BioBlend Documentation

Release 0.16.0

Enis Afgan

Jun 13, 2021

CONTENTS

CHAPTER

ONE

ABOUT

BioBlend is a Python library for interacting with Galaxy and CloudMan APIs.

BioBlend is supported and tested on:

- Python 3.6, 3.7, 3.8 and 3.9
- Galaxy release_17.09 and later.

BioBlend's goal is to make it easier to script and automate the running of Galaxy analyses, administering of a Galaxy server, and cloud infrastructure provisioning and scaling via CloudMan. In practice, it makes it possible to do things like this:

- Interact with Galaxy via a straightforward API:

```
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)
```

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

```
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:

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

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.

1.1 About the library name

The library was originally called just *Blend* but we renamed it to reflect more of its domain and a make it bit more unique so it can be easier to find. The name was intended to be short and easily pronounceable. In its original implementation, the goal was to provide a lot more support for CloudMan and other integration capabilities, allowing them to be *blended* together via code. *BioBlend* fit the bill.

CHAPTER

TWO

INSTALLATION

Stable releases of BioBlend are best installed via `pip` from PyPI:

```
$ python3 -m pip install bioblend
```

Alternatively, the most current source code from our Git repository can be installed with:

```
$ python3 -m pip install git+https://github.com/galaxyproject/bioblend
```

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 `python3 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:

```
$ python3 setup.py test
```

CHAPTER

THREE

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 :).

CHAPTER

FOUR

DEVELOPMENT

Anyone interested in contributing or tweaking the library is more then welcome to do so. To start, simply fork the Git repository on Github and start playing with it. Then, issue pull requests.

CHAPTER

FIVE

API DOCUMENTATION

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 Galaxy API

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

5.1.1 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:

```
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). If a Galaxy instance has HTTP Basic authentication with username and password, then the credentials should be included in the URL, e.g. http://user:pass@host:port/galaxy/

- **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

__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:

```
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). If a Galaxy instance has HTTP Basic authentication with username and password, then the credentials should be included in the URL, e.g. http://user:pass@host:port/galaxy/
- **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

property get_retry_delay

property max_get_attempts

Config

Contains possible interaction dealing with Galaxy configuration.

class bioblend.galaxy.config.**ConfigClient**(*galaxy_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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:

```
{'allow_library_path_paste': False,
 'allow_user_creation': True,
 'allow_user_dataset_purge': True,
 'allow_user_deletion': False,
 'enable_unique_workflow_defaults': False,
 'ftp_upload_dir': '/SOMEWHERE/galaxy/ftp_dir',
 'ftp_upload_site': 'galaxy.com',
 'library_import_dir': 'None',
 'logo_url': None,
 'support_url': 'https://galaxyproject.org/support',
 'terms_url': None,
 'user_library_import_dir': None,
 'wiki_url': 'https://galaxyproject.org/'}
```

get_version()

Get the current version of the Galaxy instance.

Return type dict

Returns

Version of the Galaxy instance For example:

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

module = 'configuration'

Datasets

Contains possible interactions with the Galaxy Datasets

class bioblend.galaxy.datasets.**DatasetClient**(*galaxy_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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

Download a dataset to file or in memory. If the dataset state is not 'ok', a `DatasetStateException` will be thrown, unless `require_ok_state=False`.

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.
- **require_ok_state** (*bool*) – If `False`, datasets will be downloaded even if not in an 'ok' state, issuing a `DatasetStateWarning` rather than raising a `DatasetStateException`.
- **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.

get_datasets(*limit: int = 500*, *offset: int = 0*, *name: Optional[str] = None*, *extension: Optional[Union[str, List[str]]] = None*, *state: Optional[Union[str, List[str]]] = None*, *visible: Optional[bool] = None*, *deleted: Optional[bool] = None*, *purged: Optional[bool] = None*, *tool_id: Optional[str] = None*, *tag: Optional[str] = None*, *history_id: Optional[str] = None*, *create_time_min: Optional[str] = None*, *create_time_max: Optional[str] = None*, *update_time_min: Optional[str] = None*, *update_time_max: Optional[str] = None*, *order: str = 'create_time-dsc'*) → List[dict]

Get the latest datasets, or select another subset by specifying optional arguments for filtering (e.g. a history ID).

Since the number of datasets may be very large, `limit` and `offset` parameters are required to specify the desired range.

If the user is an admin, this will return datasets for all the users, otherwise only for the current user.

Parameters

- **limit** (*int*) – Maximum number of datasets to return.
- **offset** (*int*) – Return datasets starting from this specified position. For example, if `limit` is set to 100 and `offset` to 200, datasets 200-299 will be returned.

- **name** (*str*) – Dataset name to filter on.
- **extension** (*str or list of str*) – Dataset extension (or list of extensions) to filter on.
- **state** (*str or list of str*) – Dataset state (or list of states) to filter on.
- **visible** (*bool*) – Optionally filter datasets by their `visible` attribute.
- **deleted** (*bool*) – Optionally filter datasets by their `deleted` attribute.
- **purged** (*bool*) – Optionally filter datasets by their `purged` attribute.
- **tool_id** (*str*) – Tool ID to filter on.
- **tag** (*str*) – Dataset tag to filter on.
- **history_id** (*str*) – Encoded history ID to filter on.
- **create_time_min** (*str*) – Show only datasets created after the provided time and date, which should be formatted as `YYYY-MM-DDTHH-MM-SS`.
- **create_time_max** (*str*) – Show only datasets created before the provided time and date, which should be formatted as `YYYY-MM-DDTHH-MM-SS`.
- **update_time_min** (*str*) – Show only datasets last updated after the provided time and date, which should be formatted as `YYYY-MM-DDTHH-MM-SS`.
- **update_time_max** (*str*) – Show only datasets last updated before the provided time and date, which should be formatted as `YYYY-MM-DDTHH-MM-SS`.
- **order** (*str*) – One or more of the following attributes for ordering datasets: `create_time` (default), `extension`, `hid`, `history_id`, `name`, `update_time`. Optionally, `-asc` or `-dsc` (default) can be appended for ascending and descending order respectively. Multiple attributes can be stacked as a comma-separated list of values, e.g. `create_time-asc, hid-dsc`.

Return type list

Param A list of datasets

`module = 'datasets'`

`publish_dataset`(*dataset_id: str*, *published: bool = False*)

Make a dataset publicly available or private. For more fine-grained control (assigning different permissions to specific roles), use the `update_permissions()` method.

Parameters

- **dataset_id** (*str*) – dataset ID
- **published** (*bool*) – Whether to make the dataset published (`True`) or private (`False`).

Return type dict

Returns Current roles for all available permission types.

Note: This method can only be used with Galaxy `release_19.05` or later.

`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

update_permissions(*dataset_id: str*, *access_ids: Optional[list] = None*, *manage_ids: Optional[list] = None*, *modify_ids: Optional[list] = None*)

Set access, manage or modify permissions for a dataset to a list of roles.

Parameters

- **dataset_id** (*str*) – dataset ID
- **access_ids** (*list*) – role IDs which should have access permissions for the dataset.
- **manage_ids** (*list*) – role IDs which should have manage permissions for the dataset.
- **modify_ids** (*list*) – role IDs which should have modify permissions for the dataset.

Return type dict

Returns Current roles for all available permission types.

Note: This method can only be used with Galaxy `release_19.05` or later.

---

wait_for_dataset(*dataset_id*, *maxwait=12000*, *interval=3*, *check=True*)

Wait until a dataset is in a terminal state.

Parameters

- **dataset_id** (*str*) – dataset ID
- **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 raised.
- **interval** (*float*) – Time (in seconds) to wait between 2 consecutive checks.
- **check** (*bool*) – Whether to check if the dataset terminal state is 'ok'.

Return type dict

Returns Details of the given dataset.

exception bioblend.galaxy.datasets.**DatasetStateException**

exception bioblend.galaxy.datasets.**DatasetStateWarning**

exception bioblend.galaxy.datasets.**DatasetTimeoutException**

Dataset collections

class bioblend.galaxy.dataset_collections.**CollectionDescription**(*name*, *type='list'*, *elements=None*)

to_dict()

class bioblend.galaxy.dataset_collections.**CollectionElement**(*name*, *type='list'*, *elements=None*)

to_dict()

class bioblend.galaxy.dataset_collections.**DatasetCollectionClient**(*galaxy_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

download_dataset_collection(*dataset_collection_id: str*, *file_path: str*) → dict

Download a history dataset collection as an archive.

Parameters

- **dataset_collection_id** (*str*) – Encoded dataset collection ID
- **file_path** (*str*) – The path to which the archive will be downloaded

Return type dict

Returns Information about the downloaded archive.

Note: This method downloads a `zip` archive for Galaxy 21.01 and later. For earlier versions of Galaxy this method downloads a `tgz` archive. This method is only supported by Galaxy 18.01 or later.

---

module = 'dataset_collections'

show_dataset_collection(*dataset_collection_id: str*, *instance_type: str = 'history'*) → dict

Get details of a given dataset collection of the current user

Parameters

- **dataset_collection_id** (*str*) – dataset collection ID
- **instance_type** (*str*) – instance type of the collection - 'history' or 'library'

Return type dict

Returns element view of the dataset collection

wait_for_dataset_collection(*dataset_collection_id: str*, *maxwait: float = 12000*, *interval: float = 3*, *proportion_complete: float = 1.0*, *check: bool = True*) → dict

Wait until all or a specified proportion of elements of a dataset collection are in a terminal state.

Parameters

- **dataset_id** – dataset collection ID
- **maxwait** (*float*) – Total time (in seconds) to wait for the dataset states in the dataset collection to become terminal. If not all datasets are in a terminal state within this time, a `DatasetCollectionTimeoutException` will be raised.
- **interval** (*float*) – Time (in seconds) to wait between two consecutive checks.

- **proportion_complete** (*float*) – Proportion of elements in this collection that have to be in a terminal state for this method to return. Must be a number between 0 and 1. For example: if the dataset collection contains 2 elements, and proportion_complete=0.5 is specified, then wait_for_dataset_collection will return as soon as 1 of the 2 datasets is in a terminal state. Default is 1, i.e. all elements must complete.
- **check** (*bool*) – Whether to check if all the terminal states of datasets in the dataset collection are 'ok'. This will raise an Exception if a dataset is in a terminal state other than 'ok'.

Return type dict

Returns Details of the given dataset collection.

class bioblend.galaxy.dataset_collections.**HistoryDatasetCollectionElement**(*name*, *id*)

class bioblend.galaxy.dataset_collections.**HistoryDatasetElement**(*name*, *id*)

class bioblend.galaxy.dataset_collections.**LibraryDatasetElement**(*name*, *id*)

Datatypes

Contains possible interactions with the Galaxy Datatype

class bioblend.galaxy.datatypes.**DatatypesClient**(*galaxy_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

get_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:

```
['snpmatrix',
 'snptest',
 'tabular',
 'taxonomy',
 'twobit',
 'txt',
 'vcf',
 'wig',
 'xgmml',
 'xml']
```

get_sniffers()

Get the list of all installed sniffers.

Return type list

Returns

A list of sniffer names. For example:

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

`module = 'datatypes'`

Folders

Contains possible interactions with the Galaxy library folders

`class bioblend.galaxy.folders.FoldersClient(galaxy_instance)`
A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

`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

`module = 'folders'`

`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*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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:

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

module = 'forms'

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:

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

FTP files

Contains possible interactions with the Galaxy FTP Files

class bioblend.galaxy.ftpfiles.**FTPFilesClient**(*galaxy_instance*)
A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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

module = 'ftp_files'

Genomes

Contains possible interactions with the Galaxy Histories

class bioblend.galaxy.genomes.**GenomeClient**(*galaxy_instance*)
A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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

Returns dict(status: 'ok', job: <job ID>) If error: dict(status: 'error', error: <error message>)

`module = 'genomes'`

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

class bioblend.galaxy.groups.**GroupsClient**(*galaxy_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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=None*, *role_ids=None*)
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:

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

delete_group_role(*group_id*, *role_id*)
Remove a role from the given group.

Parameters

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

Return type dict

Returns The role which was removed

delete_group_user(*group_id*, *user_id*)
Remove a user from the given group.

Parameters

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

Return type dict

Returns The user which was removed

get_group_roles(*group_id*)
Get the list of roles associated to the given group.

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

Return type list of dicts

Returns List of group roles' info

get_group_users(*group_id*)
Get the list of users associated to the given group.

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

Return type list of dicts

Returns List of group users' info

get_groups()
Get all (not deleted) groups.

Return type list

Returns

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

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

module = 'groups'

show_group(*group_id*)

Get details of a given group.

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

Return type dict

Returns

A description of group For example:

```
{'id': '33abac023ff186c2',
 'model_class': 'Group',
 'name': 'Listeria',
 'roles_url': '/api/groups/33abac023ff186c2/roles',
 'url': '/api/groups/33abac023ff186c2',
 'users_url': '/api/groups/33abac023ff186c2/users'}
```

update_group(*group_id*, *group_name=None*, *user_ids=None*, *role_ids=None*)

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*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

copy_content(*history_id*, *content_id*, *source='hda'*)

Copy existing content (e.g. a dataset) to a history.

Parameters

- **history_id** (*str*) – ID of the history to which the content should be copied
- **content_id** (*str*) – ID of the content to copy
- **source** (*str*) – Source of the content to be copied: 'hda' (for a history dataset, the default), 'hdca' (for a dataset collection), 'library' (for a library dataset) or 'library_folder' (for all datasets in a library folder).

Return type dict

Returns Information about the copied content

copy_dataset(*history_id*, *dataset_id*, *source='hda'*)

Copy a dataset to a history.

Parameters

- **history_id** (*str*) – history ID to which the dataset should be copied
- **dataset_id** (*str*) – dataset ID
- **source** (*str*) – Source of the dataset to be copied: 'hda' (the default), 'library' or 'library_folder'

Return type dict

Returns Information about the copied dataset

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:

```
{'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:

```
{'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.yml` configuration file.

---

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.yml` configuration file.

---

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_histories(*history_id=None*, *name=None*, *deleted=False*, *published=None*, *slug=None*)

Get all histories, or select a subset by specifying optional arguments for filtering (e.g. a history name).

Parameters

- **history_id** (*str*) – Encoded history ID to filter on

  Deprecated since version 0.15.0: To get details of a history for which you know the ID, use the much more efficient *show_history()* instead.

- **name** (*str*) – History name to filter on.

- **deleted** (*bool*) – whether to filter for the deleted histories (`True`) or for the non-deleted ones (`False`)
- **published** (*bool or None*) – whether to filter for the published histories (`True`) or for the non-published ones (`False`). If not set, no filtering is applied. Note the filtering is only applied to the user's own histories; to access all histories published by any user, use the `get_published_histories` method.
- **slug** (*str*) – History slug to filter on

Return type list

Returns List of history dicts.

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

Return type dict

Returns History representation

`get_published_histories(name=None, deleted=False, slug=None)`
Get all published histories (by any user), or select a subset by specifying optional arguments for filtering (e.g. a history name).

Parameters

- **name** (*str*) – History name to filter on.
- **deleted** (*bool*) – whether to filter for the deleted histories (`True`) or for the non-deleted ones (`False`)
- **slug** (*str*) – History slug to filter on

Return type list

Returns List of history dicts.

`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.

`import_history(file_path=None, url=None)`
Import a history from an archive on disk or a URL.

Parameters

- **file_path** (*str*) – Path to exported history archive on disk.
- **url** (*str*) – URL for an exported history archive

`module = 'histories'`

`open_history(history_id)`
Open Galaxy in a new tab of the default web browser and switch to the specified history.

Parameters **history_id** (*str*) – ID of the history to switch to

Return type NoneType

Returns None

> **Warning:** After opening the specified history, all previously opened Galaxy tabs in the browser session will have the current history changed to this one, even if the interface still shows another history. Refreshing any such tab is recommended.

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:

```
{'id': '6fbd9b2274c62ebe',
 'job_id': '5471ba76f274f929',
 'parameters': {'chromInfo': '"/usr/local/galaxy/galaxy-dist/tool-
↪data/shared/ucsc/chrom/mm9.len"',
                'dbkey': '"mm9"',
                'experiment_name': '"H3K4me3_TAC_MACS2"',
                'input_chipseq_file1': {'id': '6f0a311a444290f2',
                                        'uuid': 'null'},
                'input_control_file1': {'id': 'c21816a91f5dc24e',
                                        'uuid': '16f8ee5e-228f-41e2-
↪921e-a07866edce06'},
```

(continues on next page)

(continued from previous page)

```
                'major_command': '{"gsize": "2716965481.0", "bdg":
↪"False", "__current_case__": 0, "advanced_options": {"advanced_
↪options_selector": "off", "__current_case__": 1}, "input_chipseq_
↪file1": 104715, "xls_to_interval": "False", "major_command_selector
↪": "callpeak", "input_control_file1": 104721, "pq_options": {"pq_
↪options_selector": "qvalue", "qvalue": "0.05", "__current_case__":␣
↪1}, "bw": "300", "nomodel_type": {"nomodel_type_selector": "create_
↪model", "__current_case__": 1}}'},
 'stderr': '',
 'stdout': '',
 'tool_id': 'toolshed.g2.bx.psu.edu/repos/ziru-zhou/macs2/modencode_
↪peakcalling_macs2/2.0.10.2',
 'uuid': '5c0c43f5-8d93-44bd-939d-305e82f213c6'}
```

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 a list with info for all datasets in the given history. Note that inside each dataset info dict, the id which should be used for further requests about this history dataset is given by the value of the *id* (not *dataset_id*) key.
- **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 or list of dicts

Returns details of the given history or list of dataset info

Note: As an alternative to using the `contents=True` parameter, consider using `gi.datasets.get_datasets(history_id=history_id)` which offers more extensive functionality for filtering and ordering the results.

---

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
- **datatype** (*str*) – Replace the datatype of the history dataset with the given string. The string must be a valid Galaxy datatype, both the current and the target datatypes must allow datatype changes, and the dataset must not be in use as input or output of a running job (including uploads), otherwise an error will be raised.
- **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

Changed in version 0.8.0: Changed the return value from the status code (type int) to a dict.

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

Changed in version 0.8.0: Changed the return value from the status code (type int) to a dict.

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).
- **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

Changed in version 0.8.0: Changed the return value from the status code (type int) to a dict.

`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

Invocations

Contains possible interactions with the Galaxy workflow invocations

`class bioblend.galaxy.invocations.InvocationClient`(*galaxy_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

`cancel_invocation`(*invocation_id*)

Cancel the scheduling of a workflow.

Parameters **invocation_id** (*str*) – Encoded workflow invocation ID

Return type dict

Returns The workflow invocation being cancelled

`get_invocation_biocompute_object`(*invocation_id*)

Get a BioCompute object for an invocation.

Parameters **invocation_id** (*str*) – Encoded workflow invocation ID

Return type dict

Returns The BioCompute object

get_invocation_report(*invocation_id*)
Get a Markdown report for an invocation.

Parameters **invocation_id** (*str*) – Encoded workflow invocation ID

Return type dict

Returns

The invocation report. For example:

```
{'markdown': '\n# Workflow Execution Summary of Example workflow\n\n
 ## Workflow Inputs\n\n\n## Workflow Outputs\n\n\n
 ## Workflow\n```galaxy\n
 workflow_display(workflow_id=f2db41e1fa331b3e)\n```\n',
 'render_format': 'markdown',
 'workflows': {'f2db41e1fa331b3e': {'name': 'Example workflow'}}}
```

get_invocation_report_pdf(*invocation_id*, *file_path*, *chunk_size=4096*)
Get a PDF report for an invocation.

Parameters

- **invocation_id** (*str*) – Encoded workflow invocation ID
- **file_path** (*str*) – Path to save the report

get_invocation_step_jobs_summary(*invocation_id*)
Get a detailed summary of an invocation, listing all jobs with their job IDs and current states.

Parameters **invocation_id** (*str*) – Encoded workflow invocation ID

Return type list of dicts

Returns

The invocation step jobs summary. For example:

```
[{'id': 'e85a3be143d5905b',
  'model': 'Job',
  'populated_state': 'ok',
  'states': {'ok': 1}},
 {'id': 'c9468fdb6dc5c5f1',
  'model': 'Job',
  'populated_state': 'ok',
  'states': {'running': 1}},
 {'id': '2a56795cad3c7db3',
  'model': 'Job',
  'populated_state': 'ok',
  'states': {'new': 1}}]
```

get_invocation_summary(*invocation_id*)
Get a summary of an invocation, stating the number of jobs which succeed, which are paused and which have errored.

Parameters **invocation_id** (*str*) – Encoded workflow invocation ID

Return type dict

Returns

The invocation summary. For example:

```
{'states': {'paused': 4, 'error': 2, 'ok': 2},
 'model': 'WorkflowInvocation',
 'id': 'a799d38679e985db',
 'populated_state': 'ok'}
```

get_invocations(*workflow_id=None*, *history_id=None*, *user_id=None*, *include_terminal=True*, *limit=None*, *view='collection'*, *step_details=False*)

Get all workflow invocations, or select a subset by specifying optional arguments for filtering (e.g. a workflow ID).

Parameters

- **workflow_id** (*str*) – Encoded workflow ID to filter on
- **history_id** (*str*) – Encoded history ID to filter on
- **user_id** (*str*) – Encoded user ID to filter on. This must be your own user ID if your are not an admin user.
- **include_terminal** (*bool*) – Whether to include terminal states.
- **limit** (*int*) – Maximum number of invocations to return - if specified, the most recent invocations will be returned.
- **view** (*str*) – Level of detail to return per invocation, either 'element' or 'collection'.
- **step_details** (*bool*) – If 'view' is 'element', also include details on individual steps.

Return type list

Returns

A list of workflow invocations. For example:

```
[{'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'}]
```

module = 'invocations'

rerun_invocation(*invocation_id: str*, *inputs_update: Optional[dict] = None*, *params_update: Optional[dict] = None*, *history_id: Optional[str] = None*, *history_name: Optional[str] = None*, *import_inputs_to_history: bool = False*, *replacement_params: Optional[dict] = None*, *allow_tool_state_corrections: bool = False*, *inputs_by: Optional[str] = None*, *parameters_normalized: bool = False*)

Rerun a workflow invocation. For more extensive documentation of all parameters, see the `gi.workflows.invoke_workflow()` method.

Parameters

- **invocation_id** (*str*) – Encoded workflow invocation ID to be rerun
- **inputs_update** (*dict*) – If different datasets should be used to the original invocation, this should contain a mapping of workflow inputs to the new datasets and dataset collections.
- **params_update** (*dict*) – If different non-dataset tool parameters should be used to the original invocation, this should contain a mapping of the new parameter values.

- **history_id** (*str*) – The encoded history ID where to store the workflow outputs. 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 outputs. 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
- **inputs_by** (*str*) – Determines how inputs are referenced. Can be "step_index|step_uuid" (default), "step_index", "step_id", "step_uuid", or "name".
- **parameters_normalized** (*bool*) – Whether Galaxy should normalize the input parameters to ensure everything is referenced by a numeric step ID. Default is `False`, but when setting parameters for a subworkflow, `True` is required.

Return type dict

Returns A dict describing the new workflow invocation.

Note: This method can only be used with Galaxy `release_21.01` or later.

`run_invocation_step_action`(*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

- **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

`show_invocation`(*invocation_id*)

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

Parameters **invocation_id** (*str*) – Encoded workflow invocation ID

Return type dict

Returns

The workflow invocation. For example:

```
{'history_id': '2f94e8ae9edff68a',
 'id': 'df7a1f0c02a5b08e',
 'inputs': {'0': {'id': 'a7db2fac67043c7e',
   'src': 'hda',
   'uuid': '7932ffe0-2340-4952-8857-dbaa50f1f46a'}},
 'model_class': 'WorkflowInvocation',
 'state': 'ready',
 'steps': [{'action': None,
   'id': 'd413a19dec13d11e',
   'job_id': None,
   'model_class': 'WorkflowInvocationStep',
   'order_index': 0,
   'state': None,
   'update_time': '2015-10-31T22:00:26',
   'workflow_step_id': 'cbbbf59e8f08c98c',
   'workflow_step_label': None,
   'workflow_step_uuid': 'b81250fd-3278-4e6a-b269-56a1f01ef485'},
  {'action': None,
   'id': '2f94e8ae9edff68a',
   'job_id': 'e89067bb68bee7a0',
   'model_class': 'WorkflowInvocationStep',
   'order_index': 1,
   'state': 'new',
   'update_time': '2015-10-31T22:00:26',
   'workflow_step_id': '964b37715ec9bd22',
   'workflow_step_label': None,
   'workflow_step_uuid': 'e62440b8-e911-408b-b124-e05435d3125e'}],
 'update_time': '2015-10-31T22:00:26',
 'uuid': 'c8aa2b1c-801a-11e5-a9e5-8ca98228593c',
 'workflow_id': '03501d7626bd192f'}
```


show_invocation_step(*invocation_id*, *step_id*)

See the details of a particular workflow invocation step.

Parameters

- **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:


```
{'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'}
```

wait_for_invocation(*invocation_id*, *maxwait=12000*, *interval=3*, *check=True*)
Wait until an invocation is in a terminal state.

Parameters

- **invocation_id** (*str*) – Invocation ID to wait for.
- **maxwait** (*float*) – Total time (in seconds) to wait for the invocation state to become terminal. If the invocation state is not terminal within this time, a `TimeoutException` will be raised.
- **interval** (*float*) – Time (in seconds) to wait between 2 consecutive checks.
- **check** (*bool*) – Whether to check if the invocation terminal state is 'scheduled'.

Return type dict

Returns Details of the workflow invocation.

Jobs

Contains possible interactions with the Galaxy Jobs

class bioblend.galaxy.jobs.**JobsClient**(*galaxy_instance*)
A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

cancel_job(*job_id: str*)
Cancel a job, deleting output datasets.

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

Return type bool

Returns `True` if the job was successfully cancelled, `False` if it was already in a terminal state before the cancellation.

get_common_problems(*job_id: str*) → dict
Query inputs and jobs for common potential problems that might have resulted in job failure.

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

Return type dict

Returns dict containing potential problems

Note: This method is only supported by Galaxy 19.05 or later.

---

get_destination_params(*job_id: str*) → dict
Get destination parameters for a job, describing the environment and location where the job is run.

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

Return type dict

Returns Destination parameters for the given job

Note: This method is only supported by Galaxy 20.05 or later and requires the user to be an admin.

get_inputs(*job_id: str*) → List[dict]
Get dataset inputs used by a job.

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

Return type list of dicts

Returns Inputs for the given job

get_jobs(*state=None*, *history_id=None*, *invocation_id=None*, *tool_id=None*, *workflow_id=None*, *user_id=None*, *date_range_min=None*, *date_range_max=None*, *limit=500*, *offset=0*, *user_details=False*)
Get all jobs, or select a subset by specifying optional arguments for filtering (e.g. a state).

If the user is an admin, this will return jobs for all the users, otherwise only for the current user.

Parameters

- **state** (*str or list of str*) – Job states to filter on.
- **history_id** (*str*) – Encoded history ID to filter on.
- **invocation_id** (*string*) – Encoded workflow invocation ID to filter on.
- **tool_id** (*str or list of str*) – Tool IDs to filter on.
- **workflow_id** (*string*) – Encoded workflow ID to filter on.
- **user_id** (*str*) – Encoded user ID to filter on. Only admin users can access the jobs of other users.
- **date_range_min** (*str*) – Mininum job update date (in YYYY-MM-DD format) to filter on.
- **date_range_max** (*str*) – Maximum job update date (in YYYY-MM-DD format) to filter on.
- **limit** (*int*) – Maximum number of jobs to return.
- **offset** (*int*) – Return jobs starting from this specified position. For example, if `limit` is set to 100 and `offset` to 200, jobs 200-299 will be returned.
- **user_details** (*bool*) – If `True` and the user is an admin, add the user email to each returned job dictionary.

Return type list of dict

Returns

Summary information for each selected job. For example:

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

(continues on next page)

(continued from previous page)

```
  'id': '1cd8e2f6b131e891',
  'model_class': 'Job',
  'state': 'ok',
  'tool_id': 'upload1',
  'update_time': '2014-03-01T16:05:39.558458'}]
```

Note:

The following filtering options can only be used with Galaxy `release_21.05` or later: user_id, limit, offset, workflow_id, invocation_id

`get_metrics`(*job_id: str*) → List[dict]
Return job metrics for a given job.

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

Return type list

Returns list containing job metrics

Note: Calling `show_job()` with `full_details=True` also returns the metrics for a job if the user is an admin. This method allows to fetch metrics even as a normal user as long as the Galaxy instance has the `expose_potentially_sensitive_job_metrics` option set to `true` in the `config/galaxy.yml` configuration file.

`get_outputs`(*job_id: str*) → List[dict]
Get dataset outputs produced by a job.

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

Return type list of dicts

Returns Outputs of the given job

`get_state`(*job_id: str*) → str
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.

`module = 'jobs'`

`report_error`(*job_id: str*, *dataset_id: str*, *message: str*, *email: Optional[str] = None*) → dict
Report an error for a given job and dataset to the server administrators.

Parameters

- **`job_id`** (*str*) – job ID
- **`dataset_id`** (*str*) – Dataset ID
- **`message`** (*str*) – Error message

- **email** (*str*) – Email for error report submission. If not specified, the email associated with the Galaxy user account is used by default.

Return type dict

Returns dict containing job error reply

Note: This method is only supported by Galaxy 20.01 or later.

---

rerun_job(*job_id*, *remap=False*, *tool_inputs_update=None*, *history_id=None*)
Rerun a job.

Parameters

- **job_id** (*str*) – job ID
- **remap** (*bool*) – when `True`, the job output(s) will be remapped onto the dataset(s) created by the original job; if other jobs were waiting for this job to finish successfully, they will be resumed using the new outputs of this tool run. When `False`, new job output(s) will be created. Note that if Galaxy does not permit remapping for the job in question, specifying `True` will result in an error.
- **tool_inputs_update** (*dict*) – dictionary specifying any changes which should be made to tool parameters for the rerun job.
- **history_id** (*str*) – ID of the history in which the job should be executed; if not specified, the same history will be used as the original job run.

Return type dict

Returns Information about outputs and the rerun job

Note: This method can only be used with Galaxy `release_21.01` or later.

---

resume_job(*job_id: str*) → dict
Resume a job if it is paused.

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

Return type dict

Returns dict containing output dataset associations

Note: This method is only supported by Galaxy 18.09 or later.

---

search_jobs(*tool_id: str*, *inputs: dict*, *state: Optional[str] = None*) → List[dict]
Return jobs matching input parameters.

Parameters

- **tool_id** (*str*) – only return jobs associated with this tool ID
- **inputs** (*dict*) – return only jobs that have matching inputs
- **state** (*str*) – only return jobs in this state

Return type list of dicts

Returns Summary information for each matching job

This method is designed to scan the list of previously run jobs and find records of jobs with identical input parameters and datasets. This can be used to minimize the amount of repeated work by simply recycling the old results.

Changed in version 0.16.0: Replaced the `job_info` parameter with separate `tool_id`, `inputs` and `state`.

Note: This method is only supported by Galaxy 18.01 or later.

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:

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

show_job_lock() → bool
Show whether the job lock is active or not. If it is active, no jobs will dispatch on the Galaxy server.

Return type bool

Returns Status of the job lock

Note: This method is only supported by Galaxy 20.05 or later and requires the user to be an admin.

update_job_lock(*active=False*) → bool
Update the job lock status by setting `active` to either `True` or `False`. If `True`, all job dispatching will be blocked.

Return type bool

Returns Updated status of the job lock

Note: This method is only supported by Galaxy 20.05 or later and requires the user to be an admin.

wait_for_job(*job_id*, *maxwait=12000*, *interval=3*, *check=True*)
Wait until a job is in a terminal state.

Parameters

- **job_id** (*str*) – job ID
- **maxwait** (*float*) – Total time (in seconds) to wait for the job state to become terminal. If the job state is not terminal within this time, a `TimeoutException` will be raised.
- **interval** (*float*) – Time (in seconds) to wait between 2 consecutive checks.
- **check** (*bool*) – Whether to check if the job terminal state is 'ok'.

Return type dict

Returns Details of the given job.

Libraries

Contains possible interactions with the Galaxy Data Libraries

class bioblend.galaxy.libraries.**LibraryClient**(*galaxy_instance*)
A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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 provided, 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:

```
{'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:

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

get_dataset_permissions(*dataset_id*)
Get the permissions for a dataset.

Parameters **dataset_id** (*str*) – id of the dataset

Return type dict

Returns dictionary with all applicable permissions' values

get_folders(*library_id*, *folder_id=None*, *name=None*)
Get all the folders in a library, or select a subset by specifying a folder name for filtering.

Parameters

- **library_id** (*str*) – library id to use
- **folder_id** (*str*) – filter for folder by folder id

  Deprecated since version 0.16.0: To get details of a folder for which you know the ID, use the much more efficient *show_folder()* instead.
- **name** (*str*) – Folder name to filter on. 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 libraries, or select a subset by specifying optional arguments for filtering (e.g. a library name).

Parameters

- **library_id** (*str*) – filter for library by library id

  Deprecated since version 0.16.0: To get details of a library for which you know the ID, use the much more efficient *show_library()* instead.
- **name** (*str*) – Library name to filter on.
- **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

module = 'libraries'

set_dataset_permissions(*dataset_id*, *access_in=None*, *modify_in=None*, *manage_in=None*)

Set the permissions for a dataset. Note: it will override all security for this dataset even if you leave out a permission type.

Parameters

- **dataset_id** (*str*) – id of the dataset
- **access_in** (*list*) – list of role ids
- **modify_in** (*list*) – list of role ids
- **manage_in** (*list*) – list of role ids

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)
- **tags** (*list*) – Replace library dataset tags with the given list

Return type dict

Returns details of the updated dataset

upload_file_contents(*library_id*, *pasted_content*, *folder_id=None*, *file_type='auto'*, *dbkey='?'*, *tags=None*)

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
- **tags** (*list*) – A list of tags to add to the datasets

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='?'*, *tags=None*)

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
- **tags** (*list*) – A list of tags to add to the datasets

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=False*, *tags=None*)

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.yml` 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.

  Changed in version 0.14.0: Changed the default from `True` to `False`.
- **tags** (*list*) – A list of tags to add to the datasets

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='?'*, *tags=None*)

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
- **tags** (*list*) – A list of tags to add to the datasets

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=False*, *tags=None*)

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` option set to `true` in the `config/galaxy.yml` configuration file.

Parameters

- **library_id** (*str*) – id of the library where to place the uploaded file
- **filesystem_paths** (*str*) – file paths on the Galaxy server to upload to the library, one file per line
- **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.

  Changed in version 0.14.0: Changed the default from `True` to `False`.
- **tags** (*list*) – A list of tags to add to the datasets

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*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

create_quota(*name*, *description*, *amount*, *operation*, *default='no'*, *in_users=None*, *in_groups=None*)

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:

```
{'url': '/galaxy/api/quotas/386f14984287a0f7',
 'model_class': 'Quota',
 'message': "Quota 'Testing' has been created with 1 associated users␣
↪and 0 associated groups.",
 'id': '386f14984287a0f7',
 '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:

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

module = 'quotas'

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:

```
{'bytes': 107374182400,
 'default': [],
 'description': 'just testing',
 'display_amount': '100.0 GB',
 'groups': [],
 'id': '0604c8a56abe9a50',
 'model_class': 'Quota',
 'name': 'test ',
 'operation': '=',
 '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=None*, *in_groups=None*)
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`, `unregistered`, `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

class bioblend.galaxy.roles.**RolesClient**(*galaxy_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

create_role(*role_name*, *description*, *user_ids=None*, *group_ids=None*)

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

Return type dict

Returns

Details of the newly created role. For example:

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

Changed in version 0.15.0: Changed the return value from a 1-element list to a dict.

get_roles()

Displays a collection (list) of roles.

Return type list

Returns

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

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


module = 'roles'

show_role(*role_id*)

Display information on a single role

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

Return type dict

Returns

Details of the given role. For example:


```
{"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*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

get_citations(*tool_id: str*) → List[dict]

Get BibTeX citations for a given tool ID.

Parameters **tool_id** (*str*) – id of the requested tool

Return type list of dicts

Param list containing the citations

get_tool_panel()

Get a list of available tool elements in Galaxy's configured toolbox.

Return type list

Returns List containing tools (if not in sections) or tool sections with nested tool descriptions.

See also:

bioblend.galaxy.toolshed.get_repositories()

`get_tools`(*tool_id=None*, *name=None*, *trackster=None*)
Get all tools, or select a subset by specifying optional arguments for filtering (e.g. a tool name).

Parameters

- **tool_id** (*str*) – id of the requested tool

  Deprecated since version 0.16.0: To get details of a tool for which you know the ID, use the much more efficient *show_tool()* instead.

- **name** (*str*) – Tool name to filter on.
- **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 only to Galaxy admins.

Parameters **tool_id** (*str*) – id of the requested tool

Return type dict

Returns Tool requirement status

`module = 'tools'`

`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.

`requirements`(*tool_id*)
Return the resolver status for a specific tool. This functionality is available only to Galaxy admins.

Parameters **tool_id** (*str*) – id of the requested tool

Return type list

Returns

List containing a resolver status dict for each tool requirement. For example:

```
[{'cacheable': False,
  'dependency_resolver': {'auto_init': True,
                          'auto_install': False,
                          'can_uninstall_dependencies': True,
                          'ensure_channels': 'iuc,conda-forge,
↪bioconda,defaults',
                          'model_class': 'CondaDependencyResolver',
                          'prefix': '/mnt/galaxy/tool_dependencies/_
↪conda',
                          'resolver_type': 'conda',
                          'resolves_simple_dependencies': True,
                          'use_local': False,
                          'versionless': False},
  'dependency_type': 'conda',
  'environment_path': '/mnt/galaxy/tool_dependencies/_conda/envs/__
↪blast@2.10.1',
  'exact': True,
  'model_class': 'MergedCondaDependency',
  'name': 'blast',
  'version': '2.10.1'}]
```

run_tool(*history_id*, *tool_id*, *tool_inputs*, *input_format='legacy'*)

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)
- **input_format** (*string*) – input format for the payload. Possible values are the default 'legacy' (where inputs nested inside conditionals or repeats are identified with e.g. '<conditional_name>|<input_name>') or '21.01' (where inputs inside conditionals or repeats are nested elements).

Return type dict

Returns

Information about outputs and job For example:

```
{'implicit_collections': [],
 'jobs': [{'create_time': '2019-05-08T12:26:16.067372',
           'exit_code': None,
           'id': '7dd125b61b35d782',
           'model_class': 'Job',
           'state': 'new',
           'tool_id': 'cut1',
           'update_time': '2019-05-08T12:26:16.067389'}],
```

(continues on next page)

(continued from previous page)


```
'output_collections': [],
'outputs': [{'create_time': '2019-05-08T12:26:15.997739',
             'data_type': 'galaxy.datatypes.tabular.Tabular',
             'deleted': False,
             'file_ext': 'tabular',
             'file_size': 0,
             'genome_build': '?',
             'hda_ldda': 'hda',
             'hid': 42,
             'history_content_type': 'dataset',
             'history_id': 'df8fe5ddadbf3ab1',
             'id': 'aeb65580396167f3',
             'metadata_column_names': None,
             'metadata_column_types': None,
             'metadata_columns': None,
             'metadata_comment_lines': None,
             'metadata_data_lines': None,
             'metadata_dbkey': '?',
             'metadata_delimiter': '       ',
             'misc_blurb': 'queued',
             'misc_info': None,
             'model_class': 'HistoryDatasetAssociation',
             'name': 'Cut on data 1',
             'output_name': 'out_file1',
             'peek': None,
             'purged': False,
             'state': 'new',
             'tags': [],
             'update_time': '2019-05-08T12:26:16.069798',
             'uuid': 'd91d10af-7546-45be-baa9-902010661466',
             'visible': True}]}
```


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

uninstall_dependencies(*tool_id: str*) → dict
Uninstall dependencies for a given tool via a resolver. This works only for Conda currently. This functionality is available only to Galaxy admins.

Parameters **tool_id** (*str*) – id of the requested tool

Return type dict

Returns Tool requirement status

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

class bioblend.galaxy.tool_data.**ToolDataClient**(*galaxy_instance*)
A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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:

```
[{"model_class": "TabularToolDataTable", "name": "fasta_indexes"},
 {"model_class": "TabularToolDataTable", "name": "bwa_indexes"}]
```

`module = 'tool_data'`

`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:

```
{'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:

```
{'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'}
```

Tool dependencies

Contains interactions dealing with Galaxy dependency resolvers.

class bioblend.galaxy.tool_dependencies.**ToolDependenciesClient**(*galaxy_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

module = 'dependency_resolvers'

summarize_toolbox(*index=None*, *tool_ids=None*, *resolver_type=None*, *include_containers=False*, *container_type=None*, *index_by='requirements'*)

Summarize requirements across toolbox (for Tool Management grid).

Parameters

- **index** (*int*) – index of the dependency resolver with respect to the dependency resolvers config file
- **tool_ids** (*list*) – tool_ids to return when index_by=tools
- **resolver_type** (*str*) – restrict to specified resolver type
- **include_containers** (*bool*) – include container resolvers in resolution
- **container_type** (*str*) – restrict to specified container type
- **index_by** (*str*) – By default results are grouped by requirements. Set to 'tools' to return one entry per tool.

Return type list of dicts

Returns

dictified descriptions of the dependencies, with attribute *dependency_type: None* if no match was found. For example:

```
[{'requirements': [{'name': 'galaxy_sequence_utils',
                    'specs': [],
                    'type': 'package',
                    'version': '1.1.4'},
                   {'name': 'bx-python',
                    'specs': [],
                    'type': 'package',
                    'version': '0.8.6'}],
  'status': [{'cacheable': False,
              'dependency_type': None,
              'exact': True,
              'model_class': 'NullDependency',
              'name': 'galaxy_sequence_utils',
              'version': '1.1.4'},
             {'cacheable': False,
              'dependency_type': None,
              'exact': True,
              'model_class': 'NullDependency',
              'name': 'bx-python',
              'version': '0.8.6'}],
  'tool_ids': ['vcf_to_maf_customtrack1']}]
```

Note:

This method can only be used with Galaxy `release_20.01` or later and requires the user to be an admin. It relies on an experimental API particularly tied to the GUI and therefore is subject to breaking changes.

ToolShed

Interaction with a Galaxy Tool Shed.

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

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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:

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

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

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).
- **tool_panel_section_id** (*str*) – The ID of the Galaxy tool panel section where the tool should be insterted 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.

module = 'tool_shed_repositories'

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:

```
{'changeset_revision': 'b17455fb6222',
 'ctx_rev': '8',
 'owner': 'aaron',
 'status': 'Installed',
 'url': '/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

uninstall_repository_revision(*name*, *owner*, *changeset_revision*, *tool_shed_url*, *remove_from_disk=True*)

Uninstalls a specified repository revision from this Galaxy instance.

Parameters

- **name** (*str*) – The name of the repository

- **owner** (*str*) – The owner of the repository
- **changeset_revision** (*str*) – The revision of the repository to uninstall
- **tool_shed_url** (*str*) – URL of the Tool Shed from which the repository was installed from (e.g., `https://testtoolshed.g2.bx.psu.edu`)
- **remove_from_disk** (*bool*) – whether to also remove the repository from disk (the default) or only deactivate it

Return type dict

Returns If successful, a dictionary with a message noting the removal

---

Users

Contains possible interaction dealing with Galaxy users.

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

class bioblend.galaxy.users.**UserClient**(*galaxy_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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.yml` 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.yml` 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.yml` 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.

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.yml` configuration file.
- **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.yml` configuration file.
- **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.yml` configuration file.

Return type list

Returns

a list of dicts with user details. For example:

```
[{'email': 'a_user@example.com',
  'id': 'dda47097d9189f15',
  'url': '/api/users/dda47097d9189f15'}]
```

`module = 'users'`

`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

`update_user`(*user_id*, ***kwds*)
Update user information. Some of the attributes that can be modified are documented below.

Parameters

- **user_id** (*str*) – encoded user ID
- **username** (*str*) – Replace user name with the given string
- **email** (*str*) – Replace user email with the given string

Return type dict

Returns details of the updated user

Visual

Contains possible interactions with the Galaxy visualization

`class bioblend.galaxy.visual.VisualClient`(*galaxy_instance*)
A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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

Return type list

Returns

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

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

module = 'visualizations'

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:

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

Workflows

Contains possible interactions with the Galaxy Workflows

class bioblend.galaxy.workflows.**WorkflowClient**(*galaxy_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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*, *version=None*)
Exports a workflow.

Parameters

- **workflow_id** (*str*) – Encoded workflow ID
- **version** (*int*) – Workflow version to export

Return type dict

Returns Dictionary representing the requested workflow

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

extract_workflow_from_history(*history_id*, *workflow_name*, *job_ids=None*, *dataset_hids=None*, *dataset_collection_hids=None*)
Extract a workflow from a history.

Parameters

- **history_id** (*str*) – Encoded history ID
- **workflow_name** (*str*) – Name of the workflow to create
- **job_ids** (*list*) – Optional list of job IDs to filter the jobs to extract from the history
- **dataset_hids** (*list*) – Optional list of dataset hids corresponding to workflow inputs when extracting a workflow from history
- **dataset_collection_hids** (*list*) – Optional list of dataset collection hids corresponding to workflow inputs when extracting a workflow from history

Return type dict

Returns A description of the created workflow

get_invocations(*workflow_id*)

Get a list containing all the workflow invocations corresponding to the specified workflow.

For more advanced filtering use InvocationClient.get_invocations().

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

Return type list

Returns

A list of workflow invocations. For example:

```
[{'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 select a subset by specifying optional arguments for filtering (e.g. a workflow name).

Parameters

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

  Deprecated since version 0.16.0: To get details of a workflow for which you know the ID, use the much more efficient *show_workflow()* instead.

- **name** (*str*) – Workflow name to filter on.
- **published** (*bool*) – if `True`, return also published workflows

Return type list

Returns

A list of workflow dicts. For example:

```
[{'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:

```
{'id': 'ee0e2b4b696d9092',
 'model_class': 'StoredWorkflow',
 'name': 'Super workflow that solves everything!',
 'published': False,
 'tags': [],
 'url': '/api/workflows/ee0e2b4b696d9092'}
```

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:

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

```
{'name': 'Training: 16S rRNA sequencing with mothur: main tutorial',
 'tags': [],
 'deleted': false,
 'latest_workflow_uuid': '368c6165-ccbe-4945-8a3c-d27982206d66',
 'url': '/api/workflows/94bac0a90086bdcf',
```

(continues on next page)

(continued from previous page)

```
'number_of_steps': 44,
'published': false,
'owner': 'jane-doe',
'model_class': 'StoredWorkflow',
'id': '94bac0a90086bdcf'}
```

invoke_workflow(*workflow_id: str*, *inputs: Optional[dict] = None*, *params: Optional[dict] = None*, *history_id: Optional[str] = None*, *history_name: Optional[str] = None*, *import_inputs_to_history: bool = False*, *replacement_params: Optional[dict] = None*, *allow_tool_state_corrections: bool = False*, *inputs_by: Optional[str] = None*, *parameters_normalized: bool = False*) → dict

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': '29beef4fadeed09f', '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. Alternatively workflow steps may be addressed by the label that can be set in the workflow editor. If using uuid or label you need to also set the `inputs_by` parameter to `step_uuid` or `name`.
- **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)
- **inputs_by** (*str*) – Determines how inputs are referenced. Can be "step_index|step_uuid" (default), "step_index", "step_id", "step_uuid", or "name".
- **parameters_normalized** (*bool*) – Whether Galaxy should normalize `params` to ensure everything is referenced by a numeric step ID. Default is `False`, but when setting `params` for a subworkflow, `True` is required.

Return type dict

Returns

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

```
{'history_id': '2f94e8ae9edff68a',
 'id': 'df7a1f0c02a5b08e',
 'inputs': {'0': {'id': 'a7db2fac67043c7e',
                  'src': 'hda',
                  'uuid': '7932ffe0-2340-4952-8857-dbaa50f1f46a'}},
 'model_class': 'WorkflowInvocation',
 'state': 'ready',
 'steps': [{'action': None,
            'id': 'd413a19dec13d11e',
            'job_id': None,
            'model_class': 'WorkflowInvocationStep',
            'order_index': 0,
            'state': None,
            'update_time': '2015-10-31T22:00:26',
            'workflow_step_id': 'cbbbf59e8f08c98c',
            'workflow_step_label': None,
            'workflow_step_uuid': 'b81250fd-3278-4e6a-b269-
↪56a1f01ef485'},
           {'action': None,
            'id': '2f94e8ae9edff68a',
            'job_id': 'e89067bb68bee7a0',
            'model_class': 'WorkflowInvocationStep',
            'order_index': 1,
            'state': 'new',
            'update_time': '2015-10-31T22:00:26',
            'workflow_step_id': '964b37715ec9bd22',
            'workflow_step_label': None,
            'workflow_step_uuid': 'e62440b8-e911-408b-b124-
↪e05435d3125e'}],
 'update_time': '2015-10-31T22:00:26',
 'uuid': 'c8aa2b1c-801a-11e5-a9e5-8ca98228593c',
 'workflow_id': '03501d7626bd192f'}
```

The `params` dict should be specified as follows:

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

where PARAM_DICT is:

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

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

```
{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:

```
{'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:

```
<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" min="0" value="0" label="Min value"/>
</repeat>
```

then the PARAM_DICT should be something like:

```
{...
 "cutoff_0|name": "n_genes",
 "cutoff_0|min": "2",
 "cutoff_1|name": "n_counts",
 "cutoff_1|min": "4",
 ...}
```

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:

```
{'RenameDatasetActionout_file1': {'action_arguments': {'newname': '${output}'},
                                  'action_type': 'RenameDatasetAction',
                                  'output_name': 'out_file1'}}
```

then the following renames the output dataset to 'foo':

```
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.

module = 'workflows'

refactor_workflow(*workflow_id*, *actions*, *dry_run=False*)

Refactor workflow with given actions.

Parameters

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

- **actions** (*list of dicts*) –

  Actions to use for refactoring the workflow. The following actions are supported: update_step_label, update_step_position, update_output_label, update_name, update_annotation, update_license, update_creator, update_report, add_step, add_input, disconnect, connect, fill_defaults, fill_step_defaults, extract_input, extract_legacy_parameter, remove_unlabeled_workflow_outputs, upgrade_all_steps, upgrade_subworkflow, upgrade_tool.

  An example value for the `actions` argument might be:

```
actions = [
    {"action_type": "add_input", "type": "data", "label": "foo"},
    {"action_type": "update_step_label", "label": "bar", "step": {
↪"label": "foo"}},
]
```

- **dry_run** (*bool*) – When true, perform a dry run where the existing workflow is preserved. The refactored workflow is returned in the output of the method, but not saved on the Galaxy server.

Return type dict

Returns Dictionary containing logged messages for the executed actions and the refactored workflow.

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': '29beef4fadeed09f', '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:

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

The `params` dict should be specified as follows:

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

where PARAM_DICT is:

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

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

```
{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:

```
{'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 runtime values. For instance, if the final step has a PJA like the following:

```
{'RenameDatasetActionout_file1': {'action_arguments': {'newname': '${output}'},
                                  'action_type': 'RenameDatasetAction',
                                  'output_name': 'out_file1'}}
```

then the following renames the output dataset to 'foo':

```
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:

```
{'history_id': '2f94e8ae9edff68a',
 'id': 'df7a1f0c02a5b08e',
 'inputs': {'0': {'id': 'a7db2fac67043c7e',
                  'src': 'hda',
                  'uuid': '7932ffe0-2340-4952-8857-dbaa50f1f46a'}},
 'model_class': 'WorkflowInvocation',
 'state': 'ready',
 'steps': [{'action': None,
            'id': 'd413a19dec13d11e',
            'job_id': None,
            'model_class': 'WorkflowInvocationStep',
            'order_index': 0,
            'state': None,
            'update_time': '2015-10-31T22:00:26',
            'workflow_step_id': 'cbbbf59e8f08c98c',
            'workflow_step_label': None,
            'workflow_step_uuid': 'b81250fd-3278-4e6a-b269-
↪56a1f01ef485'},
           {'action': None,
            'id': '2f94e8ae9edff68a',
            'job_id': 'e89067bb68bee7a0',
            'model_class': 'WorkflowInvocationStep',
            'order_index': 1,
            'state': 'new',
            'update_time': '2015-10-31T22:00:26',
            'workflow_step_id': '964b37715ec9bd22',
            'workflow_step_label': None,
            'workflow_step_uuid': 'e62440b8-e911-408b-b124-
↪e05435d3125e'}],
 'update_time': '2015-10-31T22:00:26',
 'uuid': 'c8aa2b1c-801a-11e5-a9e5-8ca98228593c',
 'workflow_id': '03501d7626bd192f'}
```

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:

```
{'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_versions(*workflow_id*)
Get versions for a workflow.

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

Return type list of dicts

Returns Ordered list of version descriptions for this workflow

show_workflow(*workflow_id*, *version=None*)
Display information needed to run a workflow.

Parameters

- **workflow_id** (*str*) – Encoded workflow ID
- **version** (*int*) – Workflow version to show

Return type dict

Returns

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

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

update_workflow(*workflow_id*, ***kwds*)
Update a given workflow.

Parameters

- **workflow_id** (*str*) – Encoded workflow ID
- **workflow** (*dict*) – dictionary representing the workflow to be updated
- **name** (*str*) – New name of the workflow

- **annotation** (*str*) – New annotation for the workflow
- **menu_entry** (*bool*) – Whether the workflow should appear in the user's menu
- **tags** (*list of str*) – Replace workflow tags with the given list
- **published** (*bool*) – Whether the workflow should be published or unpublished

Return type dict

Returns Dictionary representing the updated workflow

5.1.2 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':

```
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*.

class bioblend.galaxy.objects.client.**ObjClient**(*obj_gi*)

abstract get(*id_*) → *bioblend.galaxy.objects.wrappers.Wrapper*

Retrieve the object corresponding to the given id.

abstract get_previews() → *list*

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

abstract list() → *list*

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

class bioblend.galaxy.objects.client.**ObjDatasetContainerClient**(*obj_gi*)

class bioblend.galaxy.objects.client.**ObjHistoryClient**(*obj_gi*)

Interacts with Galaxy histories.

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.yml` 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.**ObjInvocationClient**(*obj_gi*)

Interacts with Galaxy Invocations.

get(*id_*) → bioblend.galaxy.objects.wrappers.Invocation

Get an invocation by ID.

Return type Invocation

Param invocation object

get_previews() → List[bioblend.galaxy.objects.wrappers.InvocationPreview]
Get previews of all invocations.

Return type list of InvocationPreview

Param previews of invocations

list(*workflow=None*, *history=None*, *include_terminal=True*, *limit=None*) → List[bioblend.galaxy.objects.wrappers.Invocation]
Get full listing of workflow invocations, or select a subset by specifying optional arguments for filtering (e.g. a workflow).

Parameters

- **workflow** (wrappers.Workflow) – Include only invocations associated with this workflow
- **history** (*str*) – Include only invocations associated with this history
- **include_terminal** – bool
- **limit** (*int*) – Maximum number of invocations to return - if specified, the most recent invocations will be returned.

Param Whether to include invocations in terminal states

Return type list of Invocation

Param invocation objects

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.

class bioblend.galaxy.objects.wrappers.**Dataset**(**args*, ***kwargs*)
Abstract base class for Galaxy datasets.

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: Tuple[str, ...] = ('id', 'data_type', 'file_ext', 'file_name', 'file_size', 'genome_build', 'misc_info', 'name', 'state')

POLLING_INTERVAL = 1

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.**DatasetCollection**(**args*, ***kwargs*)

Abstract base class for Galaxy dataset collections.

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: Tuple[str, ...] = ('id', 'collection_type', 'deleted', 'name', 'state')

abstract delete()

refresh()

Re-fetch the attributes pertaining to this object.

Returns: self

class bioblend.galaxy.objects.wrappers.**DatasetContainer**(**args*, ***kwargs*)

Abstract base class for dataset containers (histories and libraries).

Parameters **content_infos** (list of `ContentInfo`) – info objects for the container's contents

abstract property API_MODULE

BASE_ATTRS: Tuple[str, ...] = ('id', 'deleted', 'name')

property 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.**Folder**(**args*, ***kwargs*)
Maps to a folder in a Galaxy library.

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: Tuple[str, ...] = ('id', 'deleted', 'description', 'item_count', 'name')

property 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

class bioblend.galaxy.objects.wrappers.**History**(**args*, ***kwargs*)
Maps to a Galaxy history.

Parameters **content_infos** (list of ContentInfo) – info objects for the container's contents

API_MODULE = 'histories'

BASE_ATTRS: Tuple[str, ...] = ('id', 'deleted', 'name', 'annotation', 'published', 'state', 'state_ids', 'state_details', 'tags')

CONTENT_INFO_TYPE
alias of *bioblend.galaxy.objects.wrappers.HistoryContentInfo*

DSC_TYPE
alias of *bioblend.galaxy.objects.wrappers.HistoryDatasetCollectionAssociation*

DS_TYPE
alias of *bioblend.galaxy.objects.wrappers.HistoryDatasetAssociation*

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

delete(*purge=False*)
Delete this history.

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.yml` configuration file.

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.

get_dataset_collection(*dsc_id*)
Retrieve the dataset collection corresponding to the given id.

Parameters **dsc_id** (*str*) – dataset collection id

Return type *HistoryDatasetCollectionAssociation*

Returns the dataset collection corresponding to `dsc_id`

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(***kwds*)
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).
- **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.**HistoryContentInfo**(**args*, ***kwargs*)
Instances of this class wrap dictionaries obtained by getting `/api/histories/<ID>/contents` from Galaxy.

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: Tuple[str, ...] = ('id', 'name', 'type', 'deleted', 'state', 'visible')

class bioblend.galaxy.objects.wrappers.**HistoryDatasetAssociation**(**args*, ***kwargs*)
Maps to a Galaxy `HistoryDatasetAssociation`.

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: Tuple[str, ...] = ('id', 'data_type', 'file_ext', 'file_name', 'file_size', 'genome_build', 'misc_info', 'name', '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.yml` configuration file.

---

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.**HistoryDatasetCollectionAssociation**(**args*, ***kwargs*)

Maps to a Galaxy `HistoryDatasetCollectionAssociation`.

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: Tuple[str, ...] = ('id', 'collection_type', 'deleted', 'name', 'state', 'tags', 'visible', 'elements')

SRC = 'hdca'

delete()

Delete this dataset collection.

class bioblend.galaxy.objects.wrappers.**HistoryPreview**(**args*, ***kwargs*)

Models Galaxy history 'previews'.

Instances of this class wrap dictionaries obtained by getting `/api/histories` from Galaxy.

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: Tuple[str, ...] = ('id', 'deleted', 'name', 'annotation', 'published', 'purged', 'tags')

class bioblend.galaxy.objects.wrappers.**Job**(**args*, ***kwargs*)

Maps to a Galaxy job.

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: Tuple[str, ...] = ('id', 'state')

class bioblend.galaxy.objects.wrappers.**Library**(**args*, ***kwargs*)

Maps to a Galaxy library.

Parameters **content_infos** (list of `ContentInfo`) – info objects for the container's contents

API_MODULE = 'libraries'

BASE_ATTRS: Tuple[str, ...] = ('id', 'deleted', 'name', 'description', 'synopsis')

CONTENT_INFO_TYPE
alias of *bioblend.galaxy.objects.wrappers.LibraryContentInfo*

DS_TYPE
alias of *bioblend.galaxy.objects.wrappers.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.

property 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`

property root_folder
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` option set to `true` in the `config/galaxy.yml` 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.

class bioblend.galaxy.objects.wrappers.**LibraryContentInfo**(**args*, ***kwargs*)
Instances of this class wrap dictionaries obtained by getting `/api/libraries/<ID>/contents` from Galaxy.

Parameters

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

class bioblend.galaxy.objects.wrappers.**LibraryDataset**(**args*, ***kwargs*)
Maps to a Galaxy `LibraryDataset`.

Parameters

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

SRC = 'ld'

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.**LibraryDatasetDatasetAssociation**(**args*, ***kwargs*)
Maps to a Galaxy `LibraryDatasetDatasetAssociation`.

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: Tuple[str, ...] = ('id', 'data_type', 'file_ext', 'file_name', 'file_size', 'genome_build', 'misc_info', 'name', 'state', 'deleted')

SRC = 'ldda'

class bioblend.galaxy.objects.wrappers.**LibraryPreview**(**args*, ***kwargs*)
Models Galaxy library 'previews'.

Instances of this class wrap dictionaries obtained by getting `/api/libraries` from Galaxy.

Parameters

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

class bioblend.galaxy.objects.wrappers.**Step**(**args*, ***kwargs*)
Workflow step.

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

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: Tuple[str, ...] = ('id', 'input_steps', 'name', 'tool_id', 'tool_inputs', 'tool_version', 'type')

class bioblend.galaxy.objects.wrappers.**Tool**(**args*, ***kwargs*)
Maps to a Galaxy tool.

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: Tuple[str, ...] = ('id', 'name', 'version')

POLLING_INTERVAL = 10

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.**Workflow**(**args*, ***kwargs*)

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.

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: Tuple[str, ...] = ('id', 'deleted', 'inputs', 'latest_workflow_uuid', 'name', 'owner', 'published', 'steps', 'tags')

POLLING_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.

property data_collection_input_ids

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

property 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

property input_labels

Return the labels of this workflow's input steps.

invoke(*inputs=None*, *params=None*, *history=None*, *import_inputs_to_history=None*, *replacement_params=None*, *allow_tool_state_corrections=True*, *inputs_by=None*, *parameters_normalized=False*)

Invoke the workflow. This will cause a workflow to be scheduled and return an object describing the workflow invocation.

Parameters

- **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': '29beef4fadeed09f', '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. Alternatively workflow steps may be addressed by the label that can be set in the workflow editor. If using uuid or label you need to also set the `inputs_by` parameter to `step_uuid` or `name`.

- **params** (*dict*) – A mapping of non-datasets tool parameters (see below)
- **history** (*str*) – The history in which to store the workflow output.
- **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)
- **inputs_by** (*str*) – Determines how inputs are referenced. Can be "step_index|step_uuid" (default), "step_index", "step_id", "step_uuid", or "name".
- **parameters_normalized** (*bool*) – Whether Galaxy should normalize `params` to ensure everything is referenced by a numeric step ID. Default is `False`, but when setting `params` for a subworkflow, `True` is required.

Return type Invocation

Returns the workflow invocation

The `params` dict should be specified as follows:

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

where PARAM_DICT is:

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

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

```
{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:

```
{'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:

```
<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" min="0" value="0" label="Min value"/>
</repeat>
```

then the PARAM_DICT should be something like:

```
{...
 "cutoff_0|name": "n_genes",
 "cutoff_0|min": "2",
 "cutoff_1|name": "n_counts",
 "cutoff_1|min": "4",
 ...}
```

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:

```
{'RenameDatasetActionout_file1': {'action_arguments': {'newname': '${output}'},
                                  'action_type': 'RenameDatasetAction',
                                  'output_name': 'out_file1'}}
```

then the following renames the output dataset to 'foo':

```
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.

property 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.

property parameter_input_ids
Return the ids of parameter input steps for this workflow.

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.

Deprecated since version 0.16.0: Use *invoke()* instead.

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:

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

where PARAM_DICT is:

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

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

```
{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:

```
{'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:

```
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.

property tool_ids
Return the ids of tool steps for this workflow.

class bioblend.galaxy.objects.wrappers.**WorkflowPreview**(**args*, ***kwargs*)
Models Galaxy workflow 'previews'.

Instances of this class wrap dictionaries obtained by getting `/api/workflows` from Galaxy.

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: Tuple[str, ...] = ('id', 'deleted', 'latest_workflow_uuid', 'name', 'number_of_steps', 'owner', 'published', 'show_in_tool_panel', 'tags')

class bioblend.galaxy.objects.wrappers.**Wrapper**(**args*, ***kwargs*)
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'] -> 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: Tuple[str, ...] = ('id',)

clone()
Return an independent copy of this wrapper.

classmethod from_json(*jdef*)
Build a new wrapper from a JSON dump.

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

property 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.

5.1.3 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/ .

To open a connection call:

```
from bioblend.galaxy import GalaxyInstance

gi = 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:

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

```
>>> gi.histories.show_history('f3c2b0f3ecac9f02', contents=False)
{'annotation': '',
 'contents_url': '/api/histories/f3c2b0f3ecac9f02/contents',
 'id': 'f3c2b0f3ecac9f02',
 'name': 'RNAseq_DGE_BASIC_Prep',
 'nice_size': '93.5 MB',
 'state': 'ok',
 'state_details': {'discarded': 0,
                   'empty': 0,
```

(continues on next page)

(continued from previous page)

```
                   'error': 0,
                   'failed_metadata': 0,
                   'new': 0,
                   'ok': 7,
                   'paused': 0,
                   'queued': 0,
                   'running': 0,
                   'setting_metadata': 0,
                   'upload': 0},
 'state_ids': {'discarded': [],
               'empty': [],
               'error': [],
               'failed_metadata': [],
               'new': [],
               'ok': ['d6842fb08a76e351',
                      '10a4b652da44e82a',
                      '81c601a2549966a0',
                      'a154f05e3bcee26b',
                      '1352fe19ddce0400',
                      '06d549c52d753e53',
                      '9ec54455d6279cc7'],
               'paused': [],
               'queued': [],
               'running': [],
               'setting_metadata': [],
               'upload': []}}
```

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')
{'data_type': 'fastqsanger',
 'deleted': False,
 'file_size': 16527060,
 'genome_build': 'dm3',
 'id': 17499,
 'metadata_data_lines': None,
 'metadata_dbkey': 'dm3',
 'metadata_sequences': None,
 'misc_blurb': '15.8 MB',
 'misc_info': 'Noneuploaded fastqsanger file',
 'model_class': 'HistoryDatasetAssociation',
 'name': 'C1_R2_1.chr4.fq',
 'purged': False,
 'state': 'ok',
 'visible': True}
```

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:

```
>>> gi.tools.upload_file('test.txt', 'f3c2b0f3ecac9f02')
{'implicit_collections': [],
 'jobs': [{'create_time': '2015-07-28T17:52:39.756488',
           'exit_code': None,
           'id': '9752b387803d3e1e',
           'model_class': 'Job',
           'state': 'new',
           'tool_id': 'upload1',
           'update_time': '2015-07-28T17:52:39.987509'}],
 'output_collections': [],
 'outputs': [{'create_time': '2015-07-28T17:52:39.331176',
              'data_type': 'galaxy.datatypes.data.Text',
              'deleted': False,
              'file_ext': 'auto',
              'file_size': 0,
              'genome_build': '?',
              'hda_ldda': 'hda',
              'hid': 16,
              'history_content_type': 'dataset',
              'history_id': 'f3c2b0f3ecac9f02',
              'id': '59c76a119581e190',
              'metadata_data_lines': None,
              'metadata_dbkey': '?',
              'misc_blurb': None,
              'misc_info': None,
              'model_class': 'HistoryDatasetAssociation',
              'name': 'test.txt',
              'output_name': 'output0',
              'peek': '<table cellspacing="0" cellpadding="3"></table>',
              'purged': False,
              'state': 'queued',
              'tags': [],
              'update_time': '2015-07-28T17:52:39.611887',
              'uuid': 'ff0ee99b-7542-4125-802d-7a193f388e7e',
              '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:

```
>>> gi.tools.upload_from_ftp('test.txt', 'f3c2b0f3ecac9f02')
{'implicit_collections': [],
 'jobs': [{'create_time': '2015-07-28T17:57:43.704394',
           'exit_code': None,
           'id': '82b264d8c3d11790',
           'model_class': 'Job',
           'state': 'new',
           'tool_id': 'upload1',
           'update_time': '2015-07-28T17:57:43.910958'}],
 'output_collections': [],
 'outputs': [{'create_time': '2015-07-28T17:57:43.209041',
```

(continues on next page)

(continued from previous page)

```
              'data_type': 'galaxy.datatypes.data.Text',
              'deleted': False,
              'file_ext': 'auto',
              'file_size': 0,
              'genome_build': '?',
              'hda_ldda': 'hda',
              'hid': 17,
              'history_content_type': 'dataset',
              'history_id': 'f3c2b0f3ecac9f02',
              'id': 'a676e8f07209a3be',
              'metadata_data_lines': None,
              'metadata_dbkey': '?',
              'misc_blurb': None,
              'misc_info': None,
              'model_class': 'HistoryDatasetAssociation',
              'name': 'test.txt',
              'output_name': 'output0',
              'peek': '<table cellspacing="0" cellpadding="3"></table>',
              'purged': False,
              'state': 'queued',
              'tags': [],
              'update_time': '2015-07-28T17:57:43.544407',
              'uuid': '2cbe8f0a-4019-47c4-87e2-005ce35b8449',
              '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:

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

```
>>> gi.libraries.show_library('8e6f930d00d123ea')
{'contents_url': '/api/libraries/8e6f930d00d123ea/contents',
 'description': 'RNA-Seq workshop data',
 'name': 'RNA-Seq',
 'synopsis': '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:

```
>>> 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:

```
>>> 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:

```
>>> 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'},
           '251': {'id': 251,
                   'input_steps': {'input': {'source_step': 250,
                                             'step_output': 'accepted_hits'}},
                   'tool_id': 'cufflinks',
                   'type': 'tool'},
           '252': {'id': 252,
                   'input_steps': {},
                   'tool_id': None,
                   'type': 'data_input'}},
 'url': '/api/workflows/e8b85ad72aefca86'}
```

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:

```
>>> 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:

```
>>> gi.workflows.import_workflow_dict(workflow_dict)
{'id': 'c0bacafdfe211f9a',
 'name': 'TopHat + cufflinks part 1 (imported from API)',
 'url': '/api/workflows/c0bacafdfe211f9a'}
```

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:

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

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']
{'252': {'label': 'Input RNA-seq fastq', 'value': ''}}
```

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 = {'252': {'src':'hda', 'id':'10a4b652da44e82a'}}
>>> gi.workflows.run_workflow('e8b85ad72aefca86', datamap, history_name='New output␣
↪history')
{'history': '0a7b7992a7cabaec',
 'outputs': ['33be8ad9917d9207',
             'fbee1c2dc793c114',
             '85866441984f9e28',
             '1c51aa78d3742386',
             'a68e8770e52d03b4',
             'c54baf809e3036ac',
             'ba0db8ce6cd1fe8f',
             'c019e4cf08b2ac94']}
```

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:

```
>>> gi.histories.show_history('0a7b7992a7cabaec')
{'annotation': '',
 'contents_url': '/api/histories/0a7b7992a7cabaec/contents',
 'id': '0a7b7992a7cabaec',
 'name': 'New output history',
 'nice_size': '0 bytes',
 'state': 'queued',
 'state_details': {'discarded': 0,
                   'empty': 0,
                   'error': 0,
                   'failed_metadata': 0,
                   'new': 0,
                   'ok': 0,
                   'paused': 0,
                   'queued': 8,
                   'running': 0,
                   'setting_metadata': 0,
                   'upload': 0},
 'state_ids': {'discarded': [],
               'empty': [],
               'error': [],
               'failed_metadata': [],
               'new': [],
               'ok': [],
               'paused': [],
```

(continues on next page)

(continued from previous page)

```
                'queued': ['33be8ad9917d9207',
                           'fbee1c2dc793c114',
                           '85866441984f9e28',
                           '1c51aa78d3742386',
                           'a68e8770e52d03b4',
                           'c54baf809e3036ac',
                           'ba0db8ce6cd1fe8f',
                           'c019e4cf08b2ac94'],
                'running': [],
                'setting_metadata': [],
                'upload': []}}
```

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:

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

Using BioBlend for raw API calls

BioBlend can be used to make HTTP requests to the Galaxy API in a more convenient way than using e.g. the `requests` Python library. There are 5 available methods corresponding to the most common HTTP methods: `make_get_request`, `make_post_request`, `make_put_request`, `make_delete_request` and `make_patch_request`. One advantage of using these methods is that the API keys stored in the `GalaxyInstance` object is automatically added to the request.

To make a GET request to the Galaxy API with BioBlend, call:

```
>>> gi.make_get_request(gi.base_url + "/api/version").json()
{'version_major': '19.05',
 'extra': {}}
```

To make a POST request to the Galaxy API with BioBlend, call:

```
>>> gi.make_post_request(gi.base_url + "/api/histories", payload={"name": "test history"}
↪)
{'importable': False,
 'create_time': '2019-07-05T20:10:04.823716',
```

(continues on next page)

(continued from previous page)

```
'contents_url': '/api/histories/a77b3f95070d689a/contents',
'id': 'a77b3f95070d689a',
'size': 0, 'user_id': '5b732999121d4593',
'username_and_slug': None,
'annotation': None,
'state_details': {'discarded': 0,
                  'ok': 0,
                  'failed_metadata': 0,
                  'upload': 0,
                  'paused': 0,
                  'running': 0,
                  'setting_metadata': 0,
                  'error': 0,
                  'new': 0,
                  'queued': 0,
                  'empty': 0},
'state': 'new',
'empty': True,
'update_time': '2019-07-05T20:10:04.823742',
'tags': [],
'deleted': False,
'genome_build': None,
'slug': None,
'name': 'test history',
'url': '/api/histories/a77b3f95070d689a',
'state_ids': {'discarded': [],
              'ok': [],
              'failed_metadata': [],
              'upload': [],
              'paused': [],
              'running': [],
              'setting_metadata': [],
              'error': [],
              'new': [],
              'queued': [],
              'empty': []},
'published': False,
'model_class': 'History',
'purged': False}
```

5.2 Toolshed API

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

5.2.1 API documentation for interacting with the Galaxy Toolshed

ToolShedInstance

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:

```
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:

```
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*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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:

```
[{'deleted': False,
  'description': 'Tools for manipulating data',
  'id': '175812cd7caaf439',
  'model_class': 'Category',
  'name': 'Text Manipulation',
  'url': '/api/categories/175812cd7caaf439'}]
```

New in version 0.5.2.

module = 'categories'

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*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

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:


```
{"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:

```
[{'category_ids': ['c1df3132f6334b0e', 'f6d7b0037d901d9b'],
  'deleted': False,
  'deprecated': False,
  'description': 'Order Contigs',
  'homepage_url': '',
  'id': '287bd69f724b99ce',
  'name': 'best_tool_ever',
  'owner': 'billybob',
  'private': False,
  'remote_repository_url': '',
  'times_downloaded': 0,
  'type': 'unrestricted',
  'url': '/api/repositories/287bd69f724b99ce',
  'user_id': '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:

```
[{'deleted': False,
  'deprecated': False,
  'description': 'Galaxy Freebayes Bayesian genetic variant detector␣
↪tool',
```

(continues on next page)

(continued from previous page)

```
 'homepage_url': '',
 'id': '491b7a3fddf9366f',
 'long_description': 'Galaxy Freebayes Bayesian genetic variant␣
↪detector tool originally included in the Galaxy code distribution␣
↪but migrated to the tool shed.',
 'name': 'freebayes',
 'owner': 'devteam',
 'private': False,
 'remote_repository_url': '',
 'times_downloaded': 269,
 'type': 'unrestricted',
 'url': '/api/repositories/491b7a3fddf9366f',
 'user_id': '1de29d50c3c44272'},
{'changeset_revision': 'd291dc763c4c',
 'do_not_test': False,
 'downloadable': True,
 'has_repository_dependencies': False,
 'id': '504be8aaa652c154',
 'includes_datatypes': False,
 'includes_tool_dependencies': True,
 'includes_tools': True,
 'includes_tools_for_display_in_tool_panel': True,
 'includes_workflows': False,
 'malicious': False,
 'repository_id': '491b7a3fddf9366f',
 'url': '/api/repository_revisions/504be8aaa652c154'},
{'freebayes': ['Galaxy Freebayes Bayesian genetic variant detector␣
↪tool',
               'http://testtoolshed.g2.bx.psu.edu/repos/devteam/
↪freebayes',
               'd291dc763c4c',
               '9',
               'devteam',
               {},
               {'freebayes/0.9.6_
↪9608597d12e127c847ae03aa03440ab63992fedf': {'changeset_revision':
↪'d291dc763c4c',
                                                                    ␣
↪        'name': 'freebayes',                                       ␣
↪        'repository_name': 'freebayes',                            ␣
↪        'repository_owner': 'devteam',                             ␣
↪        'type': 'package',                                         ␣
↪        'version': '0.9.6_9608597d12e127c847ae03aa03440ab63992fedf'}
↪,
                'samtools/0.1.18': {'changeset_revision':
↪'d291dc763c4c',
                                    'name': 'samtools',
                                    'repository_name': 'freebayes',
```

(continues on next page)

(continued from previous page)

```
                                        'repository_owner': 'devteam',
                                        'type': 'package',
                                        'version': '0.1.18'}}]}]
```

module = 'repositories'

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:

```
[{'changeset_revision': '6e26c5a48e9a',
  'do_not_test': False,
  'downloadable': True,
  'has_repository_dependencies': False,
  'id': '92250afff777a169',
  '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': '78f2604ff5e65707',
  'test_install_error': False,
  'time_last_tested': None,
  'tools_functionally_correct': False,
  'url': '/api/repository_revisions/92250afff777a169'},
 {'changeset_revision': '15a54fa11ad7',
  'do_not_test': False,
  'downloadable': True,
  'has_repository_dependencies': False,
  'id': 'd3823c748ae2205d',
```

(continues on next page)

(continued from previous page)

```
  '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': 'f9662009da7bfce0',
  'test_install_error': False,
  'time_last_tested': None,
  'tools_functionally_correct': False,
  'url': '/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:

```
{'hits': [{'matched_terms': [],
           'repository': {'approved': 'no',
                          'description': 'Convert export file to fastq
↪',
                          'full_last_updated': '2015-01-18 09:48 AM',
                          'homepage_url': '',
                          'id': 'bdfa208f0cf6504e',
                          'last_updated': 'less than a year',
                          'long_description': 'This is a simple too␣
↪to convert Solexas Export files to FASTQ files.',
                          'name': 'export_to_fastq',
                          'remote_repository_url': '',
                          'repo_owner_username': 'louise',
                          'times_downloaded': 164},
           'score': 4.92},
          {'matched_terms': [],
           'repository': {'approved': 'no',
                          'description': 'Convert BAM file to fastq',
                          'full_last_updated': '2015-04-07 11:57 AM',
                          'homepage_url': '',
                          'id': '175812cd7caaf439',
                          'last_updated': 'less than a month',
                          'long_description': 'Use Picards SamToFastq␣
↪to convert a BAM file to fastq. Useful for storing reads as BAM in␣
↪Galaxy and converting to fastq when needed for analysis.',
                          'name': 'bam_to_fastq',
```

(continues on next page)

(continued from previous page)

```
                          'remote_repository_url': '',
                          'repo_owner_username': 'brad-chapman',
                          'times_downloaded': 138},
          'score': 4.14}],
 'hostname': 'https://testtoolshed.g2.bx.psu.edu/',
 'page': '1',
 'page_size': '2',
 'total_results': '64'}
```

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:

```
{'category_ids': ['c1df3132f6334b0e', 'f6d7b0037d901d9b'],
 'deleted': False,
 'deprecated': False,
 'description': 'Order Contigs',
 'homepage_url': '',
 'id': '287bd69f724b99ce',
 'long_description': '',
 'name': 'best_tool_ever',
 'owner': 'billybob',
 'private': False,
 'remote_repository_url': '',
 'times_downloaded': 0,
 'type': 'unrestricted',
 'url': '/api/repositories/287bd69f724b99ce',
 'user_id': '5cefd48bc04af6d4'}
```

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:

```
{'changeset_revision': '7602de1e7f32',
 'do_not_test': False,
 'downloadable': True,
 'has_repository_dependencies': False,
 'id': '504be8aaa652c154',
 'includes_datatypes': False,
```

(continues on next page)

(continued from previous page)

```
'includes_tool_dependencies': False,
'includes_tools': True,
'includes_tools_for_display_in_tool_panel': True,
'includes_workflows': False,
'malicious': False,
'missing_test_components': True,
'repository_id': '491b7a3fddf9366f',
'test_install_error': False,
'time_last_tested': None,
'tool_test_results': {'missing_test_components': []},
'tools_functionally_correct': False,
'url': '/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:

```
{'content_alert': '',
 'message': ''}
```

New in version 0.5.2.

Tools

Interaction with a Tool Shed instance tools

class bioblend.toolshed.tools.**ToolShedToolClient**(*toolshed_instance*)

A generic Client interface defining the common fields.

All clients *must* define the following field (which will be used as part of the URL composition (e.g., `http://<galaxy_instance>/api/libraries`): `self.module = 'workflows' | 'libraries' | 'histories' | ...`

module = 'tools'

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:

```
{'hits': [{'matched_terms': [],
           'score': 3.0,
           'tool': {'description': 'convert between various FASTQ␣
↪quality formats',
                    'id': '69819b84d55f521efda001e0926e7233',
                    'name': 'FASTQ Groomer',
                    'repo_name': None,
                    'repo_owner_username': 'devteam'}},
          {'matched_terms': [],
           'score': 3.0,
           'tool': {'description': 'converts a bam file to fastq␣
↪files.',
                    'id': '521e282770fd94537daff87adad2551b',
                    'name': 'Defuse BamFastq',
                    'repo_name': None,
                    'repo_owner_username': 'jjohnson'}}],
 'hostname': 'https://testtoolshed.g2.bx.psu.edu/',
 'page': '1',
 'page_size': '2',
 'total_results': '118'}
```

5.3 CloudMan API

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

5.3.1 API documentation for interacting with CloudMan

CloudManLauncher

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.

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 eixsted).

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 sa 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 copatible 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=None*, *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 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=None*, *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 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.
- **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, 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, float or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

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.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is `None` and (',', ': ') otherwise. 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`.

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:

```
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=None*, *placement=''*, *block_until_ready=False*)

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(*num_nodes*, *instance_type=''*, *spot_price=''*)

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(*minimum_nodes=None*, *maximum_nodes=None*)

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()

Returns a boolean indicating whether autoscaling is enabled.

property cloudman_url

Returns the URL for accessing this instance of CloudMan.

disable_autoscaling()

Disable autoscaling, meaning that worker nodes will need to be manually added and removed.

enable_autoscaling(*minimum_nodes=0*, *maximum_nodes=19*)

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.

property galaxy_url

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

get_cloudman_version()

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()

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

get_cluster_type()

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

get_galaxy_state()

Get the current status of Galaxy running on the cluster.

get_master_id()

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

get_master_ip()

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

get_nodes()

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

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

get_status()
Get status information on this CloudMan instance.

initialize(*cluster_type*, *galaxy_data_option=''*, *initial_storage_size=None*, *shared_bucket=None*)
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()
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(*instance_id*)
Reboot a specific worker node.

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

remove_node(*instance_id*, *force=False*)
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(*num_nodes*, *force=False*)
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(*enable*)
Enables/disables master as execution host.

terminate(*terminate_master_instance=True*, *delete_cluster=False*)
Terminate 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()
Update 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*)
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()
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).

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

property key_pair_material
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.

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

wait_until_instance_ready(*vm_ready_timeout=300*, *vm_ready_check_interval=10*)
Wait until the VM state changes to ready/error or timeout elapses. Updates the host name once ready.

exception bioblend.cloudman.**VMLaunchException**(*value*)

bioblend.cloudman.**block_until_vm_ready**(*func*)
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** (*int*) – Maximum length of time to block before timing out. Once the timeout is reached, a VMLaunchException will be thrown.
- **vm_ready_check_interval** (*int*) – The number of seconds to pause between consecutive calls when polling the VM's ready status.

5.3.2 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:

```
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):

```
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:

```
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:

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

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

Manipulating an existing cluster

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

```
>>> cmi.initialized
False
>>> cmi.initialize('SGE')
>>> cmi.get_status()
{'all_fs': [],
 'app_status': 'yellow',
 'autoscaling': {'as_max': 'N/A',
 'as_min': 'N/A',
 'use_autoscaling': False},
 'cluster_status': 'STARTING',
 'data_status': 'green',
 'disk_usage': {'pct': '0%', 'total': '0', 'used': '0'},
 'dns': '#',
 'instance_status': {'available': '0', 'idle': '0', 'requested': '0'},
 'snapshot': {'progress': 'None', 'status': 'None'}}
>>> cmi.get_cluster_size()
1
>>> cmi.get_nodes()
[{'id': 'i-00006016',
  'instance_type': 'm1.medium',
  'ld': '0.0 0.025 0.065',
  'public_ip': '115.146.86.29',
  'time_in_state': '2268'}]
>>> cmi.add_nodes(2)
{'all_fs': [],
 'app_status': 'green',
 'autoscaling': {'as_max': 'N/A',
  'as_min': 'N/A',
  'use_autoscaling': False},
 'cluster_status': 'READY',
 'data_status': 'green',
 'disk_usage': {'pct': '0%', 'total': '0', 'used': '0'},
 'dns': '#',
 'instance_status': {'available': '0', 'idle': '0', 'requested': '2'},
 'snapshot': {'progress': 'None', 'status': 'None'}}
>>> cmi.get_cluster_size()
3
```

CHAPTER

SIX

CONFIGURATION

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 NotImplementedError.

exception bioblend.**TimeoutException**

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*)

CHAPTER

SEVEN

TESTING

If you would like to do more than just a mock test, you need 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
```

CHAPTER

EIGHT

GETTING HELP

If you have 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 the Github Issues page to ask your question.

CHAPTER

NINE

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

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W