *(str)* – DB key of the build to download, ignored unless ‘UCSC’ is specified as the source
- **ncbi_name** *(str)* – NCBI’s genome identifier, ignored unless NCBI is specified as the source
- **ensembl_dbkey** *(str)* – Ensembl’s genome identifier, ignored unless Ensembl is specified as the source
- **url_dbkey** *(str)* – DB key to use for this build, ignored unless URL is specified as the source
- **source** *(str)* – Data source for this build. Can be: UCSC, Ensembl, NCBI, URL
- **indexers** *(list)* – POST array of indexers to run after downloading (indexers[] = first, indexers[] = second, ...)
- **func** *(str)* – Allowed values: ‘download’, Download and index; ‘index’, Index only

**Return type** dict


**show_genome** *(id, num=None, chrom=None, low=None, high=None)*
Returns information about build <id>

**Parameters**
- **id** *(str)* – Genome build ID to use
- **num** *(str)* – num
- **chrom** *(str)* – chrom
- **low** *(str)* – low
• **high** *(str)* – high

**Return type**  dict

**Returns**  Information about the genome build

### Groups

Contains possible interactions with the Galaxy Groups

```python
class bioblend.galaxy.groups.GroupsClient(galaxy_instance)
```

**add_group_role** *(group_id, role_id)*

Add a role to the given group.

**Parameters**

- **group_id** *(str)* – Encoded group ID
- **role_id** *(str)* – Encoded role ID to add to the group

**Return type**  dict

**Returns**  Added group role’s info

**add_group_user** *(group_id, user_id)*

Add a user to the given group.

**Parameters**

- **group_id** *(str)* – Encoded group ID
- **user_id** *(str)* – Encoded user ID to add to the group

**Return type**  dict

**Returns**  Added group user’s info

**create_group** *(group_name, user_ids=[], role_ids=[])*

Create a new group.

**Parameters**

- **group_name** *(str)* – A name for the new group
- **user_ids** *(list)* – A list of encoded user IDs to add to the new group
- **role_ids** *(list)* – A list of encoded role IDs to add to the new group

**Return type**  list

**Returns**

A (size 1) list with newly created group details, like:

```python
[{u'id': u'7c9636938c3e83bf',
  u'model_class': u'Group',
  u'name': u'My Group Name',
  u'url': u'/api/groups/7c9636938c3e83bf'}]
```

**delete_group_role** *(group_id, role_id)*

Remove a role from the given group.

**Parameters**
• \texttt{group\_id (str)} – Encoded group ID
• \texttt{role\_id (str)} – Encoded role ID to remove from the group

Return type dict

Returns The role which was removed

\textbf{delete\_group\_user (group\_id, user\_id)}

Remove a user from the given group.

Parameters
• \texttt{group\_id (str)} – Encoded group ID
• \texttt{user\_id (str)} – Encoded user ID to remove from the group

Return type dict

Returns The user which was removed

\textbf{get\_group\_roles (group\_id)}

Get the list of roles associated to the given group.

Parameters \texttt{group\_id (str)} – Encoded group ID

Return type list of dicts

Returns List of group roles’ info

\textbf{get\_group\_users (group\_id)}

Get the list of users associated to the given group.

Parameters \texttt{group\_id (str)} – Encoded group ID

Return type list of dicts

Returns List of group users’ info

\textbf{get\_groups ()}

Get all (not deleted) groups.

Return type list

Returns A list of dicts with details on individual groups. For example:

```
[{'id': '33abac023ff186c2',
'\texttt{model\_class}': \texttt{\'Group\',
'\texttt{name}': \texttt{\'Listeria\',
'\texttt{url}': '/api/groups/33abac023ff186c2'},
{'id': '73187219cd372cf8',
'\texttt{model\_class}': \texttt{\'Group\',
'\texttt{name}': \texttt{\'LPN\',
'\texttt{url}': '/api/groups/73187219cd372cf8'}}]
```

\textbf{show\_group (group\_id)}

Get details of a given group.

Parameters \texttt{group\_id (str)} – Encoded group ID

Return type dict

Returns A description of group For example:
update_group (group_id, group_name=None, user_ids=[], role_ids=[])  
Update a group.

Parameters

• **group_id** *(str)* – Encoded group ID

• **group_name** *(str)* – A new name for the group. If None, the group name is not changed.

• **user_ids** *(list)* – New list of encoded user IDs for the group. It will substitute the previous list of users (with [] if not specified)

• **role_ids** *(list)* – New list of encoded role IDs for the group. It will substitute the previous list of roles (with [] if not specified)

Return type  None

Returns  None

Histories

Contains possible interactions with the Galaxy Histories

**class**  `bioblend.galaxy.histories.HistoryClient (galaxy_instance)`

**create_dataset_collection** *(history_id, collection_description)*

Create a new dataset collection

Parameters

• **history_id** *(str)* – Encoded history ID

• **collection_description**  *(bioblend.galaxy.dataset_collections.CollectionDescription)* – a description of the dataset collection For example:

```json
{
'collection_type': 'list',
'element_identifiers': 
[{
'id': 'f792763bee8d277a',
'name': 'element 1',
'src': 'hda'},
{
'id': 'f792763bee8d277a',
'name': 'element 2',
'src': 'hda'}],
'name': 'My collection list'
}
```

Return type  dict

Returns  Information about the new HDCA

**create_history** *(name=None)*

Create a new history, optionally setting the name.
Parameters name (str) – Optional name for new history
Return type dict
Returns Dictionary containing information about newly created history

create_history_tag (history_id, tag)
Create history tag

Parameters
• history_id (str) – Encoded history ID
• tag (str) – Add tag to history
Return type dict
Returns A dictionary with information regarding the tag. For example:

```python
{'id': 'f792763bee8d277a',
'model_class': 'HistoryTagAssociation',
'user_tname': 'NGS_PE_RUN',
'user_value': None}
```

delete_dataset (history_id, dataset_id, purge=False)
Mark corresponding dataset as deleted.

Parameters
• history_id (str) – Encoded history ID
• dataset_id (str) – Encoded dataset ID
• purge (bool) – if True, also purge (permanently delete) the dataset
Return type None
Returns None

Note: For the purge option to work, the Galaxy instance must have the allow_user_dataset_purge option set to True in the config/galaxy.ini configuration file.

Warning: If you purge a dataset which has not been previously deleted, Galaxy since release_15.03 wrongly does not set the deleted attribute of the dataset to True, see https://github.com/galaxyproject/galaxy/issues/3548

delete_dataset_collection (history_id, dataset_collection_id)
Mark corresponding dataset collection as deleted.

Parameters
• history_id (str) – Encoded history ID
• dataset_collection_id (str) – Encoded dataset collection ID
Return type None
Returns None
**delete_history** *(history_id, purge=False)*

Delete a history.

**Parameters**

- **history_id**(str) – Encoded history ID
- **purge**(bool) – if True, also purge (permanently delete) the history

**Return type** dict

**Returns** An error object if an error occurred or a dictionary containing: id (the encoded id of the history), deleted (if the history was marked as deleted), purged (if the history was purged).

**Note:** For the purge option to work, the Galaxy instance must have the allow_user_dataset_purge option set to True in the config/galaxy.ini configuration file.

**download_dataset** *(history_id, dataset_id, file_path, use_default_filename=True)*

Deprecated since version 0.8.0: Use `download_dataset()` instead.

**download_history** *(history_id, jeha_id, outf, chunk_size=4096)*

Download a history export archive. Use `export_history()` to create an export.

**Parameters**

- **history_id**(str) – history ID
- **jeha_id**(str) – jeha ID (this should be obtained via `export_history()`)
- **outf**(file) – output file object, open for writing in binary mode
- **chunk_size**(int) – how many bytes at a time should be read into memory

**Return type** None

**Returns** None

**export_history** *(history_id, gzip=True, include_hidden=False, include_deleted=False, wait=False, maxwait=None)*

Start a job to create an export archive for the given history.

**Parameters**

- **history_id**(str) – history ID
- **gzip**(bool) – create .tar.gz archive if True, else .tar
- **include_hidden**(bool) – whether to include hidden datasets in the export
- **include_deleted**(bool) – whether to include deleted datasets in the export
- **wait**(bool) – if True, block until the export is ready; else, return immediately
- **maxwait**(float) – Total time (in seconds) to wait for the export to become ready. When set, implies that wait is True.

**Return type** str

**Returns** jeha_id of the export, or empty if wait is False and the export is not ready.

**get_current_history** ()

Deprecated since version 0.5.2: Use `get_most_recently_used_history()` instead.
get_histories (history_id=None, name=None, deleted=False)
Get all histories or filter the specific one(s) via the provided name or history_id. Provide only one argument, name or history_id, but not both.

If deleted is set to True, return histories that have been deleted.

Parameters
- **history_id**(str) – Encoded history ID to filter on
- **name**(str) – Name of history to filter on
- **deleted**(bool) – whether to filter for the deleted histories (True) or for the non-deleted ones (False)

Return type list

Returns Return a list of history element dicts. If more than one history matches the given name, return the list of all the histories with the given name

get_most_recently_used_history()
Returns the current user’s most recently used history (not deleted).

Return type dict

Returns History representation

get_status (history_id)
Returns the state of this history

Parameters **history_id**(str) – Encoded history ID

Return type dict

Returns A dict documenting the current state of the history. Has the following keys: ‘state’ = This is the current state of the history, such as ok, error, new etc. ‘state_details’ = Contains individual statistics for various dataset states. ‘percent_complete’ = The overall number of datasets processed to completion.

show_dataset (history_id, dataset_id)
Get details about a given history dataset.

Parameters
- **history_id**(str) – Encoded history ID
- **dataset_id**(str) – Encoded dataset ID

Return type dict

Returns Information about the dataset

show_dataset_collection (history_id, dataset_collection_id)
Get details about a given history dataset collection.

Parameters
- **history_id**(str) – Encoded history ID
- **dataset_collection_id**(str) – Encoded dataset collection ID

Return type dict

Returns Information about the dataset collection
show_dataset_provenance (history_id, dataset_id, follow=False)

Get details related to how dataset was created (id, job_id, tool_id, stdout, stderr, parameters, inputs, etc...).

Parameters

- **history_id (str)** – Encoded history ID
- **dataset_id (str)** – Encoded dataset ID
- **follow (bool)** – If True, recursively fetch dataset provenance information for all inputs and their inputs, etc.

Return type: dict

Returns

Dataset provenance information

For example:

```json
{
    "tool_id": "toolshed.g2.bx.psu.edu/repos/ziru-zhou/macs2/modencode_peakcalling_macs2/2.0.10.2",
    "job_id": "5471ba76f274f929",
    "parameters": {
        "input_chipseq_file1": {
            "id": "6f0a311a444290f2",
            "uuid": null
        },
        "dbkey": "mm9",
        "experiment_name": "H3K4me3_TAC_MACS2",
        "input_control_file1": {
            "id": "c21816a9f51c52e",
            "uuid": null
        },
        "major_command": {
            "gsize": "2716965481.0",
            "bdg": "False",
            "__current_case__": 0,
            "advanced_options": {
                "advanced_options_selector": "off",
                "__current_case__": 1
            },
            "callpeak",
            "input_control_file1": "104721",
            "pg_options": {
                "pq_options_selector": "qvalue",
                "qvalue": "0.05",
                "__current_case__": 1
            },
            "bw": "300",
            "nomodel_type": {
                "nomodel_type_selector": "create_model",
                "__current_case__": 1
            }
        },
        "chromInfo": "/usr/local/galaxy/galaxy-dist/tool-data/shared/ucsc/chrom/mm9.len"
    },
    "stdout": "",
    "stderr": "",
    "id": "6fbd9b2274f62ebe",
    "uuid": null
}
```

show_history (history_id, contents=False, deleted=None, visible=None, details=None, types=None)

Get details of a given history. By default, just get the history meta information.

Parameters

- **history_id (str)** – Encoded history ID to filter on
- **contents (bool)** – When True, instead of the history details, return the list of datasets in the given history.
• **deleted** *(bool or None)* – When *contents=True*, whether to filter for the deleted datasets *(True)* or for the non-deleted ones *(False)*. If not set, no filtering is applied.

• **visible** *(bool or None)* – When *contents=True*, whether to filter for the visible datasets *(True)* or for the hidden ones *(False)*. If not set, no filtering is applied.

• **details** *(str)* – When *contents=True*, include dataset details. Set to ‘all’ for the most information.

• **types** *(list)* – When *contents=True*, filter for history content types. If set to ['dataset'], return only datasets. If set to ['dataset_collection'], return only dataset collections. If not set, no filtering is applied.

**Return type**  dict

**Returns**  details of the given history

**show_matching_datasets**(history_id, name_filter=None)

Get dataset details for matching datasets within a history.

**Parameters**

• **history_id** *(str)* – Encoded history ID

• **name_filter** *(str)* – Only datasets whose name matches the name_filter regular expression will be returned; use plain strings for exact matches and None to match all datasets in the history

**Return type**  list

**Returns**  List of dictionaries

**undelete_history**(history_id)

Undelete a history

**Parameters**  history_id *(str)* – Encoded history ID

**Return type**  str

**Returns**  ‘OK’ if it was deleted

**update_dataset**(history_id, dataset_id, **kwds)

Update history dataset metadata. Some of the attributes that can be modified are documented below.

**Parameters**

• **history_id** *(str)* – Encoded history ID

• **dataset_id** *(str)* – Id of the dataset

• **name** *(str)* – Replace history dataset name with the given string

• **genome_build** *(str)* – Replace history dataset genome build (dbkey)

• **annotation** *(str)* – Replace history dataset annotation with given string

• **deleted** *(bool)* – Mark or unmark history dataset as deleted

• **visible** *(bool)* – Mark or unmark history dataset as visible

**Return type**  dict

**Returns**  details of the updated dataset (for Galaxy release_15.01 and earlier only the updated attributes)
update_dataset_collection(history_id, dataset_collection_id, **kwds)

Update history dataset collection metadata. Some of the attributes that can be modified are documented below.

Parameters

- history_id (str) – Encoded history ID
- dataset_collection_id (str) – Encoded dataset_collection ID
- name (str) – Replace history dataset collection name with the given string
- deleted (bool) – Mark or unmark history dataset collection as deleted
- visible (bool) – Mark or unmark history dataset collection as visible

Return type  dict

Returns  the updated dataset collection attributes

update_history(history_id, **kwds)

Update history metadata information. Some of the attributes that can be modified are documented below.

Parameters

- history_id (str) – Encoded history ID
- name (str) – Replace history name with the given string
- annotation (str) – Replace history annotation with given string
- deleted (bool) – Mark or unmark history as deleted
- purged (bool) – If True, mark history as purged (permanently deleted). Ignored on Galaxy release_15.01 and earlier
- published (bool) – Mark or unmark history as published
- importable (bool) – Mark or unmark history as importable
- tags (list) – Replace history tags with the given list

Return type  dict

Returns  details of the updated history (for Galaxy release_15.01 and earlier only the updated attributes)

upload_dataset_from_library(history_id, lib_dataset_id)

Upload a dataset into the history from a library. Requires the library dataset ID, which can be obtained from the library contents.
Parameters

- **history_id**(str) – Encoded history ID
- **lib_dataset_id**(str) – Encoded library dataset ID

Return type dict

Returns Information about the newly created HDA

Jobs

Contains possible interactions with the Galaxy Jobs

class bioblend.galaxy.jobs.JobsClient(galaxy_instance)

**get_jobs()**

Get the list of jobs of the current user.

**Return type** list

**Returns**

list of dictionaries containing summary job information. For example:

```python
[{u'create_time': u'2014-03-01T16:16:48.640550',
  u'exit_code': 0,
  u'id': u'ebfb8f50c6abde6d',
  u'model_class': u'Job',
  u'state': u'ok',
  u'tool_id': u'fasta2tab',
  u'update_time': u'2014-03-01T16:16:50.657399'},
{u'create_time': u'2014-03-01T16:05:34.851246',
  u'exit_code': 0,
  u'id': u'1cd8e2f6b131e891',
  u'model_class': u'Job',
  u'state': u'ok',
  u'tool_id': u'upload1',
  u'update_time': u'2014-03-01T16:05:39.558458'}]
```

**get_state(job_id)**

Display the current state for a given job of the current user.

**Parameters** **job_id**(str) – job ID

**Return type** str

**Returns** state of the given job among the following values: new, queued, running, waiting, ok. If the state cannot be retrieved, an empty string is returned.

New in version 0.5.3.

**search_jobs(job_info)**

Return jobs for the current user based payload content.

**Parameters** **job_info**(dict) – dictionary containing description of the requested job. This is in the same format as a request to POST /api/tools would take to initiate a job

**Return type** list
Returns list of dictionaries containing summary job information of the jobs that match the requested job run

This method is designed to scan the list of previously run jobs and find records of jobs that had the exact some input parameters and datasets. This can be used to minimize the amount of repeated work, and simply recycle the old results.

**show_job** *(job_id, full_details=False)*

Get details of a given job of the current user.

**Parameters**

- **job_id** *(str)* – job ID
- **full_details** *(bool)* – when True, the complete list of details for the given job.

**Return type** *dict*

**Returns**

A description of the given job. For example:

```
{u'create_time': u'2014-03-01T16:17:29.828624',
 u'exit_code': 0,
 u'id': u'a799d38679e985db',
 u'inputs': {u'input': {u'id': u'ebfb8f50c6abde6d',
 u'src': u'hda'}},
 u'model_class': u'Job',
 u'outputs': {u'output': {u'id': u'a799d38679e985db',
 u'src': u'hda'}},
 u'params': {u'chromInfo': u'"/opt/galaxy-central/tool-data/shared/
 →ucsc/chrom/?.len"',
 u'dbkey': u'"?\"',
 u'seq_col': u'"2\"',
 u'title_col': u'"1\"'},
 u'state': u'ok',
 u'tool_id': u'tab2fasta',
 u'update_time': u'2014-03-01T16:17:31.930728'}
```

**Libraries**

Contains possible interactions with the Galaxy Data Libraries

```python
class bioblend.galaxy.libraries.LibraryClient(galaxy_instance)

    copy_from_dataset(library_id, dataset_id, folder_id=None, message="")

Copy a Galaxy dataset into a library.

**Parameters**

- **library_id** *(str)* – id of the library where to place the uploaded file
- **dataset_id** *(str)* – id of the dataset to copy from
- **folder_id** *(str)* – id of the folder where to place the uploaded files. If not provided, the root folder will be used
- **message** *(str)* – message for copying action

**Return type** *dict*
```
Returns  LDDA information

create_folder (library_id, folder_name, description=None, base_folder_id=None)
Create a folder in a library.

Parameters
• library_id (str) – library id to use
• folder_name (str) – name of the new folder in the data library
• description (str) – description of the new folder in the data library
• base_folder_id (str) – id of the folder where to create the new folder. If not pro-
  vided, the root folder will be used

Return type  list

Returns  List with a single dictionary containing information about the new folder

create_library (name, description=None, synopsis=None)
Create a data library with the properties defined in the arguments.

Parameters
• name (str) – Name of the new data library
• description (str) – Optional data library description
• synopsis (str) – Optional data library synopsis

Return type  dict

Returns  Details of the created library. For example:

```python
{'id': 'f740ab636b360a70',
 'name': 'Library from bioblend',
 'url': '/api/libraries/f740ab636b360a70'}
```

delete_library (library_id)
Delete a data library.

Parameters  library_id (str) – Encoded data library ID identifying the library to be deleted

Return type  dict

Returns  Information about the deleted library

Warning:  Deleting a data library is irreversible - all of the data from the library will be permanently
deleted.

delete_library_dataset (library_id, dataset_id, purged=False)
Delete a library dataset in a data library.

Parameters
• library_id (str) – library id where dataset is found in
• dataset_id (str) – id of the dataset to be deleted
• purged (bool) – Indicate that the dataset should be purged (permanently deleted)

Return type  dict
Returns
A dictionary containing the dataset id and whether the dataset has been deleted. For example:

```python
{u'deleted': True,
 u'id': u'60e680a037f41974'}
```

`get_folders` *(library_id, folder_id=None, name=None)*
Get all the folders or filter specific one(s) via the provided name or folder_id in data library with id library_id. Provide only one argument: name or folder_id, but not both.

**Parameters**
- `library_id` *(str)* – library id to use
- `folder_id` *(str)* – filter for folder by folder id
- `name` *(str)* – filter for folder by name. For name specify the full path of the folder starting from the library’s root folder, e.g. /subfolder/subsubfolder.

**Return type** list

**Returns** list of dicts each containing basic information about a folder

`get_libraries` *(library_id=None, name=None, deleted=False)*
Get all the libraries or filter for specific one(s) via the provided name or ID. Provide only one argument: name or library_id, but not both.

**Parameters**
- `library_id` *(str)* – filter for library by library id
- `name` *(str)* – If name is set and multiple names match the given name, all the libraries matching the argument will be returned
- `deleted` *(bool)* – If False (the default), return only non-deleted libraries. If True, return only deleted libraries. If ‘’None’, return both deleted and non-deleted libraries.

**Return type** list

**Returns** list of dicts each containing basic information about a library

`get_library_permissions` *(library_id)*
Get the permissions for a library.

**Parameters**
- `library_id` *(str)* – id of the library

**Return type** dict

**Returns** dictionary with all applicable permissions’ values

`set_library_permissions` *(library_id, access_in=None, modify_in=None, add_in=None, manage_in=None)*
Set the permissions for a library. Note: it will override all security for this library even if you leave out a permission type.

**Parameters**
- `library_id` *(str)* – id of the library
- `access_in` *(list)* – list of role ids
- `modify_in` *(list)* – list of role ids
- `add_in` *(list)* – list of role ids
- `manage_in` *(list)* – list of role ids
Return type  dict
Returns  General information about the library

**show_dataset** (library_id, dataset_id)
Get details about a given library dataset. The required library_id can be obtained from the datasets’s library content details.

Parameters
- **library_id** (str) – library id where dataset is found in
- **dataset_id** (str) – id of the dataset to be inspected

Return type  dict
Returns  A dictionary containing information about the dataset in the library

**show_folder** (library_id, folder_id)
Get details about a given folder. The required folder_id can be obtained from the folder’s library content details.

Parameters
- **library_id** (str) – library id to inspect folders in
- **folder_id** (str) – id of the folder to be inspected

Return type  dict
Returns  Information about the folder

**show_library** (library_id, contents=False)
Get information about a library.

Parameters
- **library_id** (str) – filter for library by library id
- **contents** (bool) – whether to get contents of the library (rather than just the library details)

Return type  dict
Returns  details of the given library

**update_library_dataset** (dataset_id, **kwds)**
Update library dataset metadata. Some of the attributes that can be modified are documented below.

Parameters
- **dataset_id** (str) – id of the dataset to be deleted
- **name** (str) – Replace library dataset name with the given string
- **misc_info** (str) – Replace library dataset misc_info with given string
- **file_ext** (str) – Replace library dataset extension (must exist in the Galaxy registry)
- **genome_build** (str) – Replace library dataset genome build (dbkey)

Return type  dict
Returns  details of the updated dataset

**upload_file_contents** (library_id, pasted_content, folder_id=None, file_type='auto', dbkey='?')
Upload pasted_content to a data library as a new file.

Parameters
• **library_id** *(str)* – id of the library where to place the uploaded file

• **pasted_content** *(str)* – Content to upload into the library

• **folder_id** *(str)* – id of the folder where to place the uploaded file. If not provided, the root folder will be used

• **file_type** *(str)* – Galaxy file format name

• **dbkey** *(str)* – Dbkey

**Return type**  list

**Returns**  List with a single dictionary containing information about the LDDA

### upload_file_from_local_path

```
(library_id, file_local_path, folder_id=None, file_type='auto',
 dbkey='?')
```

Read local file contents from file_local_path and upload data to a library.

**Parameters**

• **library_id** *(str)* – id of the library where to place the uploaded file

• **file_local_path** *(str)* – path of local file to upload

• **folder_id** *(str)* – id of the folder where to place the uploaded file. If not provided, the root folder will be used

• **file_type** *(str)* – Galaxy file format name

• **dbkey** *(str)* – Dbkey

**Return type**  list

**Returns**  List with a single dictionary containing information about the LDDA

### upload_file_from_server

```
(library_id, server_dir, folder_id=None, file_type='auto',
 dbkey='?', link_data_only=None, roles='", preserve_dirs=False, tag_using_filenames=True)
```

Upload all files in the specified subdirectory of the Galaxy library import directory to a library.

**Note:**  For this method to work, the Galaxy instance must have the library_import_dir option configured in the config/galaxy.ini configuration file.

**Parameters**

• **library_id** *(str)* – id of the library where to place the uploaded file

• **server_dir** *(str)* – relative path of the subdirectory of library_import_dir to upload. All and only the files (i.e. no subdirectories) contained in the specified directory will be uploaded

• **folder_id** *(str)* – id of the folder where to place the uploaded files. If not provided, the root folder will be used

• **file_type** *(str)* – Galaxy file format name

• **dbkey** *(str)* – Dbkey

• **link_data_only** *(str)* – either ‘copy_files’ (default) or ‘link_to_files’. Setting to ‘link_to_files’ symlinks instead of copying the files

• **roles** *(str)* – ???
• **preserve_dirs** (*bool*) – Indicate whether to preserve the directory structure when importing dir

• **tag_using_filenames** (*bool*) – Indicate whether to generate dataset tags from filenames

**Return type** list

**Returns** List with a single dictionary containing information about the LDDA

**upload_file_from_url**(*library_id*, *file_url*, *folder_id=None*, *file_type='auto'*, *dbkey='?'*)

Upload a file to a library from a URL.

**Parameters**

• **library_id** (*str*) – id of the library where to place the uploaded file

• **file_url** (*str*) – URL of the file to upload

• **folder_id** (*str*) – id of the folder where to place the uploaded file. If not provided, the root folder will be used

• **file_type** (*str*) – Galaxy file format name

• **dbkey** (*str*) – Dbkey

**Return type** list

**Returns** List with a single dictionary containing information about the LDDA

**upload_from_galaxy_filesystem**(*library_id*, *filesystem_paths*, *folder_id=None*, *file_type='auto'*, *dbkey='?'*, *link_data_only=None*, *roles='*', *preserve_dirs=False*, *tag_using_filenames=True*)

Upload a set of files already present on the filesystem of the Galaxy server to a library.

**Note:** For this method to work, the Galaxy instance must have the `allow_path_paste` (allow_library_path_paste in Galaxy release 17.05 and earlier) option set to True in the config/galaxy.ini configuration file.
**Return type**  list

**Returns**  List with a single dictionary containing information about the LDDA

`wait_for_dataset(library_id, dataset_id, maxwait=12000, interval=3)`

Wait until the library dataset state is terminal (‘ok’, ‘empty’, ‘error’, ‘discarded’ or ‘failed_metadata’).

**Parameters**

- **library_id (str)** – library id where dataset is found in
- **dataset_id (str)** – id of the dataset to wait for
- **maxwait (float)** – Total time (in seconds) to wait for the dataset state to become terminal. If the dataset state is not terminal within this time, a `DatasetTimeoutException` will be thrown.
- **interval (float)** – Time (in seconds) to wait between 2 consecutive checks.

**Return type**  dict

**Returns**  A dictionary containing information about the dataset in the library

---

**Quotas**

Contains possible interactions with the Galaxy Quota

`class bioblend.galaxy.quotas.QuotaClient(galaxy_instance)`

`create_quota(name, description, amount, operation, default=’no’, in_users=[], in_groups=[])`  
Create a new quota

**Parameters**

- **name (str)** – Name for the new quota. This must be unique within a Galaxy instance.
- **description (str)** – Quota description
- **amount (str)** – Quota size (E.g. 10000MB, 99 gb, 0.2T, unlimited)
- **operation (str)** – One of (+, -, =)
- **default (str)** – Whether or not this is a default quota. Valid values are no, unregistered, registered. None is equivalent to no.
- **in_users (list of str)** – A list of user IDs or user emails.
- **in_groups (list of str)** – A list of group IDs or names.

**Return type**  dict

**Returns**  A description of quota. For example:

```
{
    u'url': '/galaxy/api/quotas/386f14984287a0f7',
    u'model_class': 'Quota',
    u'message': "Quota 'Testing' has been created with 1 associated users and 0 associated groups.",
    u'id': '386f14984287a0f7',
    u'name': 'Testing'
}
```
**delete_quota**(quota_id)
Delete a quota

Before a quota can be deleted, the quota must not be a default quota.

**Parameters** quota_id**(str)** – Encoded quota ID.

**Return type** str

**Returns**
A description of the changes, mentioning the deleted quota. For example:

"Deleted 1 quotas: Testing-B"

**get_quotas**(deleted=False)
Get a list of quotas

**Parameters** deleted**(bool)** – Only return quota(s) that have been deleted

**Return type** list

**Returns**
A list of dicts with details on individual quotas. For example:

```python
[
    {u'id': u'0604c8a56abe9a50',
     u'model_class': u'Quota',
     u'name': u'test ',
     u'url': u'/api/quotas/0604c8a56abe9a50'},
    {u'id': u'1ee267091d0190af',
     u'model_class': u'Quota',
     u'name': u'workshop',
     u'url': u'/api/quotas/1ee267091d0190af'}
]
```

**show_quota**(quota_id, deleted=False)
Display information on a quota

**Parameters**
• quota_id**(str)** – Encoded quota ID

• deleted**(bool)** – Search for quota in list of ones already marked as deleted

**Return type** dict

**Returns**
A description of quota. For example:

```python
{u'bytes': 107374182400,
 u'default': [],
 u'description': u'just testing',
 u'display_amount': u'100.0 GB',
 u'groups': [],
 u'id': u'0604c8a56abe9a50',
 u'model_class': u'Quota',
 u'name': u'test ',
 u'operation': u'=',
 u'users': []
}
```

**undelete_quota**(quota_id)
Undelete a quota
Parameters **quota_id** *(str)* – Encoded quota ID.

Return type **str**

Returns

A description of the changes, mentioning the undeleted quota. For example:

"Undeleted 1 quotas: Testing-B"

**update_quota** *(quota_id, name=None, description=None, amount=None, operation=None, default='no', in_users=[], in_groups=[])*

Update an existing quota

Parameters

- **quota_id** *(str)* – Encoded quota ID
- **name** *(str)* – Name for the new quota. This must be unique within a Galaxy instance.
- **description** *(str)* – Quota description. If you supply this parameter, but not the name, an error will be thrown.
- **amount** *(str)* – Quota size (E.g. 10000MB, 99 gb, 0.2T, unlimited)
- **operation** *(str)* – One of (+, -, =). If you wish to change this value, you must also provide the amount, otherwise it will not take effect.
- **default** *(str)* – Whether or not this is a default quota. Valid values are no, unregister, registered. Calling this method with default="no" on a non-default quota will throw an error. Not passing this parameter is equivalent to passing no.
- **in_users** *(list of str)* – A list of user IDs or user emails.
- **in_groups** *(list of str)* – A list of group IDs or names.

Return type **str**

Returns

A semicolon separated list of changes to the quota. For example:

"Quota 'Testing-A' has been renamed to 'Testing-B'; Quota 'Testing-e →' is now '-100.0 GB'; Quota 'Testing-B' is now the default for unregistered users"

**Roles**

Contains possible interactions with the Galaxy Roles

```python
class bioblend.galaxy.roles.RolesClient(galaxy_instance)
```

**create_role** *(role_name, description, user_ids=[], group_ids=[])*

Create a new role.

Parameters

- **role_name** *(str)* – A name for the new role
- **description** *(str)* – Description for the new role
- **user_ids** *(list)* – A list of encoded user IDs to add to the new role
BioBlend Documentation, Release 0.12.0

- **group_ids** *(list)* – A list of encoded group IDs to add to the new role

  **Return type**  list

  **Returns**

  A (size 1) list with newly created role details, like:

  ```python
  [{
      'description': 'desc',
      'url': '/api/roles/ebfb8f50c6abde6d',
      'model_class': 'Role',
      'type': 'admin',
      'id': 'ebfb8f50c6abde6d',
      'name': 'Foo'}]
  ```

  get_roles()

  Displays a collection (list) of roles.

  **Return type**  list

  **Returns**

  A list of dicts with details on individual roles. For example:

  ```python
  [{'id': 'f2db41e1fa331b3e',
    'model_class': 'Role',
    'name': 'Foo',
    'url': '/api/roles/f2db41e1fa331b3e'},
   {'id': 'f597429621d6eb2b',
    'model_class': 'Role',
    'name': 'Bar',
    'url': '/api/roles/f597429621d6eb2b'}]
  ```

  show_role *(role_id)*

  Display information on a single role

  **Parameters**  role_id *(str)* – Encoded role ID

  **Return type**  dict

  **Returns**

  A description of role For example:

  ```python
  {'description': 'Private Role for Foo',
   'id': 'f2db41e1fa331b3e',
   'model_class': 'Role',
   'name': 'Foo',
   'type': 'private',
   'url': '/api/roles/f2db41e1fa331b3e'}
  ```

Tools

Contains possible interaction dealing with Galaxy tools.

**class**  `bioblend.galaxy.tools.ToolClient` *(galaxy_instance)*

  get_tool_panel()

  Get a list of available tool elements in Galaxy’s configured toolbox.
get_tools (tool_id=None, name=None, trackster=None)

Get all tools or filter the specific one(s) via the provided name or tool_id. Provide only one argument, name or tool_id, but not both.

If name is set and multiple names match the given name, all the tools matching the argument will be returned.

Parameters

• tool_id (str) – id of the requested tool
• name (str) – name of the requested tool(s)
• trackster (bool) – whether to return only tools that are compatible with Trackster

Return type

list

Returns

List of tool descriptions.

See also:

bioblend.galaxy.toolshed.get_repositories()

install_dependencies (tool_id)

Install dependencies for a given tool via a resolver. This works only for Conda currently. This functionality is available since Galaxy release_16.10 and is available only to Galaxy admins.

Parameters

• tool_id (str) – id of the requested tool

Return type

dict

Returns

Tool requirement status

paste_content (content, history_id, **kwds)

Upload a string to a new dataset in the history specified by history_id.

Parameters

• content (str) – content of the new dataset to upload or a list of URLs (one per line) to upload
• history_id (str) – id of the history where to upload the content

Return type

dict

Returns

Information about the created upload job

See upload_file() for the optional parameters.

put_url (content, history_id, **kwds)

Upload a string to a new dataset in the history specified by history_id.

Parameters

• content (str) – content of the new dataset to upload or a list of URLs (one per line) to upload
• history_id (str) – id of the history where to upload the content

Return type

dict
Returns Information about the created upload job

See upload_file() for the optional parameters.

run_tool (history_id, tool_id, tool_inputs)

Runs tool specified by tool_id in history indicated by history_id with inputs from dict tool_inputs.

Parameters

• **history_id** (str) – encoded ID of the history in which to run the tool

• **tool_id** (str) – ID of the tool to be run

• **tool_inputs** (dict) – dictionary of input datasets and parameters for the tool (see below)

Return type dict

Returns

Information about outputs and job For example:

```
{
    "outputs": [
    {
        "misc_blurb": "queued",
        "peek": null,
        "update_time": "2019-05-08T12:26:16.069798",
        "data_type": "galaxy.datatypes.tabular.Tabular",
        "tags": [],
        "deleted": false,
        "history_id": "df8fe5ddadbf3ab1",
        "metadata_column_names": null,
        "metadata_delimiter": "",
        "visible": true,
        "genome_build": "?",
        "create_time": "2019-05-08T12:26:15.997739",
        "hid": 42,
        "file_size": 0,
        "metadata_data_lines": null,
        "file_ext": "tabular",
        "id": "aeb65580396167f3",
        "misc_info": null,
        "hda_ldda": "hda",
        "history_content_type": "dataset",
        "name": "Cut on data 1",
        "metadata_columns": null,
        "uuid": "d91d10af-7546-45be-baa9-902010661466",
        "state": "new",
        "metadata_comment_lines": null,
        "model_class": "HistoryDatasetAssociation",
        "metadata_dbkey": "?",
        "output_name": "out_file1",
        "purged": false,
        "metadata_column_types": null
    }
    ],
    "implicit_collections": [],
    "jobs": [
    {
```
(continues on next page)
The `tool_inputs` dict should contain input datasets and parameters in the (largely undocumented) format used by the Galaxy API. Some examples can be found in Galaxy’s API test suite.

**show_tool** (*tool_id*, *io_details=False*, *link_details=False*)

Get details of a given tool.

- **Parameters**
  - `tool_id` *(str)* – id of the requested tool
  - `io_details` *(bool)* – whether to get also input and output details
  - `link_details` *(bool)* – whether to get also link details

- **Return type** dict

- **Returns** Information about the tool’s interface

**upload_file** (*path*, *history_id*, **keywords*)

Upload the file specified by `path` to the history specified by `history_id`

- **Parameters**
  - `path` *(str)* – path of the file to upload
  - `history_id` *(str)* – id of the history where to upload the file
  - `file_name` *(str)* – (optional) name of the new history dataset
  - `file_type` *(str)* – (optional) Galaxy datatype for the new dataset, default is auto
  - `dbkey` *(str)* – (optional) genome dbkey
  - `to_posix_lines` *(bool)* – if `True` (the default), convert universal line endings to POSIX line endings. Set to `False` when uploading a gzip, bz2 or zip archive containing a binary file
  - `space_to_tab` *(bool)* – whether to convert spaces to tabs. Default is `False`. Applicable only if to_posix_lines is `True`

- **Return type** dict

- **Returns** Information about the created upload job

**upload_from_ftp** (*path*, *history_id*, **keywords*)

Upload the file specified by `path` from the user’s FTP directory to the history specified by `history_id`

- **Parameters**
  - `path` *(str)* – path of the file in the user’s FTP directory
  - `history_id` *(str)* – id of the history where to upload the file
See `upload_file()` for the optional parameters.

**Return type**  dict

**Returns**  Information about the created upload job

---

### Tool data tables

Contains possible interactions with the Galaxy Tool data tables

```python
class bioblend.galaxy.tool_data.ToolDataClient(galaxy_instance)
```

#### delete_data_table(data_table_id, values)

Delete an item from a data table.

**Parameters**

- `data_table_id` (str) – ID of the data table
- `values` (str) – a “|” separated list of column contents, there must be a value for all the columns of the data table

**Return type**  dict

**Returns**  Remaining contents of the given data table

#### get_data_tables()

Get the list of all data tables.

**Return type**  list

**Returns**

A list of dicts with details on individual data tables. For example:

```json
[{
  "model_class": "TabularToolDataTable",
  "name": "fasta_index",
  "columns": ["value", "dbkey", "name", "path"],
  "fields": [["test id", "test", "test name", "/opt/galaxy-dist/tool-data/test/seq/test_id.fa")]],
  "model_class": "TabularToolDataTable",
  "name": "all_fasta"
}
```

#### reload_data_table(data_table_id)

Reload a data table.

**Parameters**  `data_table_id` (str) – ID of the data table

**Return type**  dict

**Returns**

A description of the given data table and its content. For example:

```json
{"columns": ["value", "dbkey", "name", "path"],
  "fields": [["test id", "test", "test name", "/opt/galaxy-dist/tool-data/test/seq/test_id.fa")]],
  "model_class": "TabularToolDataTable",
  "name": "all_fasta"
}
```

#### show_data_table(data_table_id)

Get details of a given data table.

**Parameters**  `data_table_id` (str) – ID of the data table
Return type  dict

Returns

A description of the given data table and its content. For example:

```json
{"columns": ["value", "dbkey", "name", "path"],
 "fields": ["test id",
    "test",
    "test name",
    "/opt/galaxy-dist/tool-data/test/seq/test id.fa"],
 "model_class": "TabularToolDataTable",
 "name": "all_fasta"}
```

---

**ToolShed**

Interaction with a Galaxy Tool Shed.

class `bioblend.galaxy.toolshed.ToolShedClient(galaxy_instance)`

`get_repositories()`  
Get the list of all installed Tool Shed repositories on this Galaxy instance.

Return type  list

Returns

a list of dictionaries containing information about repositories present in the Tool Shed. For example:

```json
[{u'changeset_revision': u'4afe13ac23b6',
  u'deleted': False,
  u'dist_to_shed': False,
  u'error_message': u'',
  u'name': u'velvet_toolsuite',
  u'owner': u'edward-kirton',
  u'status': u'Installed'}]
```

Changed in version 0.4.1: Changed method name from `get_tools` to `get_repositories` to better align with the Tool Shed concepts

See also:

`bioblend.galaxy.tools.get_tool_panel()`

`install_repository_revision(tool_shed_url, name, owner, changeset_revision, install_tool_dependencies=False, install_repository_dependencies=False, install_resolver_dependencies=False, tool_panel_section_id=None, new_tool_panel_section_label=None)`  
Install a specified repository revision from a specified Tool Shed into this Galaxy instance. This example demonstrates installation of a repository that contains valid tools, loading them into a section of the Galaxy tool panel or creating a new tool panel section. You can choose if tool dependencies or repository dependencies should be installed through the Tool Shed, (use `install_tool_dependencies` or `install_repository_dependencies`) or through a resolver that supports installing dependencies (use `install_resolver_dependencies`). Note that any combination of the three dependency resolving variables is valid.
Installing the repository into an existing tool panel section requires the tool panel config file (e.g., tool_conf.xml, shed_tool_conf.xml, etc) to contain the given tool panel section:

```xml
<section id="from_test_tool_shed" name="From Test Tool Shed" version="" />
```

### Parameters

- **tool_shed_url** *(str)* – URL of the Tool Shed from which the repository should be installed from (e.g., https://testtoolshed.g2.bx.psu.edu)
- **name** *(str)* – The name of the repository that should be installed
- **owner** *(str)* – The name of the repository owner
- **changeset_revision** *(str)* – The revision of the repository to be installed
- **install_tool_dependencies** *(bool)* – Whether or not to automatically handle tool dependencies (see https://galaxyproject.org/toolshed/tool-dependency-recipes/ for more details)
- **install_repository_dependencies** *(bool)* – Whether or not to automatically handle repository dependencies (see https://galaxyproject.org/toolshed/defining-repository-dependencies/ for more details)
- **install_resolver_dependencies** *(bool)* – Whether or not to automatically install resolver dependencies (e.g., conda). This parameter is silently ignored in Galaxy release_16.04 and earlier.
- **tool_panel_section_id** *(str)* – The ID of the Galaxy tool panel section where the tool should be inserted under. Note that you should specify either this parameter or the new_tool_panel_section_label. If both are specified, this one will take precedence.
- **new_tool_panel_section_label** *(str)* – The name of a Galaxy tool panel section that should be created and the repository installed into.

#### show_repository** *(toolShed_id)*

Get details of a given Tool Shed repository as it is installed on this Galaxy instance.

**Parameters**

- **toolShed_id** *(str)* – Encoded toolShed ID

**Return type** *dict*

**Returns**

Information about the tool For example:

```
{u'changeset_revision': u'b17455fb6222',
 u'ctx_rev': u'8',
 u'owner': u'aaron',
 u'status': u'Installed',
 u'url': u'/api/tool_shed_repositories/82de4a4c7135b20a')
```

Changed in version 0.4.1: Changed method name from show_tool to show_repository to better align with the Tool Shed concepts
**Users**

Contains possible interaction dealing with Galaxy users.

Most of these methods must be executed by a registered Galaxy admin user.

```python
class bioblend.galaxy.users.UserClient(galaxy_instance)
```

`create_local_user(username, user_email, password)`
Create a new Galaxy local user.

**Note:** For this method to work, the Galaxy instance must have the `allow_user_creation` option set to `True` and `use_remote_user` option set to `False` in the `config/galaxy.ini` configuration file.

**Parameters**

- `username` (**str**) – username of the user to be created
- `user_email` (**str**) – email of the user to be created
- `password` (**str**) – password of the user to be created

**Return type** dict

**Returns** a dictionary containing information about the created user

`create_remote_user(user_email)`
Create a new Galaxy remote user.

**Note:** For this method to work, the Galaxy instance must have the `allow_user_creation` and `use_remote_user` options set to `True` in the `config/galaxy.ini` configuration file. Also note that setting `use_remote_user` will require an upstream authentication proxy server; however, if you do not have one, access to Galaxy via a browser will not be possible.

**Parameters**

- `user_email` (**str**) – email of the user to be created

**Return type** dict

**Returns** a dictionary containing information about the created user

`create_user_apikey(user_id)`
Create a new API key for a given user.

**Parameters**

- `user_id` (**str**) – encoded user ID

**Return type** str

**Returns** the API key for the user

`delete_user(user_id, purge=False)`
Delete a user.

**Note:** For this method to work, the Galaxy instance must have the `allow_user_deletion` option set to `True` in the `config/galaxy.ini` configuration file.
Parameters

- **user_id** *(str)* – encoded user ID
- **purge** *(bool)* – if True, also purge (permanently delete) the history

Return type  dict

Returns  a dictionary containing information about the deleted user

---

**get_current_user()**

Display information about the user associated with this Galaxy connection.

Return type  dict

Returns  a dictionary containing information about the current user

---

**get_user_apikey(user_id)**

Get the current API key for a given user. This functionality is available since Galaxy release_17.01.

Parameters

- **user_id** *(str)* – encoded user ID

Return type  str

Returns  the API key for the user

---

**get_users(deleted=False, f_email=None, f_name=None, f_any=None)**

Get a list of all registered users. If `deleted` is set to True, get a list of deleted users.

Parameters

- **deleted** *(bool)* – Whether to include deleted users
- **f_email** *(str)* – filter for user emails. The filter will be active for non-admin users only if the Galaxy instance has the expose_user_email option set to True in the config/galaxy.ini configuration file. This parameter is silently ignored for non-admin users in Galaxy release_15.01 and earlier.
- **f_name** *(str)* – filter for user names. The filter will be active for non-admin users only if the Galaxy instance has the expose_user_name option set to True in the config/galaxy.ini configuration file. This parameter is silently ignored in Galaxy release_15.10 and earlier.
- **f_any** *(str)* – filter for user email or name. Each filter will be active for non-admin users only if the Galaxy instance has the corresponding expose_user_* option set to True in the config/galaxy.ini configuration file. This parameter is silently ignored in Galaxy release_15.10 and earlier.

Return type  list

Returns  a list of dicts with user details. For example:

```python
[]
```

---

**show_user(user_id, deleted=False)**

Display information about a user.

Parameters

- **user_id** *(str)* – encoded user ID

---
• **deleted**(bool) – whether to return results for a deleted user

**Return type**  
dict

**Returns**  
a dictionary containing information about the user

---

**Visual**

Contains possible interactions with the Galaxy visualization

**class**  
`bioblend.galaxy.visual.VisualClient(galaxy_instance)`

**get_visualizations**()  
Get the list of all visualizations.

**Return type**  
list

**Returns**  
A list of dicts with details on individual visualizations. For example:

```plaintext
[
    {u'dbkey': u'eschColi_K12',
     u'id': u'df1c7c96fc427c2d',
     u'title': u'AVTest1',
     u'type': u'trackster',
     u'url': u'/api/visualizations/df1c7c96fc427c2d'},
    {u'dbkey': u'mm9',
     u'id': u'a669f50f8bf55b02',
     u'title': u'Bam to Bigwig',
     u'type': u'trackster',
     u'url': u'/api/visualizations/a669f50f8bf55b02'}
]
```

**show_visualization**(visual_id)  
Get details of a given visualization.

**Parameters**  
visual_id**(str)** – Encoded visualization ID

**Return type**  
dict

**Returns**  
A description of the given visualization. For example:

```plaintext
{u'annotation': None,
 u'dbkey': u'mm9',
 u'id': u'18df9134ea75e49c',
 u'latest_revision': { ... },
 u'model_class': u'Visualization',
 u'revisions': [u'aa90649bb3ec7dcb', u'20622bc6249c0c71'],
 u'slug': u'visualization-for-grant-1',
 u'title': u'Visualization For Grant',
 u'type': u'trackster',
 u'url': u'/u/azaron/v/visualization-for-grant-1',
 u'user_id': u'21e4aed91386ca8b'}
```
Workflows

Contains possible interactions with the Galaxy Workflows

```python
class bioblend.galaxy.workflows.WorkflowClient(galaxy_instance)
```

- `cancel_invocation(workflow_id, invocation_id)`
  Cancel the scheduling of a workflow.
  
  **Parameters**
  
  - `workflow_id` *(str)* – Encoded workflow ID
  - `invocation_id` *(str)* – Encoded workflow invocation ID
  
  **Return type** `dict`
  
  **Returns** The workflow invocation being cancelled

- `delete_workflow(workflow_id)`
  Delete a workflow identified by `workflow_id`.
  
  **Parameters** `workflow_id` *(str)* – Encoded workflow ID
  
  **Return type** `str`
  
  **Returns** A message about the deletion

  **Warning:** Deleting a workflow is irreversible - all workflow data will be permanently deleted.

- `export_workflow_dict(workflow_id)`
  Exports a workflow.
  
  **Parameters** `workflow_id` *(str)* – Encoded workflow ID
  
  **Return type** `dict`
  
  **Returns** Dictionary representing the requested workflow

- `export_workflow_json(workflow_id)`
  Deprecated since version 0.9.0: Use `export_workflow_dict()` instead.

- `export_workflow_to_local_path(workflow_id, file_local_path, use_default_filename=True)`
  Exports a workflow in JSON format to a given local path.
  
  **Parameters**
  
  - `workflow_id` *(str)* – Encoded workflow ID
  - `file_local_path` *(str)* – Local path to which the exported file will be saved. (Should not contain filename if use_default_name=True)
  - `use_default_filename` *(bool)* – If the use_default_name parameter is True, the exported file will be saved as `file_local_path/Galaxy-Workflow-%s.ga`, where %s is the workflow name. If use_default_name is False, file_local_path is assumed to contain the full file path including filename.
  
  **Return type** None
  
  **Returns** None

- `get_invocations(workflow_id)`
  Get a list containing all the workflow invocations corresponding to the specified workflow.
**Parameters** `workflow_id (str)` – Encoded workflow ID

**Return type** list

**Returns**

A list of workflow invocations. For example:

```python
[{
    'history_id': '2f94e8ae9edff68a',
    'id': 'df7a1f0c02a5b08e',
    'model_class': 'WorkflowInvocation',
    'state': 'new',
    'update_time': '2015-10-31T22:00:22',
    'uuid': 'c8aa2b1c-801a-11e5-a9e5-8ca98228593c',
    'workflow_id': '03501d7626bd192f' },
]```

**get_workflow_inputs (workflow_id, label)**

Get a list of workflow input IDs that match the given label. If no input matches the given label, an empty list is returned.

**Parameters**

- `workflow_id (str)` – Encoded workflow ID
- `label (str)` – label to filter workflow inputs on

**Return type** list

**Returns** list of workflow inputs matching the label query

**get_workflows (workflow_id=None, name=None, published=False)**

Get all workflows or filter the specific one(s) via the provided `name` or `workflow_id`. Provide only one argument, `name` or `workflow_id`, but not both.

**Parameters**

- `workflow_id (str)` – Encoded workflow ID (incompatible with `name`)
- `name (str)` – Filter by name of workflow (incompatible with `workflow_id`). If multiple names match the given name, all the workflows matching the argument will be returned.
- `published (bool)` – if True, return also published workflows

**Return type** list

**Returns**

A list of workflow dicts. For example:

```python
[{
    'id': '92c56938c2f9b315',
    'name': 'Simple',
    'url': '/api/workflows/92c56938c2f9b315'}]
```

**import_shared_workflow (workflow_id)**

Imports a new workflow from the shared published workflows.

**Parameters** `workflow_id (str)` – Encoded workflow ID

**Return type** dict

**Returns**

A description of the workflow. For example:
import_workflow_dict (workflow_dict, publish=False)

Imports a new workflow given a dictionary representing a previously exported workflow.

Parameters

- **workflow_dict** (dict) – dictionary representing the workflow to be imported
- **publish** (bool) – if True the uploaded workflow will be published; otherwise it will be visible only by the user which uploads it (default)

Return type  dict

Returns

Information about the imported workflow. For example:

```
{u'name': 'Training: 16S rRNA sequencing with mothur: main tutorial',
 u'tags': [],
 u'deleted': False,
 u'latest_workflow_uuid': '368c6165-ccbe-4945-8a3c-d27982206d66',
 u'url': '/api/workflows/94bac0a90086bdcf',
 u'number_of_steps': 44,
 u'published': False,
 u'owner': 'jane-doe',
 u'model_class': 'StoredWorkflow',
 u'id': '94bac0a90086bdcf'}
```

import_workflow_from_local_path (file_local_path, publish=False)

Imports a new workflow given the path to a file containing a previously exported workflow.

Parameters

- **file_local_path** (str) – File to upload to the server for new workflow
- **publish** (bool) – if True the uploaded workflow will be published; otherwise it will be visible only by the user which uploads it (default)

Return type  dict

Returns

Information about the imported workflow. For example:

```
{u'name': 'Training: 16S rRNA sequencing with mothur: main tutorial',
 u'tags': [],
 u'deleted': False,
 u'latest_workflow_uuid': '368c6165-ccbe-4945-8a3c-d27982206d66',
 u'url': '/api/workflows/94bac0a90086bdcf',
 u'number_of_steps': 44,
 u'published': False,
 u'owner': 'jane-doe',
 u'model_class': 'StoredWorkflow',
 u'id': '94bac0a90086bdcf'}
```
import_workflow_json(workflow_json)
   Deprecated since version 0.9.0: Use import_workflow_dict() instead.

Parameters workflow_json(dict) – dictionary representing the workflow to be imported

invoke_workflow(workflow_id, inputs=None, params=None, history_id=None, history_name=None, import_inputs_to_history=False, replacement_params=None, allow_tool_state_corrections=None)
   Invoke the workflow identified by workflow_id. This will cause a workflow to be scheduled and return an object describing the workflow invocation.

Parameters
   • workflow_id(str) – Encoded workflow ID
   • inputs(dict) – A mapping of workflow inputs to datasets and dataset collections. The datasets source can be a LibraryDatasetDatasetAssociation (ldda), LibraryDataset (ld), HistoryDatasetAssociation (hda), or HistoryDatasetCollectionAssociation (hdca).

   The map must be in the following format: {'<input_index>': {'id': <encoded dataset ID>, 'src': [ldda, ld, hda, hdca]}} (e.g. {'2': {'id': '29beef4faded09f', 'src': 'hda'}})

   This map may also be indexed by the UUIDs of the workflow steps, as indicated by the uuid property of steps returned from the Galaxy API.
   • params(dict) – A mapping of non-datasets tool parameters (see below)
   • history_id(str) – The encoded history ID where to store the workflow output. Alternatively, history_name may be specified to create a new history.
   • history_name(str) – Create a new history with the given name to store the workflow output. If both history_id and history_name are provided, history_name is ignored. If neither is specified, a new 'Unnamed history' is created.
   • import_inputs_to_history(bool) – If True, used workflow inputs will be imported into the history. If False, only workflow outputs will be visible in the given history.
   • allow_tool_state_corrections(bool) – If True, allow Galaxy to fill in missing tool state when running workflows. This may be useful for workflows using tools that have changed over time or for workflows built outside of Galaxy with only a subset of inputs defined.
   • replacement_params(dict) – pattern-based replacements for post-job actions (see below)

Return type dict

Returns

A dict containing the workflow invocation describing the scheduling of the workflow. For example:

```json
{u'history_id': u'2f94e8ae9edff68a',
 u'id': u'df7a1f0c02a5b08e',
 u'inputs': {u'0': {u'id': u'a7db2fac67043c7e',
   u'src': u'hda',
   u'uuid': u'7932ffe0-2340-4952-8857-dbaa50f1f46a'}},
 u'model_class': u'WorkflowInvocation',
 u'state': u'ready',
 u'steps': [{u'action': None,
   ...}, ...
```
The `params` dict should be specified as follows:

```python
{STEP_ID: PARAM_DICT, ...}
```

where `PARAM_DICT` is:

```python
{PARAM_NAME: VALUE, ...}
```

For backwards compatibility, the following (deprecated) format is also supported for `params`:

```python
{TOOL_ID: PARAM_DICT, ...}
```

in which case `PARAM_DICT` affects all steps with the given tool id. If both by-tool-id and by-step-id specifications are used, the latter takes precedence.

Finally (again, for backwards compatibility), `PARAM_DICT` can also be specified as:

```python
[{'param': PARAM_NAME, 'value': VALUE}]
```

Note that this format allows only one parameter to be set per step.

For a repeat parameter, the names of the contained parameters needs to be specified as `<repeat name>_<repeat index>|<param name>`, with the repeat index starting at 0. For example, if the tool XML contains:

```xml
<repeat name="cutoff" title="Parameters used to filter cells" min="1">
  <param name="name" type="text" value="n_genes" label="Name of param...">
    <option value="n_genes">n_genes</option>
    <option value="n_counts">n_counts</option>
  </param>
  <param name="min" type="float" value="0" min="0" label="Min value"/>
</repeat>
```

then the `PARAM_DICT` should be something like:

```python
{u'params': [u'n_genes'], u'value': u'800'}
```
At the time of this writing, it is not possible to change the number of times the contained parameters are repeated. Therefore, the parameter indexes can go from 0 to n-1, where n is the number of times the repeated element was added when the workflow was saved in the Galaxy UI.

The replacement_params dict should map parameter names in post-job actions (PJAs) to their runtime values. For instance, if the final step has a PJA like the following:

```python
{u'RenameDatasetAction\text{\textbackslash}out_file1': {u'action_arguments': {u'newname': u'\$\rightarrow\{output\}'}, u'action_type': u'RenameDatasetAction', u'output_name': u'out_file1'}}
```

then the following renames the output dataset to ‘foo’:

```python
replacement_params = {'output': 'foo'}
```

see also this email thread.

**Warning:** Historically, the run_workflow method consumed a dataset_map data structure that was indexed by unencoded workflow step IDs. These IDs would not be stable across Galaxy instances. The new inputs property is instead indexed by either the order_index property (which is stable across workflow imports) or the step UUID which is also stable.

**run_invocation_step_action** *(workflow_id, invocation_id, step_id, action)*

Execute an action for an active workflow invocation step. The nature of this action and what is expected will vary based on the the type of workflow step (the only currently valid action is True/False for pause steps).

**Parameters**

- **workflow_id** *(str)* – Encoded workflow ID
- **invocation_id** *(str)* – Encoded workflow invocation ID
- **step_id** *(str)* – Encoded workflow invocation step ID
- **action** *(object)* – Action to use when updating state, semantics depends on step type.

**Return** type **dict**

**Returns** Representation of the workflow invocation step

**run_workflow** *(workflow_id, dataset_map=None, params=None, history_id=None, history_name=None, import_inputs_to_history=False, replacement_params=None)*

Run the workflow identified by workflow_id.

Deprecated since version 0.7.0: Use invoke_workflow() instead.

**Parameters**

- **workflow_id** *(str)* – Encoded workflow ID
dataset_map (dict) – A mapping of workflow inputs to datasets. The datasets source can be a LibraryDatasetDatasetAssociation (ldda), LibraryDataset (ld), or HistoryDatasetAssociation (hda). The map must be in the following format: {'<input>': {'id': <encoded dataset ID>, 'src': ['ldda, ld, hda']}) (e.g. {23: {'id': '29beef4fae6ed09f', 'src': 'ld'}})

params (dict) – A mapping of non-datasets tool parameters (see below)

history_id (str) – The encoded history ID where to store the workflow output. Alternatively, history_name may be specified to create a new history.

history_name (str) – Create a new history with the given name to store the workflow output. If both history_id and history_name are provided, history_name is ignored. If neither is specified, a new ‘Unnamed history’ is created.

import_inputs_to_history (bool) – If True, used workflow inputs will be imported into the history. If False, only workflow outputs will be visible in the given history.

replacement_params (dict) – pattern-based replacements for post-job actions (see below)

Return type  dict

Returns

A dict containing the history ID where the outputs are placed as well as output dataset IDs. For example:

```python
{u'history': u'64177123325c9cfd',
 u'outputs': [u'aa4d3084af404259']}
```

The params dict should be specified as follows:

```python
{STEP_ID: PARAM_DICT, ...}
```

where PARAM_DICT is:

```python
{PARAM_NAME: VALUE, ...}
```

For backwards compatibility, the following (deprecated) format is also supported for params:

```python
{TOOL_ID: PARAM_DICT, ...}
```

in which case PARAM_DICT affects all steps with the given tool id. If both by-tool-id and by-step-id specifications are used, the latter takes precedence.

Finally (again, for backwards compatibility), PARAM_DICT can also be specified as:

```python
{'param': PARAM_NAME, 'value': VALUE}
```

Note that this format allows only one parameter to be set per step.

The replacement_params dict should map parameter names in post-job actions (PJAs) to their run-time values. For instance, if the final step has a PJA like the following:

```python
{u'RenameDatasetActionout_file1': {u'action_arguments': {u'newname': u'$\rightarrow${output}'),
 u'action_type': u'RenameDatasetAction',
 u'output_name': u'out_file1'}}
```

62 Chapter 5. API Documentation
then the following renames the output dataset to ‘foo’:

```python
replacement_params = {'output': 'foo'}
```

see also this email thread.

**Warning:** This method waits for the whole workflow to be scheduled before returning and does not scale to large workflows as a result. This method has therefore been deprecated in favor of `invoke_workflow()`, which also features improved default behavior for dataset input handling.

### show_invocation(workflow_id, invocation_id)

Get a workflow invocation object representing the scheduling of a workflow. This object may be sparse at first (missing inputs and invocation steps) and will become more populated as the workflow is actually scheduled.

**Parameters**

- **workflow_id**(str) – Encoded workflow ID
- **invocation_id**(str) – Encoded workflow invocation ID

**Return type** dict

**Returns**

The workflow invocation. For example:

```python
{u'history_id': u'2f94e8ae9edff68a',
 u'id': u'df7a1f0c02a5b08e',
 u'inputs': {u'0': {u'id': u'a7db2fac67043c7e',
 u'src': u'hda',
 u'uuid': u'7932ffe0-2340-4952-8857-dbba50f1f46a'}},
 u'model_class': u'WorkflowInvocation',
 u'state': u'ready',
 u'steps': [{u'action': None,
 u'id': u'd413a19dec13d11e',
 u'model_class': u'WorkflowInvocationStep',
 u'order_index': 0,
 u'state': None,
 u'update_time': u'2015-10-31T22:00:26',
 u'workflow_step_id': u'cbbbf59e8f08c98c',
 u'workflow_step_label': None,
 u'workflow_step_uuid': u'b81250fd-3278-4e6a-b269-56a1f01ef485'},
 {u'action': None,
 u'id': u'2f94e8ae9edff68a',
 u'job_id': u'e89067bb68bee7a0',
 u'model_class': u'WorkflowInvocationStep',
 u'order_index': 1,
 u'state': 'new',
 u'update_time': u'2015-10-31T22:00:26',
 u'workflow_step_id': u'964b37715ec9bd22',
 u'workflow_step_label': None,
 u'workflow_step_uuid': u'e62440b8-e911-408b-b124-e05435d3125e'},
 {u'action': None,
 u'history_id': u'2f94e8ae9edff68a',
 u'id': u'df7a1f0c02a5b08e',
 u'inputs': {u'0': {u'id': u'a7db2fac67043c7e',
 u'src': u'hda',
 u'uuid': u'7932ffe0-2340-4952-8857-dbba50f1f46a'}},
 u'model_class': u'WorkflowInvocation',
 u'state': u'ready',
 u'steps': [{u'action': None,
 u'id': u'd413a19dec13d11e',
 u'model_class': u'WorkflowInvocationStep',
 u'order_index': 0,
 u'state': None,
 u'update_time': u'2015-10-31T22:00:26',
 u'workflow_step_id': u'cbbbf59e8f08c98c',
 u'workflow_step_label': None,
 u'workflow_step_uuid': u'b81250fd-3278-4e6a-b269-56a1f01ef485'}]}
```
show_invocation_step(workflow_id, invocation_id, step_id)

See the details of a particular workflow invocation step.

Parameters

- workflow_id (str) – Encoded workflow ID
- invocation_id (str) – Encoded workflow invocation ID
- step_id (str) – Encoded workflow invocation step ID

Return type  dict

Returns

The workflow invocation step. For example:

```python
{'action': None,
 'id': '63cd3858d057a6d1',
 'job_id': None,
 'model_class': 'WorkflowInvocationStep',
 'order_index': 2,
 'state': None,
 'update_time': '2015-10-31T22:11:14',
 'workflow_step_id': '52e496b945151ee8',
 'workflow_step_label': None,
 'workflow_step_uuid': '4060554c-1dd5-4287-9040-8b4f281cf9dc'}
```

show_workflow(workflow_id)

Display information needed to run a workflow.

Parameters  workflow_id (str) – Encoded workflow ID

Return type  dict

Returns

A description of the workflow and its inputs. For example:

```python
{u'id': u'92c56938c2f9b315',
 u'inputs': {u'23': {u'label': u'Input Dataset', u'value': u''}},
 u'name': u'Simple',
 u'url': u'/api/workflows/92c56938c2f9b315'}
```

5.2.2 Object-oriented Galaxy API

class bioblend.galaxy.objects.galaxy_instance.GalaxyInstance(url, api_key=None, email=None, password=None, verify=True)

A representation of an instance of Galaxy, identified by a URL and a user’s API key.

Parameters

- url (str) – a FQDN or IP for a given instance of Galaxy. For example: http://127.0.0.1:8080
- api_key (str) – user’s API key for the given instance of Galaxy, obtained from the Galaxy web UI.
This is actually a factory class which instantiates the entity-specific clients.

Example: get a list of all histories for a user with API key ‘foo’:

```python
from bioblend.galaxy.objects import *
gi = GalaxyInstance('http://127.0.0.1:8080', 'foo')
histories = gi.histories.list()
```

**Client**

Clients for interacting with specific Galaxy entity types.

Classes in this module should not be instantiated directly, but used via their handles in `GalaxyInstance`.

```python
class bioblend.galaxy.objects.client.ObjClient(obj_gi)
```

**get_previews(**kwargs**)

Get a list of object previews.

Previews entity summaries provided by REST collection URIs, e.g. `http://host:port/api/libraries`. Being the most lightweight objects associated to the various entities, these are the ones that should be used to retrieve their basic info.

Return type list

Returns a list of object previews

```python
list(**kwargs**)
```

Get a list of objects.

This method first gets the entity summaries, then gets the complete description for each entity with an additional GET call, so may be slow.

Return type list

Returns a list of objects

```python
class bioblend.galaxy.objects.client.ObjDatasetContainerClient(obj_gi)
```

```python
class bioblend.galaxy.objects.client.ObjHistoryClient(obj_gi)
```

Interacts with Galaxy histories.

```python
class bioblend.galaxy.objects.client.ObjHistoryClient(obj_gi)
```

**create**(name= None)

Create a new Galaxy history, optionally setting its name.

Return type History

Returns the history just created

**delete**(id_= None, name= None, purge= False)

Delete the history with the given id or name.

Note that the same name can map to multiple histories.

Parameters purge (bool) – if True, also purge (permanently delete) the history

**Note:** For the purge option to work, the Galaxy instance must have the `allow_user_dataset_purge` option set to True in the config/galaxy.ini configuration file.
**get (id_)**
Retrieve the history corresponding to the given id.

- **Return type**: History
- **Returns**: the history corresponding to id_

**get_previews (name=None, deleted=False)**
Get a list of object previews.

Previews entity summaries provided by REST collection URIs, e.g. http://host:port/api/libraries. Being the most lightweight objects associated to the various entities, these are the ones that should be used to retrieve their basic info.

- **Return type**: list
- **Returns**: a list of object previews

**list (name=None, deleted=False)**
Get histories owned by the user of this Galaxy instance.

- **Parameters**
  - name (str) – return only histories with this name
  - deleted (bool) – if True, return histories that have been deleted
- **Return type**: list of History

**class bioblend.galaxy.objects.client.ObjJobClient (obj_gi)**
Interacts with Galaxy jobs.

**get (id_, full_details=False)**
Retrieve the job corresponding to the given id.

- **Parameters**
  - full_details (bool) – if True, return the complete list of details for the given job.
- **Return type**: Job
- **Returns**: the job corresponding to id_

**get_previews ()**
Get a list of object previews.

Previews entity summaries provided by REST collection URIs, e.g. http://host:port/api/libraries. Being the most lightweight objects associated to the various entities, these are the ones that should be used to retrieve their basic info.

- **Return type**: list
- **Returns**: a list of object previews

**list ()**
Get the list of jobs of the current user.

- **Return type**: list of Job

**class bioblend.galaxy.objects.client.ObjLibraryClient (obj_gi)**
Interacts with Galaxy libraries.

**create (name, description=None, synopsis=None)**
Create a data library with the properties defined in the arguments.

- **Return type**: Library
- **Returns**: the library just created
**delete** (*id_=None, name=None*)
Delete the library with the given id or name.

Note that the same name can map to multiple libraries.

**Warning:** Deleting a data library is irreversible - all of the data from the library will be permanently deleted.

**get** (*id_*)
Retrieve the data library corresponding to the given id.

*Return type* *Library*
*Returns* the library corresponding to *id_*

**get_previews** (*name=None, deleted=False*)
Get a list of object previews.

Previews entity summaries provided by REST collection URIs, e.g. http://host:port/api/libraries. Being the most lightweight objects associated to the various entities, these are the ones that should be used to retrieve their basic info.

*Return type* *list*
*Returns* a list of object previews

**list** (*name=None, deleted=False*)
Get libraries owned by the user of this Galaxy instance.

*Parameters*

- *name* (*str*) – return only libraries with this name
- *deleted* (*bool*) – if True, return libraries that have been deleted

*Return type* *list* of *Library*

**class** `bioblend.galaxy.objects.client.ObjToolClient` *(obj_gi)*
Interacts with Galaxy tools.

**get** (*id_, io_details=False, link_details=False*)
Retrieve the tool corresponding to the given id.

*Parameters*

- *io_details* (*bool*) – if True, get also input and output details
- *link_details* (*bool*) – if True, get also link details

*Return type* *Tool*
*Returns* the tool corresponding to *id_*

**get_previews** (*name=None, trackster=None*)
Get the list of tools installed on the Galaxy instance.

*Parameters*

- *name* (*str*) – return only tools with this name
- *trackster* (*bool*) – if True, only tools that are compatible with Trackster are returned

*Return type* *list* of *Tool*
list (name=None, trackster=None)
Get the list of tools installed on the Galaxy instance.

Parameters
- name (str) – return only tools with this name
- trackster (bool) – if True, only tools that are compatible with Trackster are returned

Return type list of Tool

class bioblend.galaxy.objects.client.ObjWorkflowClient (obj_gi)
Interacts with Galaxy workflows.

delete (id_=None, name=None)
Delete the workflow with the given id or name.

Note that the same name can map to multiple workflows.

Warning: Deleting a workflow is irreversible - all of the data from the workflow will be permanently deleted.

get (id_)
Retrieve the workflow corresponding to the given id.

Return type Workflow

Returns the workflow corresponding to id_

get_previews (name=None, published=False)
Get a list of object previews.

Previews entity summaries provided by REST collection URIs, e.g. http://host:port/api/libraries. Being the most lightweight objects associated to the various entities, these are the ones that should be used to retrieve their basic info.

Return type list

Returns a list of object previews

import_new (src, publish=False)
Imports a new workflow into Galaxy.

Parameters
- src (dict or str) – deserialized (dictionary) or serialized (str) JSON dump of the workflow (this is normally obtained by exporting a workflow from Galaxy).
- publish (bool) – if True the uploaded workflow will be published; otherwise it will be visible only by the user which uploads it (default).

Return type Workflow

Returns the workflow just imported

import_shared (id_)
Imports a shared workflow to the user’s space.

Parameters id (str) – workflow id

Return type Workflow

Returns the workflow just imported
**list** *(name=None, published=False)*

Get workflows owned by the user of this Galaxy instance.

**Parameters**

- **name** *(str)* – return only workflows with this name
- **published** *(bool)* – if True, return also published workflows

**Return type** list of *Workflow*

---

**Wrappers**

A basic object-oriented interface for Galaxy entities.

```python
class bioblend.galaxy.objects.wrappers_WRAPPER(wrapped, parent=None, gi=None)
```

Abstract base class for Galaxy entity wrappers.

Wrapper instances wrap deserialized JSON dictionaries such as the ones obtained by the Galaxy web API, converting key-based access to attribute-based access (e.g., `library['name']` \(\rightarrow\) `library.name`).

Dict keys that are converted to attributes are listed in the `BASE_ATTRS` class variable: this is the ‘stable’ interface. Note that the wrapped dictionary is accessible via the `wrapped` attribute.

**Parameters**

- **wrapped** *(dict)* – JSON-serializable dictionary
- **parent** *(Wrapper)* – the parent of this wrapper
- **gi** *(GalaxyInstance)* – the GalaxyInstance through which we can access this wrapper

**BASE_ATTRS** = ('id', 'name')

**clone** ()

Return an independent copy of this wrapper.

**classmethod from_json** *(jdef)*

Build a new wrapper from a JSON dump.

**gi_module**

The GalaxyInstance module that deals with objects of this type.

**is_mapped**

True if this wrapper is mapped to an actual Galaxy entity.

**parent**

The parent of this wrapper.

**to_json** ()

Return a JSON dump of this wrapper.

**touch** ()

Mark this wrapper as having been modified since its creation.

**unmap** ()

Disconnect this wrapper from Galaxy.

```python
class bioblend.galaxy.objects.wrappers_STEP(step_dict, parent)
```

Abstract base class for workflow steps.

Steps are the main building blocks of a Galaxy workflow. A step can be: an input (type ‘data_collection_input’ or ‘data_input’), a computational tool (type ‘tool’) or a pause (type ‘pause’).

**BASE_ATTRS** = ('id', 'name', 'input_steps', 'tool_id', 'tool_inputs', 'tool_version', 'type')
BioBlend Documentation, Release 0.12.0

```
import gi_module

class bioblend.galaxy.objects.wrappers.Workflow(wf_dict, gi=None):
    Workflows represent ordered sequences of computations on Galaxy.
    A workflow defines a sequence of steps that produce one or more results from an input dataset.
    BASE_ATTRS = ('id', 'name', 'deleted', 'inputs', 'published', 'steps', 'tags')
    POLLLING_INTERVAL = 10

    convert_input_map(input_map)
        Convert input_map to the format required by the Galaxy web API.
        Parameters
        input_map (dict) -- a mapping from input labels to datasets
        Return type dict
        Returns a mapping from input slot ids to dataset ids in the format required by the Galaxy web API.

    data_collection_input_ids
        Return the ids of data collection input steps for this workflow.

    data_input_ids
        Return the ids of data input steps for this workflow.

    delete()
        Delete this workflow.
        Warning: Deleting a workflow is irreversible - all of the data from the workflow will be permanently deleted.

    export()
        Export a re-importable representation of the workflow.
        Return type dict
        Returns a JSON-serializable dump of the workflow

    input_labels
        Return the labels of this workflow's input steps.

    is_runnable
        Return True if the workflow can be run on Galaxy.
        A workflow is considered runnable on a Galaxy instance if all of the tools it uses are installed in that instance.

    preview()

    run(input_map=None, history='', params=None, import_inputs=False, replacement_params=None, wait=False, polling_interval=10, break_on_error=True)
        Run the workflow in the current Galaxy instance.
        Parameters
            * input_map (dict) -- a mapping from workflow input labels to datasets, e.g.: dict(zip(workflow.input_labels, library.get_datasets()))
            * history (History or str) -- either a valid history object (results will be stored there) or a string (a new history will be created with the given name).
```
• **params** (*dict*) – a mapping of non-datasets tool parameters (see below)

• **import_inputs** (*bool*) – If True, workflow inputs will be imported into the history; if False, only workflow outputs will be visible in the history.

• **replacement_params** (*dict*) – pattern-based replacements for post-job actions (see the docs for `invoke_workflow()`)

• **wait** (*bool*) – whether to wait while the returned datasets are in a pending state

• **polling_interval** (*float*) – polling interval in seconds

• **break_on_error** (*bool*) – whether to break as soon as at least one of the returned datasets is in the `error` state

**Return type** tuple

**Returns** list of output datasets, output history

The `params` dict should be specified as follows:

```python
{STEP_ID: PARAM_DICT, ...}
```

where PARAM_DICT is:

```python
{PARAM_NAME: VALUE, ...}
```

For backwards compatibility, the following (deprecated) format is also supported for `params`:

```python
{TOOL_ID: PARAM_DICT, ...}
```

in which case PARAM_DICT affects all steps with the given tool id. If both by-tool-id and by-step-id specifications are used, the latter takes precedence.

Finally (again, for backwards compatibility), PARAM_DICT can also be specified as:

```python
{'param': PARAM_NAME, 'value': VALUE}
```

Note that this format allows only one parameter to be set per step.

Example: set `a` to 1 for the third workflow step:

```python
params = {workflow.steps[2].id: {'a': 1}}
```

**Warning:** This is a blocking operation that can take a very long time. If `wait` is set to False, the method will return as soon as the workflow has been `scheduled`, otherwise it will wait until the workflow has been `run`. With a large number of steps, however, the delay may not be negligible even in the former case (e.g. minutes for 100 steps).

**sorted_step_ids()**

Return a topological sort of the workflow’s DAG.

**tool_ids**

Return the ids of tool steps for this workflow.

```python
class bioblend.galaxy.objects.wrappers.ContentInfo(info_dict, gi=None)
```

Instances of this class wrap dictionaries obtained by getting `/api/{histories,libraries}/<ID>/contents` from Galaxy.

```
BASE_ATTRS = ('id', 'name', 'type')
```
Instances of this class wrap dictionaries obtained by getting /api/libraries/<ID>/contents from Galaxy.

**BASE_ATTRS** = ('id', 'name', 'type', 'deleted', 'state', 'visible')

**gi_module**

Instances of this class wrap dictionaries obtained by getting /api/histories/<ID>/contents from Galaxy.

**BASE_ATTRS** = ('id', 'name', 'deleted')

**dataset_ids**

Return the ids of the contained datasets.

**get_dataset**(ds_id)

Retrieve the dataset corresponding to the given id.

**Parameters**

- **ds_id**(str) – dataset id

**Return type**

HistoryDatasetAssociation or LibraryDataset

**Returns**

the dataset corresponding to ds_id

**get_datasets**(name=None)

Get all datasets contained inside this dataset container.

**Parameters**

- **name**(str) – return only datasets with this name

**Return type**

list of HistoryDatasetAssociation or list of LibraryDataset

**Returns**

datasets with the given name contained inside this container

---

**Note:** when filtering library datasets by name, specify their full paths starting from the library's root folder, e.g., /seqdata/reads.fastq. Full paths are available through the content_infos attribute of Library objects.

**preview()**

**refresh()**

Re-fetch the attributes pertaining to this object.

**Returns:** self

**class** bioblend.galaxy.objects.wrappers.LibraryContentInfo(info_dict, gi=None)

Maps to a Galaxy library.

**API_MODULE** = 'libraries'

**BASE_ATTRS** = ('id', 'name', 'deleted', 'annotation', 'published', 'state', 'state_ids')
create_dataset_collection(collection_description)
Create a new dataset collection in the history by providing a collection description.

Parameters
collection_description (bioblend.galaxy.
dataset_collections.CollectionDescription) – a description of the
dataset collection

Return type HistoryDatasetCollectionAssociation

Returns the new dataset collection

download(jeha_id, outf, chunk_size=4096)
Download an export archive for this history. Use export() to create an export and get the required
jeha_id. See download_history() for parameter and return value info.

export(gzip=True, include_hidden=False, include_deleted=False, wait=False, maxwait=None)
Start a job to create an export archive for this history. See export_history() for parameter and return
value info.

gi_module

import_dataset(lds)
Import a dataset into the history from a library.

Parameters lds (LibraryDataset) – the library dataset to import

Return type HistoryDatasetAssociation

Returns the imported history dataset

paste_content(content, **kwargs)
Upload a string to a new dataset in this history.

Parameters content (str) – content of the new dataset to upload

See upload_file() for the optional parameters (except file_name).
Return type: `HistoryDatasetAssociation`

Returns the uploaded dataset

**update(** **kwargs)

Update history metadata information. Some of the attributes that can be modified are documented below.

**Parameters**

- `name` (`str`): Replace history name with the given string
- `annotation` (`str`): Replace history annotation with the given string
- `deleted` (`bool`): Mark or unmark history as deleted
- `purged` (`bool`): If True, mark history as purged (permanently deleted). Ignored on Galaxy release_15.01 and earlier
- `published` (`bool`): Mark or unmark history as published
- `importable` (`bool`): Mark or unmark history as importable
- `tags` (`list`): Replace history tags with the given list

**upload_dataset** (*path*, **kwargs)

Upload the file specified by *path* to this history.

**Parameters**

- `path` (`str`): path of the file to upload

See `upload_file()` for the optional parameters.

**Return type:** `HistoryDatasetAssociation`

**Returns:** the uploaded dataset

**upload_file** (*path*, **kwargs)

Upload the file specified by *path* to this history.

**Parameters**

- `path` (`str`): path of the file to upload

See `upload_file()` for the optional parameters.

**Return type:** `HistoryDatasetAssociation`

**Returns:** the uploaded dataset

**upload_from_ftp** (*path*, **kwargs)

Upload the file specified by *path* from the user’s FTP directory to this history.

**Parameters**

- `path` (`str`): path of the file in the user’s FTP directory

See `upload_file()` for the optional parameters.

**Return type:** `HistoryDatasetAssociation`

**Returns:** the uploaded dataset

**class** `bioblend.galaxy.objects.wrappers.Library`

Maps to a Galaxy library.

**API_MODULE** = 'libraries'

**BASE_ATTRS** = ('id', 'name', 'deleted', 'description', 'synopsis')

**CONTENT_INFO_TYPE**

alias of `LibraryContentInfo`
**DS_TYPE**

alias of *LibraryDataset*

**copy_from_dataset** *(hda, folder=None, message='')*

Copy a history dataset into this library.

**Parameters**

- **hda** *(HistoryDatasetAssociation)* – history dataset to copy into the library

See `upload_data()` for info on other params.

**create_folder** *(name, description=None, base_folder=None)*

Create a folder in this library.

**Parameters**

- **name** *(str)* – folder name
- **description** *(str)* – optional folder description
- **base_folder** *(Folder)* – parent folder, or `None` to create in the root folder

**Return type** *Folder*

**Returns** the folder just created

**delete()**

Delete this library.

**folder_ids**

Return the ids of the contained folders.

**get_folder** *(f_id)*

Retrieve the folder corresponding to the given id.

**Return type** *Folder*

**Returns** the folder corresponding to `f_id`

**gi_module**

The root folder of this library.

**Return type** *Folder*

**Returns** the root folder of this library

**upload_data** *(data, folder=None, **kwargs)*

Upload data to this library.

**Parameters**

- **data** *(str)* – dataset contents
- **folder** *(Folder)* – a folder object, or `None` to upload to the root folder

**Return type** *LibraryDataset*

**Returns** the dataset object that represents the uploaded content

Optional keyword arguments: `file_type`, `dbkey`.

**upload_from_galaxy_fs** *(paths, folder=None, link_data_only=None, **kwargs)*

Upload data to this library from filesystem paths on the server.
Note: For this method to work, the Galaxy instance must have the allow_path_paste (allow_library_path_paste in Galaxy release_17.05 and earlier) option set to True in the config/galaxy.ini configuration file.

Parameters

- **paths** (str or Iterable of str) – server-side file paths from which data should be read
- **link_data_only** (str) – either 'copy_files' (default) or 'link_to_files'. Setting to 'link_to_files' symlinks instead of copying the files

Return type list of LibraryDataset

Returns the dataset objects that represent the uploaded content

See upload_data() for info on other params.

**upload_from_local** *(path, folder=None, **kwargs)*

Upload data to this library from a local file.

Parameters **path** (str) – local file path from which data should be read

See upload_data() for info on other params.

**upload_from_url** *(url, folder=None, **kwargs)*

Upload data to this library from the given URL.

Parameters **url** (str) – URL from which data should be read

See upload_data() for info on other params.

**Folder**

Maps to a folder in a Galaxy library.

**parent** The parent folder of this folder. The parent of the root folder is None.

Return type Folder

Returns the parent of this folder

**refresh**()

Re-fetch the attributes pertaining to this object.

Returns: self

**Dataset**

Abstract base class for Galaxy datasets.

**BASE_ATTRS** = ('id', 'name', 'description', 'deleted', 'item_count')

**container_id**

Deprecated property.

Id of the folder container. Use container.id instead.

**gi_module**

**parent**

The parent folder of this folder. The parent of the root folder is None.

Return type Folder

Returns the parent of this folder

**refresh**()

Re-fetch the attributes pertaining to this object.

Returns: self
container_id
   Deprecated property.
   Id of the dataset container. Use container.id instead.

download (file_object, chunk_size=4096)
   Open dataset for reading and save its contents to file_object.
   Parameters file_object (file) – output file object
   See get_stream() for info on other params.

get_contents (chunk_size=4096)
   Open dataset for reading and return its full contents.
   See get_stream() for param info.

get_stream (chunk_size=4096)
   Open dataset for reading and return an iterator over its contents.
   Parameters chunk_size (int) – read this amount of bytes at a time

peek (chunk_size=4096)
   Open dataset for reading and return the first chunk.
   See get_stream() for param info.

refresh ()
   Re-fetch the attributes pertaining to this object.
   Returns: self

wait (polling_interval=1, break_on_error=True)
   Wait for this dataset to come out of the pending states.
   Parameters
      • polling_interval (float) – polling interval in seconds
      • break_on_error (bool) – if True, raise a RuntimeError exception if the dataset ends in the 'error' state.
   Warning: This is a blocking operation that can take a very long time. Also, note that this method does not return anything; however, this dataset is refreshed (possibly multiple times) during the execution.

class bioblend.galaxy.objects.wrappers.HistoryDatasetAssociation (ds_dict, container, gi=None)
Maps to a Galaxy HistoryDatasetAssociation.
BASE_ATTRS = ('id', 'name', 'data_type', 'file_ext', 'file_name', 'file_size', 'genome_build', 'misc_info', 'state', 'annotation', 'deleted', 'purged', 'tags', 'visible')
SRC = 'hda'
delete (purge=False)
   Delete this history dataset.
   Parameters purge (bool) – if True, also purge (permanently delete) the dataset
   Note: For the purge option to work, the Galaxy instance must have the
allow_user_dataset_purge option set to True in the config/galaxy.ini configuration file.

**Warning:** If you purge a dataset which has not been previously deleted, Galaxy since release 15.03 wrongly does not set the deleted attribute of the dataset to True, see https://github.com/galaxyproject/galaxy/issues/3548

```
gi_module
update(**kwds)

Update this history dataset metadata. Some of the attributes that can be modified are documented below.

Parameters

- name (str) – Replace history dataset name with the given string
- genome_build (str) – Replace history dataset genome build (dbkey)
- annotation (str) – Replace history dataset annotation with given string
- deleted (bool) – Mark or unmark history dataset as deleted
- visible (bool) – Mark or unmark history dataset as visible
```

class bioblend.galaxy.objects.wrappers.DatasetCollection(dsc_dict, container, gi=None)

Abstract base class for Galaxy dataset collections.

BASE_ATTRS = ('id', 'name', 'state', 'deleted', 'collection_type')

refresh()

Re-fetch the attributes pertaining to this object.

Returns: self

class bioblend.galaxy.objects.wrappers.HistoryDatasetCollectionAssociation(dsc_dict, container, gi=None)

Maps to a Galaxy HistoryDatasetCollectionAssociation.

BASE_ATTRS = ('id', 'name', 'state', 'deleted', 'collection_type', 'tags', 'visible', 'elements')

SRC = 'hdca'

delete()

Delete this dataset collection.

```
gi_module

class bioblend.galaxy.objects.wrappers.LibraryDatasetDatasetAssociation(ds_dict, container, gi=None)

Maps to a Galaxy LibraryDatasetDatasetAssociation.

BASE_ATTRS = ('id', 'name', 'data_type', 'file_ext', 'file_name', 'file_size', 'genome_build', 'misc_info', 'state', 'deleted')

SRC = 'ldda'

class bioblend.galaxy.objects.wrappers.LibraryDataset(ds_dict, container, gi=None)

Maps to a Galaxy LibraryDataset.
SRC = 'ld'

def delete(purged=False)
    Delete this library dataset.
    
    Parameters
    purged (bool) – if True, also purge (permanently delete) the dataset

update(**kwds)
    Update this library dataset metadata. Some of the attributes that can be modified are documented below.
    
    Parameters
    • name (str) – Replace history dataset name with the given string
    • genome_build (str) – Replace history dataset genome build (dbkey)

class bioblend.galaxy.objects.wrappers.Tool(t_dict, gi=None)
    Maps to a Galaxy tool.
    BASE_ATTRS = ('id', 'name', 'version')
    POLLING_INTERVAL = 10

    gi_module
run(inputs, history, wait=False, polling_interval=10)
    Execute this tool in the given history with inputs from dict inputs.
    
    Parameters
    • inputs (dict) – dictionary of input datasets and parameters for the tool (see below)
    • history (History) – the history where to execute the tool
    • wait (bool) – whether to wait while the returned datasets are in a pending state
    • polling_interval (float) – polling interval in seconds
    
    Return type list of HistoryDatasetAssociation
    Returns list of output datasets

    The inputs dict should contain input datasets and parameters in the (largely undocumented) format used by the Galaxy API. Some examples can be found in Galaxy's API test suite. The value of an input dataset can also be a Dataset object, which will be automatically converted to the needed format.

class bioblend.galaxy.objects.wrappers.Job(j_dict, gi=None)
    Maps to a Galaxy job.
    BASE_ATTRS = ('id', 'state')

    gi_module

class bioblend.galaxy.objects.wrappers.Preview(pw_dict, gi=None)
    Abstract base class for Galaxy entity 'previews'.
    
    Classes derived from this one model the short summaries returned by global getters such as /api/libraries.
    
    BASE_ATTRS = ('id', 'name', 'deleted')

class bioblend.galaxy.objects.wrappers.LibraryPreview(pw_dict, gi=None)
    Models Galaxy library 'previews'.
    
    Instances of this class wrap dictionaries obtained by getting /api/libraries from Galaxy.
    
    gi_module
**BioBlend Documentation, Release 0.12.0**

```python
class bioblend.galaxy.objects.wrappers.HistoryPreview(pw_dict, gi=None)
    Models Galaxy history ‘previews’.
    Instances of this class wrap dictionaries obtained by getting /api/histories from Galaxy.
    BASE_ATTRS = ('id', 'name', 'deleted', 'tags')
    gi_module
class bioblend.galaxy.objects.wrappers.WorkflowPreview(pw_dict, gi=None)
    Models Galaxy workflow ‘previews’.
    Instances of this class wrap dictionaries obtained by getting /api/workflows from Galaxy.
    BASE_ATTRS = ('id', 'name', 'deleted', 'published', 'tags')
    gi_module
```

### 5.2.3 Usage documentation

This page describes some sample use cases for the Galaxy API and provides examples for these API calls. In addition to this page, there are functional examples of complete scripts in the `docs/examples` directory of the BioBlend source code repository.

#### Connect to a Galaxy server

To connect to a running Galaxy server, you will need an account on that Galaxy instance and an API key for the account. Instructions on getting an API key can be found at [https://galaxyproject.org/develop/api/](https://galaxyproject.org/develop/api/).

To open a connection call:

```python
from bioblend.galaxy import GalaxyInstance

qi = GalaxyInstance(url='http://example.galaxy.url', key='your-API-key')
```

We now have a `GalaxyInstance` object which allows us to interact with the Galaxy server under our account, and access our data. If the account is a Galaxy admin account we also will be able to use this connection to carry out admin actions.

#### View Histories and Datasets

Methods for accessing histories and datasets are grouped under `GalaxyInstance.histories.*` and `GalaxyInstance.datasets.*` respectively.

To get information on the Histories currently in your account, call:

```python
>>> gi.histories.get_histories()
[{u'id': u'f3c2b0f3ecac9f02',
  u'name': u'RNAseq_DGE_BASIC_Prep',
  u'url': u'/api/histories/f3c2b0f3ecac9f02'},
 {u'id': u'8a91dcf1866a80c2',
  u'name': u'June demo',
  u'url': u'/api/histories/8a91dcf1866a80c2'}]
```

This returns a list of dictionaries containing basic metadata, including the id and name of each History. In this case, we have two existing Histories in our account, ‘RNAseq_DGE_BASIC_Prep’ and ‘June demo’. To get more detailed information about a History we can pass its id to the `show_history` method:
This gives us a dictionary containing the History’s metadata. With `contents=False` (the default), we only get a list of ids of the datasets contained within the History; with `contents=True` we would get metadata on each dataset. We can also directly access more detailed information on a particular dataset by passing its id to the `show_dataset` method:

```
>>> gi.datasets.show_dataset('10a4b652da44e82a')
{u'data_type': u'fastqsanger',
 'deleted': False,
 'file_size': 16527060,
 'genome_build': u'dm3',
 'id': 17499,
 'metadata_data_lines': None,
 'metadata_dbkey': u'dm3',
 'metadata_sequences': None,
 'misc_blurb': u'15.8 MB',
 'misc_info': u'None uploaded fastqsanger file',
 'model_class': u'HistoryDatasetAssociation',
 'name': u'C1_R2_1.chr4.fq',
 'purged': False,
```
Uploading Datasets to a History

To upload a local file to a Galaxy server, you can run the `upload_file` method, supplying the path to a local file:

```python
>>> gi.tools.upload_file('test.txt', 'f3c2b0f3ecac9f02')
{u'implicit_collections': [],
 u'jobs': [{u'create_time': u'2015-07-28T17:52:39.756488',
         u'exit_code': None,
         u'id': u'9752b387803d3e1e',
         u'model_class': u'Job',
         u'state': u'new',
         u'tool_id': u'upload1',
         u'update_time': u'2015-07-28T17:52:39.987509'}],
 u'output_collections': [],
 u'outputs': [{u'create_time': u'2015-07-28T17:52:39.331176',
               u'data_type': u'galaxy.datatypes.data.Text',
               u'deleted': False,
               u'file_ext': u'auto',
               u'file_size': 0,
               u'genome_build': u'?',
               u'hda_ldda': u'hda',
               u'hid': 16,
               u'history_content_type': u'dataset',
               u'history_id': u'f3c2b0f3ecac9f02',
               u'id': u'59c76a119581e190',
               u'metadata_data_lines': None,
               u'metadata_dbkey': u'?',
               u'misc_blurb': None,
               u'misc_info': None,
               u'model_class': u'HistoryDatasetAssociation',
               u'output_name': u'output0',
               u'peek': u'<table cellspacing="0" cellpadding="3"></table>',
               u'purged': False,
               u'state': u'queued',
               u'tags': [],
               u'uuid': u'ff0ee99b-7542-4125-802d-7a193f388e7e',
               u'visible': True}]
```

If files are greater than 2GB in size, they will need to be uploaded via FTP. Importing files from the user’s FTP folder can be done via running the upload tool again:

```python
>>> gi.tools.upload_from_ftp('test.txt', 'f3c2b0f3ecac9f02')
{u'implicit_collections': [],
 u'jobs': [{u'create_time': u'2015-07-28T17:52:39.331176',
         u'exit_code': None,
         u'id': u'9752b387803d3e1e',
         u'model_class': u'Job',
         u'state': u'new',
         u'tool_id': u'upload1',
         u'update_time': u'2015-07-28T17:52:39.987509'}],
 u'output_collections': [],
 u'outputs': [{u'create_time': u'2015-07-28T17:52:39.331176',
               u'data_type': u'galaxy.datatypes.data.Text',
               u'deleted': False,
               u'file_ext': u'auto',
               u'file_size': 0,
               u'genome_build': u'?',
               u'hda_ldda': u'hda',
               u'hid': 16,
               u'history_content_type': u'dataset',
               u'history_id': u'f3c2b0f3ecac9f02',
               u'id': u'59c76a119581e190',
               u'metadata_data_lines': None,
               u'metadata_dbkey': u'?',
               u'misc_blurb': None,
               u'misc_info': None,
               u'model_class': u'HistoryDatasetAssociation',
               u'output_name': u'output0',
               u'peek': u'<table cellspacing="0" cellpadding="3"></table>',
               u'purged': False,
               u'state': u'queued',
               u'tags': [],
               u'uuid': u'ff0ee99b-7542-4125-802d-7a193f388e7e',
               u'visible': True}]}
```
View Data Libraries

Methods for accessing Data Libraries are grouped under `GalaxyInstance.libraries.*`. Most Data Library methods are available to all users, but as only administrators can create new Data Libraries within Galaxy, the `create_folder` and `create_library` methods can only be called using an API key belonging to an admin account.

We can view the Data Libraries available to our account using:

```python
>>> gi.libraries.get_libraries()
[(u'id': u'8e6f930d00d123ea',
  u'name': u'RNA-seq workshop data',
  u'url': u'/api/libraries/8e6f930d00d123ea/'),
 (u'id': u'f740ab636b360a70',
  u'name': u'1000 genomes',
  u'url': u'/api/libraries/f740ab636b360a70')]
```

This gives a list of metadata dictionaries with basic information on each library. We can get more information on a particular Data Library by passing its id to the `show_library` method:

```python
>>> gi.libraries.show_library('8e6f930d00d123ea')
{u'contents_url': u'/api/libraries/8e6f930d00d123ea/contents',
 u'description': u'RNA-Seq workshop data',
 u'name': u'RNA-Seq',
 u'synopsis': u'Data for the RNA-Seq tutorial'}
```
Upload files to a Data Library

We can get files into Data Libraries in several ways: by uploading from our local machine, by retrieving from a URL, by passing the new file content directly into the method, or by importing a file from the filesystem on the Galaxy server.

For instance, to upload a file from our machine we might call:

```python
>>> gi.libraries.upload_file_from_local_path('8e6f930d00d123ea', '/local/path/to/mydata.fastq', file_type='fastqsanger')
```

Note that we have provided the id of the destination Data Library, and in this case we have specified the type that Galaxy should assign to the new dataset. The default value for `file_type` is `auto`, in which case Galaxy will attempt to guess the dataset type.

View Workflows

Methods for accessing workflows are grouped under `GalaxyInstance.workflows.*`.

To get information on the Workflows currently in your account, use:

```python
>>> gi.workflows.get_workflows()
[{'id': 'e8b85ad72aefca86',
  'name': 'TopHat + cufflinks part 1',
  'url': '/api/workflows/e8b85ad72aefca86'},
 {'id': 'b0631c44aa74526d',
  'name': 'CuffDiff',
  'url': '/api/workflows/b0631c44aa74526d'}]
```

This returns a list of metadata dictionaries. We can get the details of a particular Workflow, including its steps, by passing its id to the `show_workflow` method:

```python
>>> gi.workflows.show_workflow('e8b85ad72aefca86')
{'id': 'e8b85ad72aefca86',
 'inputs':
  {'252':
   {'label': 'Input RNA-seq fastq',
    'value': ''
  },
 'name': 'TopHat + cufflinks part 1',
 'steps':
  {'250':
   {'id': 250,
    'input_steps':
      {'input1':
        {'source_step': 252,
         'step_output': 'output'
      },
    'tool_id': 'tophat',
    'type': 'tool'
  },
   {'id': 251,
    'input_steps':
      {'input':
        ...
    },
  }
}
```

(continues on next page)
Export or import a workflow

Workflows can be exported from or imported into Galaxy. This makes it possible to archive workflows, or to move them between Galaxy instances.

To export a workflow, we can call:

```python
>>> workflow_dict = gi.workflows.export_workflow_dict('e8b85ad72aefca86')
```

This gives us a complex dictionary representing the workflow. We can import this dictionary as a new workflow with:

```python
>>> gi.workflows.import_workflow_dict(workflow_dict)
```

This call returns a dictionary containing basic metadata on the new workflow. Since in this case we have imported the dictionary into the original Galaxy instance, we now have a duplicate of the original workflow in our account:

```python
>>> gi.workflows.get_workflows()  
[{'id': 'c0bacafdfe211f9a', 'name': 'TopHat + cufflinks part 1 (imported from API)', 'url': '/api/workflows/c0bacafdfe211f9a'}, {u'id': u'e8b85ad72aefca86', 'name': u'TopHat + cufflinks part 1 (imported from API)', 'url': u'/api/workflows/e8b85ad72aefca86'}, {u'id': u'b0631c44aa74526d', 'name': u'CuffDiff', 'url': u'/api/workflows/b0631c44aa74526d'}]
```

Instead of using dictionaries directly, workflows can be exported to or imported from files on the local disk using the `export_workflow_to_local_path` and `import_workflow_from_local_path` methods. See the API reference for details.

**Note:** If we export a workflow from one Galaxy instance and import it into another, Galaxy will only run it without modification if it has the same versions of the tool wrappers installed. This is to ensure reproducibility. Otherwise, we will need to manually update the workflow to use the new tool versions.

5.2. Galaxy API
Run a Workflow

To run a Workflow, we need to tell Galaxy which datasets to use for which workflow inputs. We can use datasets from Histories or Data Libraries.

Examine the Workflow above. We can see that it takes only one input file. That is:

```
>>> wf = gi.workflows.show_workflow('e8b85ad72aefca86')
>>> wf['inputs']
{u'252':
    {u'label':
        u'Input RNA-seq fastq',
        u'value': u''
    }
}
```

There is one input, labelled ‘Input RNA-seq fastq’. This input is passed to the Tophat tool and should be a fastq file. We will use the dataset we examined above, under **View Histories and Datasets**, which had name ‘C1_R2_1.chr4.fq’ and id ‘10a4b652da44e82a’.

To specify the inputs, we build a data map and pass this to the `run_workflow` method. This data map is a nested dictionary object which maps inputs to datasets. We call:

```
>>> datamap = dict()
>>> datamap['252'] = { 'src': 'hda', 'id': '10a4b652da44e82a' }
>>> gi.workflows.run_workflow('e8b85ad72aefca86', datamap, history_name='New output history')
```

In this case the only input id is ‘252’ and the corresponding dataset id is ‘10a4b652da44e82a’. We have specified the dataset source to be ‘hda’ (HistoryDatasetAssociation) since the dataset is stored in a History. See the **API reference** for allowed dataset specifications. We have also requested that a new History be created and used to store the results of the run, by setting `history_name='New output history'`.

The `run_workflow` call submits all the jobs which need to be run to the Galaxy workflow engine, with the appropriate dependencies so that they will run in order. The call returns immediately, so we can continue to submit new jobs while waiting for this workflow to execute. `run_workflow` returns the id of the output History and of the datasets that will be created as a result of this run. Note that these dataset ids are valid immediately, so we can specify these datasets as inputs to new jobs even before the files have been created, and the new jobs will be added to the queue with the appropriate dependencies.

If we view the output History immediately after calling `run_workflow`, we will see something like:
In this case, because the submitted jobs have not had time to run, the output History contains 8 datasets in the ‘queued’ state and has a total size of 0 bytes. If we make this call again later we should instead see completed output files.

View Users

Methods for managing users are grouped under `GalaxyInstance.users.*`. User management is only available to Galaxy administrators, that is, the API key used to connect to Galaxy must be that of an admin account.

To get a list of users, call:

```python
>>> gi.users.get_users()
[{'email': 'userA@unimelb.edu.au',
 u'id': '975a9ce09b49502a',
 u'quota_percent': None,
 u'url': u'/api/users/975a9ce09b49502a'},
 {'email': 'userB@student.unimelb.edu.au',
 u'id': '0193a95acf427d2c',
 u'quota_percent': None,
 u'url': u'/api/users/0193a95acf427d2c'}]
```
5.3 Toolshed API

API used to interact with the Galaxy Toolshed, including repository management.

5.3.1 API documentation for interacting with the Galaxy Toolshed

**ToolShedInstance**

```python
class bioblend.toolshed.ToolShedInstance(url, key=None, email=None, password=None, verify=True)
```

A base representation of a connection to a ToolShed instance, identified by the ToolShed URL and user credentials.

After you have created a `ToolShedInstance` object, access various modules via the class fields. For example, to work with repositories and get a list of all public repositories, the following should be done:

```python
from bioblend import toolshed
ts = toolshed.ToolShedInstance(url='https://testtoolshed.g2.bx.psu.edu')
rl = ts.repositories.get_repositories()
tools = ts.tools.search_tools('fastq')
```

**Parameters**

- `url (str)` – A FQDN or IP for a given instance of ToolShed. For example: `https://testtoolshed.g2.bx.psu.edu`. If a ToolShed instance is served under a prefix (e.g. `http://127.0.0.1:8080/toolshed/`), supply the entire URL including the prefix (note that the prefix must end with a slash).

- `key (str)` – If required, user’s API key for the given instance of ToolShed, obtained from the user preferences.

- `email (str)` – ToolShed e-mail address corresponding to the user. Ignored if key is supplied directly.

- `password (str)` – Password of ToolShed account corresponding to the above e-mail address. Ignored if key is supplied directly.

- `verify (bool)` – Whether to verify the server’s TLS certificate

`__init__(url, key=None, email=None, password=None, verify=True)`

A base representation of a connection to a ToolShed instance, identified by the ToolShed URL and user credentials.

After you have created a `ToolShedInstance` object, access various modules via the class fields. For example, to work with repositories and get a list of all public repositories, the following should be done:

```python
from bioblend import toolshed
ts = toolshed.ToolShedInstance(url='https://testtoolshed.g2.bx.psu.edu')
rl = ts.repositories.get_repositories()
tools = ts.tools.search_tools('fastq')
```
Parameters

- **url** *(str)* – A FQDN or IP for a given instance of ToolShed. For example: https://testtoolshed.g2.bx.psu.edu. If a ToolShed instance is served under a prefix (e.g. http://127.0.0.1:8080/toolshed/), supply the entire URL including the prefix (note that the prefix must end with a slash).

- **key** *(str)* – If required, user’s API key for the given instance of ToolShed, obtained from the user preferences.

- **email** *(str)* – ToolShed e-mail address corresponding to the user. Ignored if key is supplied directly.

- **password** *(str)* – Password of ToolShed account corresponding to the above e-mail address. Ignored if key is supplied directly.

- **verify** *(bool)* – Whether to verify the server’s TLS certificate

Categories

Interaction with a Tool Shed instance categories

**class bioblend.toolshed.categories.ToolShedCategoryClient**(toolshed_instance)

**get_categories**(deleted=False)

Returns a list of dictionaries that contain descriptions of the repository categories found on the given Tool Shed instance.

**Parameters**

deleted *(bool)* – whether to show deleted categories

**Return type**

list

**Returns**

A list of dictionaries containing information about repository categories present in the Tool Shed. For example:

```
[{u'deleted': False,
  u'description': u'Tools for manipulating data',
  u'id': u'175812cd7caaf439',
  u'model_class': u'Category',
  u'name': u'Text Manipulation',
  u'url': u'/api/categories/175812cd7caaf439'}]
```

New in version 0.5.2.

**show_category**(category_id)

Get details of a given category.

**Parameters**

category_id *(str)* – Encoded category ID

**Return type**

dict

**Returns**

details of the given category

Repositories

Interaction with a Tool Shed instance repositories
class bioblend.toolshed.repositories.ToolShedRepositoryClient(toolshed_instance)

create_repository(name, synopsis, description=None, type='unrestricted', remote_repository_url=None, homepage_url=None, category_ids=None)
Create a new repository in a Tool Shed.

Parameters

• name (str) – Name of the repository
• synopsis (str) – Synopsis of the repository
• description (str) – Optional description of the repository
• type (str) – type of the repository. One of “unrestricted”, “repository_suite_definition”, or “tool_dependency_definition”
• remote_repository_url (str) – Remote URL (e.g. GitHub/Bitbucket repository)
• homepage_url (str) – Upstream’s homepage for the project
• category_ids (list) – List of encoded category IDs

Return type dict

Returns

a dictionary containing information about the new repository. For example:

```python
{"deleted": false,
"deprecated": false,
"description": "new_synopsis",
"homepage_url": "https://github.com/galaxyproject/",
"id": "8cf91205f2f737f4",
"long_description": "this is some repository",
"model_class": "Repository",
"name": "new_repo_17",
"owner": "qqqqqq",
"private": false,
"remote_repository_url": "https://github.com/galaxyproject/tools-devteam",
"times_downloaded": 0,
"type": "unrestricted",
"user_id": "adb5f5c93f827949"}
```

get_ordered_installable_revisions(name, owner)

Returns the ordered list of changeset revision hash strings that are associated with installable revisions. As in the changelog, the list is ordered oldest to newest.

Parameters

• name (str) – the name of the repository
• owner (str) – the owner of the repository

Return type list

Returns List of changeset revision hash strings from oldest to newest

get_repositories()

Get a list of all the repositories in a Galaxy Tool Shed.

Return type list
Returns

Returns a list of dictionaries containing information about repositories present in the Tool Shed. For example:

```json
[{u'category_ids': [u'c1df3132f6334b0e', u'f6d7b0037d901d9b'],
  u'deleted': False,
  u'deprecated': False,
  u'description': u'Order Contigs',
  u'homepage_url': u'',
  u'id': u'287bd69f724b99ce',
  u'name': u'best_tool_ever',
  u'owner': u'billybob',
  u'private': False,
  u'remote_repository_url': u'',
  u'times_downloaded': 0,
  u'type': u'unrestricted',
  u'url': u'/api/repositories/287bd69f724b99ce',
  u'user_id': u'5cefd48bc04af6d4'}]
```

Changed in version 0.4.1: Changed method name from `get_tools` to `get_repositories` to better align with the Tool Shed concepts.

`get_repository_revision_install_info(name, owner, changeset_revision)`
Return a list of dictionaries of metadata about a certain changeset revision for a single tool.

Parameters

- **name** (str) – the name of the repository
- **owner** (str) – the owner of the repository
- **changeset_revision** (str) – the changeset_revision of the RepositoryMetadata object associated with the repository

Return type List of dictionaries

Returns

Returns a list of the following dictionaries:

1. a dictionary defining the repository
2. a dictionary defining the repository revision (RepositoryMetadata)
3. a dictionary including the additional information required to install the repository

For example:

```json
[{u'deleted': False,
  u'deprecated': False,
  u'description': u'Galaxy Freebayes Bayesian genetic variant detector tool',
  u'homepage_url': u'',
  u'id': u'491b7a3fddf9366f',
  u'long_description': u'Galaxy Freebayes Bayesian genetic variant detector tool originally included in the Galaxy code distribution but migrated to the tool shed.',
  u'name': u'freebayes',
  u'owner': u'devteam',
  u'private': False,
  u'remote_repository_url': u''}]
```

(continues on next page)
repository_revisions (downloadable=None, malicious=None, tools_functionally_correct=None, missing_test_components=None, do_not_test=None, includes_tools=None, test_install_error=None, skip_tool_test=None)

Returns a (possibly filtered) list of dictionaries that include information about all repository revisions. The following parameters can be used to filter the list.

Parameters

- downloadable (bool) – Can the tool be downloaded
- malicious (bool) –
- tools_functionally_correct (bool) –
- missing_test_components (bool) –
- do_not_test (bool) –
- includes_tools (bool) –
- test_install_error (bool) –
skip_tool_test (bool) –

Return type List of dictionaries

Returns

Returns a (possibly filtered) list of dictionaries that include information about all repository revisions. For example:

```python
{{'changeset_revision': u'6e26c5a48e9a',
'do_not_test': False,
'downloadable': True,
'has_repository_dependencies': False,
'id': u'92250aff777a169',
'includes_datatypes': False,
'includes_tool_dependencies': False,
'includes_tools': True,
'includes_tools_for_display_in_tool_panel': True,
'includes_workflows': False,
'malicious': False,
'missing_test_components': False,
'repository_id': u'78f2604ff5e65707',
'test_install_error': False,
'time_last_tested': None,
'tools_functionally_correct': False,
'url': u'/api/repository_revisions/92250aff777a169'},
{{'changeset_revision': u'15a54fa11ad7',
'do_not_test': False,
'downloadable': True,
'has_repository_dependencies': False,
'id': u'd3823c748ae2205d',
'includes_datatypes': False,
'includes_tool_dependencies': False,
'includes_tools': True,
'includes_tools_for_display_in_tool_panel': True,
'includes_workflows': False,
'malicious': False,
'missing_test_components': False,
'repository_id': u'f9662009da7bfce0',
'test_install_error': False,
'time_last_tested': None,
'tools_functionally_correct': False,
'url': u'/api/repository_revisions/d3823c748ae2205d'}}
```

**search_repositories (q, page=1, page_size=10)**

Search for repositories in a Galaxy Tool Shed.

Parameters

- **q (str)** – query string for searching purposes
- **page (int)** – page requested
- **page_size (int)** – page size requested

Return type dict

Returns

dictionary containing search hits as well as metadata for the search. For example:
show_repository (toolShed_id)
Display information of a repository from Tool Shed

Parameters

- **toolShed_id** *(str)* – Encoded Tool Shed ID

Return type

- **dict**

Returns

Information about the tool. For example:

```python
{u'category_ids': [u'c1df3132f6334b0e', u'f6d7b0037d01d9b'],
 u'deleted': False,
 u'deprecated': False,
 u'description': u'Order Contigs',
 u'homepage_url': u'',
 u'id': u'287bd69f724b99ce',
 u'long_description': u'',
 u'name': u'best_tool_ever',
 u'owner': u'billybob',
 u'private': False,
 u'remote_repository_url': u'',
 u'times_downloaded': 0,
 u'type': u'unrestricted',
}
```
Changed in version 0.4.1: Changed method name from `show_tool` to `show_repository` to better align with the Tool Shed concepts.

**`show_repository_revision`** *(`metadata_id`)*

Returns a dictionary that includes information about a specified repository revision.

**Parameters**

`metadata_id` *(str)* – Encoded repository metadata ID

**Return type**

dict

**Returns**

Returns a dictionary that includes information about a specified repository revision. For example:

```python
{u'changeset_revision': u'7602de1e7f32',
 u'do_not_test': False,
 u'downloadable': True,
 u'has_repository_dependencies': False,
 u'id': u'504be8aaa652c154',
 u'included_datatypes': False,
 u'includes_tool_dependencies': False,
 u'included_tools': True,
 u'includes_tools_for_display_in_tool_panel': True,
 u'included_workflows': False,
 u'malicious': False,
 u'missing_test_components': True,
 u'repository_id': u'491b7a33fddf9366f',
 u'test_install_error': False,
 u'time_last_tested': None,
 u'tool_test_results': {u'missing_test_components': []},
 u'tools_functionally_correct': False,
 u'url': u'/api/repository_revisions/504be8aaa652c154'}
```

**update_repository** *(`id`, `tar_ball_path`, `commit_message`=None)*

Update the contents of a Tool Shed repository with specified tar ball.

**Parameters**

- `id` *(str)* – Encoded repository ID
- `tar_ball_path` *(str)* – Path to file containing tar ball to upload.
- `commit_message` *(str)* – Commit message used for the underlying Mercurial repository backing Tool Shed repository.

**Return type**

dict

**Returns**

Returns a dictionary that includes repository content warnings. Most valid uploads will result in no such warning and an exception will be raised generally if there are problems. For example a successful upload will look like:

```python
{u'content_alert': u'',
 u'message': u''}
```

New in version 0.5.2.
Tools

Interaction with a Tool Shed instance tools

```python
class bioblend.toolshed.tools.ToolShedToolClient(toolshed_instance)

search_tools(q, page=1, page_size=10)
```

Search for tools in a Galaxy Tool Shed.

**Parameters**

- `q` *(str)* – query string for searching purposes
- `page` *(int)* – page requested
- `page_size` *(int)* – page size requested

**Return type** *dict*

**Returns**

dictionary containing search hits as well as metadata for the search. For example:

```
{u'hits': [{u'matched_terms': [],
            u'score': 3.0,
            u'tool': {u'description': u'convert between various FASTQ
                        quality formats',
                      u'id': u'69819b84d55f521efda001e0926e7233',
                      u'name': u'FASTQ Groomer',
                      u'repo_name': None,
                      u'repo_owner_username': u'devteam'}},
   {u'matched_terms': [],
    u'score': 3.0,
    u'tool': {u'description': u'converts a bam file to fastq files.',
             u'id': u'521e282770f69522e452c72551b',
             u'name': u'Defuse BamFastq',
             u'repo_name': None,
             u'repo_owner_username': u'jjohnson'}},
   u'hostname': u'https://testtoolshed.g2.bx.psu.edu/',
   u'page': u'1',
   u'page_size': u'2',
   u'total_results': u'118'}
```
BioBlend allows library-wide configuration to be set in external files. These configuration files can be used to specify access keys, for example.

## 6.1 Configuration documents for BioBlend

### 6.1.1 BioBlend

**exception bioblend.ConnectionError** *(message, body=None, status_code=None)*

An exception class that is raised when unexpected HTTP responses come back.

Should make it easier to debug when strange HTTP things happen such as a proxy server getting in the way of the request etc. @see: body attribute to see the content of the http response

**class bioblend.NullHandler**(level=0)

Initializes the instance - basically setting the formatter to None and the filter list to empty.

**emit**(record)

Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplmentedError.

**bioblend.get_version()**

Returns a string with the current version of the library (e.g., “0.2.0”)

**bioblend.init_logging()**

Initialize BioBlend’s logging from a configuration file.

**bioblend.set_file_logger**(name, filepath, level=20, format_string=None)

**bioblend.set_stream_logger**(name, level=10, format_string=None)
6.1.2 Config

class bioblend.config.Config(path=None, fp=None, do_load=True)

BioBlend allows library-wide configuration to be set in external files. These configuration files can be used to
specify access keys, for example. By default we use two locations for the BioBlend configurations:

• System wide: /etc/bioblend.cfg
• Individual user: ~/.bioblend (which works on both Windows and Unix)

get (section, name, default=None)
get_value (section, name, default=None)
getbool (section, name, default=False)
getfloat (section, name, default=0.0)
getint (section, name, default=0)
If you’d like to do more than just a mock test, you’ll want to point BioBlend to an instance of Galaxy. Do so by exporting the following two variables:

```
$ export BIOBLEND_GALAXY_URL=http://127.0.0.1:8080
$ export BIOBLEND_GALAXY_API_KEY=<API key>
```

The unit tests, stored in the tests folder, can be run using pytest. From the project root:

```
$ pytest
```
If you’ve run into issues, found a bug, or can’t seem to find an answer to your question regarding the use and functionality of BioBlend, please use Github Issues page to ask your question.
Related documentation

Links to other documentation and libraries relevant to this library:

• Galaxy API documentation
• Blend4j: Galaxy API wrapper for Java
• clj-blend: Galaxy API wrapper for Clojure
CHAPTER 10

Indices and tables

• genindex
• modindex
• search
Python Module Index

b
bioblend, 97
bioblend.cloudman, 12
bioblend.config, 98
bioblend.galaxy.config, 20
bioblend.galaxy.datasets, 21
bioblend.galaxy.datatypes, 22
bioblend.galaxy.folders, 23
bioblend.galaxy.forms, 25
bioblend.galaxy.ftpfiles, 25
bioblend.galaxy.genomes, 26
bioblend.galaxy.groups, 27
bioblend.galaxy.histories, 29
bioblend.galaxy.jobs, 36
bioblend.galaxy.libraries, 37
bioblend.galaxy.objects.client, 65
bioblend.galaxy.objects.wrappers, 69
bioblend.galaxy.quotas, 43
bioblend.galaxy.roles, 45
bioblend.galaxy.tool_data, 50
bioblend.galaxy.tools, 46
bioblend.galaxy.toolshed, 51
bioblend.galaxy.users, 53
bioblend.galaxy.visual, 55
bioblend.galaxy.workflows, 56
bioblend.toolshed.categories, 89
bioblend.toolshed.repositories, 89
bioblend.toolshed.tools, 96
Symbols

__init__() (bioblend.galaxy.GalaxyInstance method), 20
__init__() (bioblend.toolshed.ToolShedInstance method), 88

A
add_group_role() (bioblend.galaxy.groups.GroupsClient method), 27
add_group_user() (bioblend.galaxy.groups.GroupsClient method), 27
add_nodes() (bioblend.cloudman.CloudManInstance method), 14
adjust_autoscaling() (bioblend.cloudman.CloudManInstance method), 14
API_MODULE (bioblend.galaxy.objects.wrappers.History attribute), 72
API_MODULE (bioblend.galaxy.objects.wrappers.Library attribute), 74
assign_floating_ip() (bioblend.cloudman.launch.CloudManLauncher method), 9
autoscaling_enabled() (bioblend.cloudman.CloudManInstance method), 14

B
BASE_ATTRS (bioblend.galaxy.objects.wrappers.ContentInfo attribute), 71
BASE_ATTRS (bioblend.galaxy.objects.wrappers.Dataset attribute), 76
BASE_ATTRS (bioblend.galaxy.objects.wrappers.DatasetCollection attribute), 78
BASE_ATTRS (bioblend.galaxy.objects.wrappers.DatasetContainer attribute), 72
BASE_ATTRS (bioblend.galaxy.objects.wrappers.LibraryDatasetDatasetAssociation attribute), 78
BASE_ATTRS (bioblend.galaxy.objects.wrappers.Library attribute), 74
BASE_ATTRS (bioblend.galaxy.objects.wrappers.LibraryDatasetCollectionAssociation attribute), 78
BASE_ATTRS (bioblend.galaxy.objects.wrappers.LibraryDatasetContainer attribute), 78
BASE_ATTRS (bioblend.galaxy.objects.wrappers.LibraryDataset file attribute), 76
BASE_ATTRS (bioblend.galaxy.objects.wrappers.LibraryDatasetFolder attribute), 76
BASE_ATTRS (bioblend.galaxy.objects.wrappers.LibraryPreview attribute), 80
BASE_ATTRS (bioblend.galaxy.objects.wrappers.LibraryStep attribute), 69
BASE_ATTRS (bioblend.galaxy.objects.wrappers.LibraryWorkflow attribute), 70
BASE_ATTRS (bioblend.galaxy.objects.wrappers.LibraryWorkflowPreview attribute), 80
BASE_ATTRS (bioblend.galaxy.objects.wrappers.LibraryWrapper attribute), 69

bioblend (module), 97
bioblend.cloudman (module), 12
bioblend.config (module), 98
bioblend.galaxy (module), 20
bioblend.galaxy.datasets (module), 21
bioblend.galaxy.datatypes (module), 22
bioblend.galaxy.folders (module), 23
bioblend.galaxy.forms (module), 25
bioblend.galaxy.ftpfiles (module), 25
bioblend.galaxy.genomes (module), 26
bioblend.galaxy.groups (module), 27
bioblend.galaxy.histories (module), 29
bioblend.galaxy.jobs (module), 36
bioblend.galaxy.libraries (module), 37
bioblend.galaxy.objects.client (module), 65
bioblend.galaxy.objects.wrappers (module), 69
bioblend.galaxy.quotas (module), 43
bioblend.galaxy.roles (module), 45
bioblend.galaxy.tool_data (module), 50
bioblend.galaxy.tools (module), 46
bioblend.galaxy.toolshed (module), 51
bioblend.galaxy.users (module), 53
bioblend.galaxy.visual (module), 55
bioblend.galaxy.workflows (module), 56
bioblend.toolshed.categories (module), 89
bioblend.toolshed.repositories (module), 89
bioblend.toolshed.tools (module), 96
block_until_vm_ready() (in module bioblend.cloudman), 16

C

cancel_invocation()
(bioblend.galaxy.workflows.WorkflowClient method), 56
clone() (bioblend.galaxy.objects.wrappers.Wrapper method), 69
cloudman_url (bioblend.cloudman.CloudManInstance attribute), 14
CloudManConfig (class in bioblend.cloudman), 12
CloudManConfig.CustomTypeEncoder (class in bioblend.cloudman), 13
CloudManInstance (class in bioblend.cloudman), 14
CloudManLauncher (class in bioblend.cloudman.launch), 9
Config (class in bioblend.config), 98
ConfigClient (class in bioblend.galaxy.config), 20
connect_ec2() (bioblend.cloudman.launch.CloudManLauncher method), 9
connect_s3() (bioblend.cloudman.launch.CloudManLauncher method), 10
connect_vpc() (bioblend.cloudman.launch.CloudManLauncher method), 10
ConnectionError, 97
container_id (bioblend.galaxy.objects.wrappers.Dataset attribute), 76
container_id (bioblend.galaxy.objects.wrappers.Folder attribute), 76
CONTENT_INFO_TYPE (bioblend.galaxy.objects.wrappers.History attribute), 72
CONTENT_INFO_TYPE (bioblend.galaxy.objects.wrappers.Library attribute), 74
ContentInfo (class in bioblend.galaxy.objects.wrappers), 71
convert_input_map()
(bioblend.galaxy.objects.wrappers.Workflow method), 70
copy_from_dataset()
(bioblend.galaxy.libraries.LibraryClient method), 37
copy_from_dataset()
(bioblend.galaxy.objects.wrappers.Library method), 75
create() (bioblend.galaxy.objects.client.ObjHistoryClient method), 65
create() (bioblend.galaxy.objects.client.ObjLibraryClient method), 66
create_cm_security_group()
(bioblend.cloudman.launch.CloudManLauncher method), 10
create_dataset_collection()
(bioblend.galaxy.histories.HistoryClient method), 29
create_dataset_collection()
(bioblend.galaxy.objects.wrappers.History method), 73
create_folder() (bioblend.galaxy.folders.FoldersClient method), 23
create_folder() (bioblend.galaxy.libraries.LibraryClient method), 38
create_folder() (bioblend.galaxy.objects.wrappers.Library method), 75
create_form() (bioblend.galaxy.forms.FormsClient method), 25
create_group() (bioblend.galaxy.groups.GroupsClient method), 27
create_history() (bioblend.galaxy.histories.HistoryClient method), 29
create_history_tag()
(bioblend.galaxy.histories.HistoryClient method), 30
create_key_pair()
(bioblend.cloudman.launch.CloudManLauncher method), 10
create_library() (bioblend.galaxy.libraries.LibraryClient method), 38
create_local_user()
(bioblend.galaxy.users.UserClient method), 53
create_quota() (bioblend.galaxy.quotas.QuotaClient method), 43
create_remote_user()
(bioblend.galaxy.users.UserClient method), 53
create_repository()
(bioblend.toolshed.repositories.ToolShedRepositoryClient method), 90
create_role()
(bioblend.galaxy.roles.RolesClient method), 110
create_user_apikey() (bioblend.galaxy.users.UserClient method), 53

CustomTypeDecoder() (bioblend.cloudman.CloudManConfig static method), 13

data_collection_input_ids (bioblend.galaxy.objects.wrappers.Workflow attribute), 70

data_input_ids (bioblend.galaxy.objects.wrappers.Workflow attribute), 70

Dataset (class in bioblend.galaxy.objects.wrappers), 76
dataset_ids (bioblend.galaxy.objects.wrappers.DatasetContainer attribute), 72

DatasetClient (class in bioblend.galaxy.datasets), 21

DatasetCollection (class in bioblend.galaxy.objects.wrappers), 78

DatasetContainer (class in bioblend.galaxy.objects.wrappers), 72

DatasetStateException, 22

DatasetTimeoutException, 22

datatypesClient (class in bioblend.galaxy.datatypes), 22
default() (bioblend.cloudman.CloudManConfig.CustomTypeEncoder method), 13
delete() (bioblend.galaxy.objects.client.ObjHistoryClient method), 65
delete() (bioblend.galaxy.objects.client.ObjLibraryClient method), 66
delete() (bioblend.galaxy.objects.client.ObjWorkflowClient method), 68
delete() (bioblend.galaxy.objects.wrappers.Dataset method), 73
delete() (bioblend.galaxy.objects.wrappers.HistoryDatasetAssociation method), 77
delete() (bioblend.galaxy.objects.wrappers.HistoryDatasetCollectionAssociation method), 78
delete() (bioblend.galaxy.objects.wrappers.Library method), 74
delete() (bioblend.galaxy.objects.wrappers.LibraryDataset method), 79
delete() (bioblend.galaxy.objects.wrappers.Workflow method), 70
delete_dataset() (bioblend.galaxy.histories.HistoryClient method), 30

delete_dataset_collection() (bioblend.galaxy.histories.HistoryClient method), 30
delete_folder() (bioblend.galaxy.folders.FoldersClient method), 23
delete_group_role() (bioblend.galaxy.groups.GroupsClient method), 27
delete_group_user() (bioblend.galaxy.groups.GroupsClient method), 28
delete_history() (bioblend.galaxy.histories.HistoryClient method), 30
delete_library() (bioblend.galaxy.libraries.LibraryClient method), 38
delete_library_dataset() (bioblend.galaxy.libraries.LibraryClient method), 38
delete_quota() (bioblend.galaxy.quotas.QuotaClient method), 43

delete_user() (bioblend.galaxy.users.UserClient method), 53
delete_workflow() (bioblend.galaxy.workflows.WorkflowClient method), 56
disable_autoscaling() (bioblend.cloudman.CloudManInstance method), 14
download() (bioblend.galaxy.objects.wrappers.Dataset method), 77
download() (bioblend.galaxy.objects.wrappers.History method), 73
download_dataset() (bioblend.galaxy.datasets.DatasetClient method), 21
download_dataset() (bioblend.galaxy.histories.HistoryClient method), 31
download_history() (bioblend.galaxy.histories.HistoryClient method), 31

DS_TYPE (bioblend.galaxy.objects.wrappers.HistoryDatasetAssociation attribute), 73

dS_TYPE (bioblend.galaxy.objects.wrappers.HistoryDatasetCollectionAssociation attribute), 73

dS_TYPE (bioblend.galaxy.objects.wrappers.Library attribute), 74

dS_TYPE (bioblend.galaxy.objects.wrappers.LibraryDataset attribute), 73

dS_TYPE (bioblend.galaxy.objects.wrappers.History attribute), 73

emit() (bioblend_NullHandler method), 97
enable_autoscaling() (bioblend.cloudman.CloudManInstance method), 14
export() (bioblend.galaxy.objects.wrappers.History method), 73

Index 111
export() (bioblend.galaxy.objects.wrappers.Workflow method), 70
export_history() (bioblend.galaxy.histories.HistoryClient method), 31
export_workflow_dict() (bioblend.galaxy.workflows.WorkflowClient method), 56
export_workflow_json() (bioblend.galaxy.workflows.WorkflowClient method), 56
export_workflow_to_local_path() (bioblend.galaxy.workflows.WorkflowClient method), 56
find_placements() (bioblend.cloudman.launch.CloudManLauncher method), 10
Folder (class in bioblend.galaxy.objects.wrappers), 76
collection (bioblend.galaxy.objects.wrappers.Library attribute), 75
FoldersClient (class in bioblend.galaxy.folders), 23
FormsClient (class in bioblend.galaxy.forms), 25
from_json() (bioblend.galaxy.objects.wrappers.Wrapper class method), 69
FTPFilesClient (class in bioblend.galaxy.ftpfiles), 25
galaxy_url (bioblend.cloudman.CloudManInstanceId attribute), 14
GalaxyInstance (class in bioblend.galaxy), 19
GalaxyInstance (class in bioblend.galaxy.objects.galaxy_instance), 64
GenericVMInstance (class in bioblend.cloudman), 16
GenomeClient (class in bioblend.galaxy.genomes), 26
get() (bioblend.config.Config method), 98
get() (bioblend.galaxy.objects.client.ObjHistoryClient method), 65
get() (bioblend.galaxy.objects.client.ObjJobClient method), 66
get() (bioblend.galaxy.objects.client.ObjLibraryClient method), 67
get() (bioblend.galaxy.objects.client.ObjToolClient method), 67
get() (bioblend.galaxy.objects.client.ObjWorkflowClient method), 68
get_categories() (bioblend.toolshed.categories.ToolShedCategoryClient method), 89
get_cloudman_version() (bioblend.cloudman.CloudManInstance method), 15
get_cluster_pd() (bioblend.cloudman.launch.CloudManLauncher method), 11
get_cluster_size() (bioblend.cloudman.CloudManInstanceId method), 15
get_cluster_type() (bioblend.cloudman.CloudManInstance method), 15
get_clusters_pd() (bioblend.cloudman.launch.CloudManLauncher method), 11
get_config() (bioblend.galaxy.config.ConfigClient method), 20
get_contents() (bioblend.galaxy.objects.wrappers.Dataset method), 77
get_current_history() (bioblend.galaxy.histories.HistoryClient method), 31
get_current_user() (bioblend.galaxy.users.UserClient method), 54
get_data_tables() (bioblend.galaxy.tool_data.ToolDataClient method), 50
get_dataset() (bioblend.galaxy.objects.wrappers.DatasetContainer method), 72
get_dataset_collection() (bioblend.galaxy.objects.wrappers.Dataset method), 73
get_datasets() (bioblend.galaxy.objects.wrappers.DatasetContainer method), 72
get_datatypes() (bioblend.galaxy.datatypes.DatatypesClient method), 22
get_folder() (bioblend.galaxy.objects.wrappers.Library method), 22
get_folders() (bioblend.galaxy.libraries.LibraryClient method), 25
get_forms() (bioblend.galaxy.forms.FormsClient method), 25
get_ftp_files() (bioblend.galaxy.ftpfiles.FTPFilesClient method), 25
get_galaxy_state() (bioblend.cloudman.CloudManInstance method), 15
get_genomes() (bioblend.galaxy.genomes.GenomeClient method), 26
get_group_roles() (bioblend.galaxy.groups.GroupsClient method), 28
get_group_users() (bioblend.galaxy.groups.GroupsClient method), 28
get_groups() (bioblend.galaxy.groups.GroupsClient method), 28
get_histories() (bioblend.galaxy.histories.HistoryClient method), 28
list() (bioblend.galaxy.objects.client.ObjHistoryClient method), 66
list() (bioblend.galaxy.objects.client.ObjJobClient method), 66
list() (bioblend.galaxy.objects.client.ObjLibraryClient method), 67
list() (bioblend.galaxy.objects.client.ObjToolClient method), 67
list() (bioblend.galaxy.objects.client.ObjWorkflowClient method), 68
load_config() (bioblend.cloudman.CloudManConfig static method), 14
load_config() (bioblend.galaxy.GalaxyInstance attribute), 20
max_get_attempts (bioblend.galaxy.GalaxyInstance attribute), 20
NullHandler (class in bioblend), 97
ObjClient (class in bioblend.galaxy.objects.client), 65
ObjDatasetContainerClient (class in bioblend.galaxy.objects.client), 65
ObjHistoryClient (class in bioblend.galaxy.objects.client), 65
ObjJobClient (class in bioblend.galaxy.objects.client), 66
ObjLibraryClient (class in bioblend.galaxy.objects.client), 66
ObjToolClient (class in bioblend.galaxy.objects.client), 67
ObjWorkflowClient (class in bioblend.galaxy.objects.client), 68
parent (bioblend.galaxy.objects.wrappers.Folder attribute), 76
parent (bioblend.galaxy.objects.wrappers_WRAPPER attribute), 69
paste_content() (bioblend.galaxy.objects.wrappers.History method), 73
paste_content() (bioblend.galaxy.tools.ToolClient method), 47
peek() (bioblend.galaxy.objects.wrappers.Dataset method), 77
POLLING_INTERVAL (bioblend.galaxy.objects.wrappers.Dataset attribute), 76
POLLING_INTERVAL (bioblend.galaxy.objects.wrappers.Tool attribute), 79
POLLING_INTERVAL (bioblend.galaxy.objects.wrappers.Workflow attribute), 70
Preview (class in bioblend.galaxy.objects.wrappers), 79
preview() (bioblend.galaxy.objects.wrappers.DatasetContainer method), 72
preview() (bioblend.galaxy.objects.wrappers.Workflow method), 70
put_url() (bioblend.galaxy.tools.ToolClient method), 47
QuotaClient (class in bioblend.galaxy.quotas), 43
reboot_node() (bioblend.cloudman.CloudManInstance method), 15
refresh() (bioblend.galaxy.objects.wrappers.Dataset method), 77
refresh() (bioblend.galaxy.objects.wrappers.DatasetCollection method), 78
refresh() (bioblend.galaxy.objects.wrappers.DatasetContainer method), 72
refresh() (bioblend.galaxy.objects.wrappers.Folder method), 76
reload_data_table() (bioblend.galaxy.tool_data.ToolDataClient method), 50
in remove_node() (bioblend.cloudman.CloudManInstance method), 15
in remove_nodes() (bioblend.cloudman.CloudManInstance method), 15
in repository_revisions() (bioblend.toolshed.repositoriea.ToolShedRepositoryClient method), 92
in root_folder (bioblend.galaxy.objects.wrappers.Library attribute), 75
rule_exists() (bioblend.cloudman.launch.CloudManLauncher method), 12
run() (bioblend.galaxy.objects.wrappers.Tool method), 79
run() (bioblend.galaxy.objects.wrappers.Workflow method), 70
run_invocation_step_action() (bioblend.galaxy.workflows.WorkflowClient method), 61
run_tool() (bioblend.galaxy.tools.ToolClient method), 48
run_workflow() (bioblend.galaxy.workflows.WorkflowClient method), 61
save_config() (bioblend.cloudman.CloudManConfig method), 14
search_jobs() (bioblend.galaxy.jobs.JobsClient method), 36
search_repositories() (bioblend.toolshed.repositories.ToolShedRepositoryClient method), 93
search_tools() (bioblend.toolshed.tools.ToolShedToolClient method), 96
set_connection_parameters() (bioblend.cloudman.CloudManConfig method), 14
set_extra_parameters() (bioblend.cloudman.CloudManConfig method), 21
set_file_logger() (in module bioblend), 97
set_library_permissions() (bioblend.galaxy.libaries.LibraryClient method), 39
set_master_as_execution_host() (bioblend.cloudman.CloudManInstance method), 16
set_permissions() (bioblend.galaxy.folders.FoldersClient method), 24
set_post_launch_parameters() (bioblend.cloudman.CloudManConfig method), 14
set_pre_launch_parameters() (bioblend.cloudman.CloudManConfig method), 14
set_stream_logger() (in module bioblend), 97
show_category() (bioblend.toolshed.categories.ToolShedCategoryClient method), 89
show_data_table() (bioblend.galaxy.tool_data.ToolDataClient method), 50
show_dataset() (bioblend.galaxy.datasets.DatasetClient method), 21
show_dataset() (bioblend.galaxy.histories.HistoryClient method), 32
show_dataset() (bioblend.galaxy.libraries.LibraryClient method), 40
show_dataset_collection() (bioblend.galaxy.histories.HistoryClient method), 32
show_dataset_provenance() (bioblend.galaxy.histories.HistoryClient method), 32
show_folder() (bioblend.galaxy.folders.FoldersClient method), 24
show_folder() (bioblend.galaxy.libraries.LibraryClient method), 40
show_form() (bioblend.galaxy.forms.FormsClient method), 25
show_genome() (bioblend.galaxy.genomes.GenomeClient method), 26
show_group() (bioblend.galaxy.groups.GroupsClient method), 28
show_history() (bioblend.galaxy.histories.HistoryClient method), 33
show_invocation() (bioblend.galaxy.workflows.WorkflowClient method), 63
show_invocation_step() (bioblend.galaxy.workflows.WorkflowClient method), 63
show_job() (bioblend.galaxy.jobs.JobsClient method), 37
show_library() (bioblend.galaxy.libraries.LibraryClient method), 40
show_matching_datasets() (bioblend.galaxy.histories.HistoryClient method), 34
show_quota() (bioblend.galaxy.quotas.QuotaClient method), 44
show_repository() (bioblend.galaxy.toolshed.ToolshedClient method), 52
show_repository() (bioblend.galley.toolshed.repositories.ToolshedRepositoryClient method), 94
show_repository_revision() (bioblend.galaxy.toolshed.repositories.ToolshedRepositoryClient method), 95
show_role() (bioblend.galaxy.roles.RolesClient method), 46
show_stderr() (bioblend.galaxy.datasets.DatasetClient method), 22
show_stdout() (bioblend.galaxy.datasets.DatasetClient method), 22
show_visualization() (bioblend.galaxy.visual.VisualClient method), 55
T
terminate() (bioblend.cloudman.CloudManInstance method), 16
to_json() (bioblend.galaxy.objects.wrappers.Wrapper method), 69
Tool (class in bioblend.galaxy.objects.wrappers), 79
tool_ids (bioblend.galaxy.objects.wrappers.Workflow attribute), 71
ToolClient (class in bioblend.galaxy.tools), 46
ToolDataClient (class in bioblend.galaxy.tool_data), 50
ToolShedCategoryClient (class in bioblend.toolshed.categories), 89
ToolShedClient (class in bioblend.galaxy.toolshed), 51
ToolShedRepositoryClient (class in bioblend.toolshed.repositories), 89
ToolShedToolClient (class in bioblend.toolshed.tools), 96
touch() (bioblend.galaxy.objects.wrappers.Wrapper method), 69

U
undelete_history() (bioblend.galaxy.histories.HistoryClient method), 34
undelete_quota() (bioblend.galaxy.quotas.QuotaClient method), 45
unmap() (bioblend.galaxy.objects.wrappers.Wrapper method), 69
update() (bioblend.cloudman.CloudManInstance method), 16
update() (bioblend.galaxy.objects.wrappers.History method), 74
update() (bioblend.galaxy.objects.wrappers.HistoryDatasetAssociation method), 78
update() (bioblend.galaxy.objects.wrappers.LibraryDataset method), 79
update_dataset() (bioblend.galaxy.histories.HistoryDataset method), 34
update_dataset_collection() (bioblend.galaxy.histories.HistoryClient method), 35
update_folder() (bioblend.galaxy.folders.FoldersClient method), 24
update_group() (bioblend.galaxy.groups.GroupsClient method), 29
update_history() (bioblend.galaxy.histories.HistoryClient method), 35
update_library_dataset() (bioblend.galaxy.libraries.LibraryClient method), 40
update_quota() (bioblend.galaxy.quotas.QuotaClient method), 45
update_repository() (bioblend.toolshed.repositories.ToolShedRepositoryClient method), 95
upload_data() (bioblend.galaxy.objects.wrappers.Library method), 75
upload_dataset() (bioblend.galaxy.objects.wrappers.History method), 74
upload_dataset_from_library() (bioblend.galaxy.histories.HistoryClient method), 35
upload_file() (bioblend.galaxy.objects.wrappers.History method), 74
upload_file() (bioblend.galaxy.tools.ToolClient method), 49
upload_file_contents() (bioblend.galaxy.libraries.LibraryClient method), 40
upload_file_from_local_path() (bioblend.galaxy.libraries.LibraryClient method), 41
upload_file_from_server() (bioblend.galaxy.libraries.LibraryClient method), 41
upload_file_from_url() (bioblend.galaxy.libraries.LibraryClient method), 42
upload_from_ftp() (bioblend.galaxy.objects.wrappers.History method), 74
upload_from_ftp() (bioblend.galaxy.tools.ToolClient method), 49
upload_from_galaxy_filesystem() (bioblend.galaxy.libraries.LibraryClient method), 42
upload_from_galaxy_fs() (bioblend.galaxy.objects.wrappers.Library method), 75
upload_from_local() (bioblend.galaxy.objects.wrappers.Library method), 76
upload_from_url() (bioblend.galaxy.objects.wrappers.Library method), 76
UserClient (class in bioblend.galaxy.users), 53

V
validate() (bioblend.cloudman.CloudManConfig method), 14
VisualClient (class in bioblend.galaxy.visual), 55
VMLaunchException, 16
W

wait()  (bioblend.galaxy.objects.wrappers.Dataset method), 77
wait_for_dataset()
   (bioblend.galaxy.libraries.LibraryClient method), 43
wait_until_instance_ready()
   (bioblend.cloudman.GenericVMInstance method), 16
Workflow (class in bioblend.galaxy.objects.wrappers), 70
WorkflowClient (class in bioblend.galaxy.workflows), 56
WorkflowPreview (class in bioblend.galaxy.objects.wrappers), 80
Wrapper (class in bioblend.galaxy.objects.wrappers), 69