Galaxy Code Documentation

Release

Galaxy Team

May 13, 2015

Contents

Galaxy is an open, web-based platform for accessible, reproducible, and transparent computational biomedical research.

- *Accessible:* Users without programming experience can easily specify parameters and run tools and workflows.
- *Reproducible:* Galaxy captures information so that any user can repeat and understand a complete computational analysis.
- *Transparent:* Users share and publish analyses via the web and create Pages, interactive, web-based documents that describe a complete analysis.

Two copies of the Galaxy code doumentation are published by the Galaxy Project

- Galaxy-Dist: This describes the code in the most recent official release of Galaxy.
- Galaxy-Central: Describes the current code in the development branch of Galaxy. This is the latest checkin, bleeding edge version of the code. The documentation should never be more than an hour behind the code.

Both copies are hosted at ReadTheDocs, a publicly supported web site for hosting project documentation.

If you have your own copy of the Galaxy source code, you can also generate your own version of this documentation:

```
$ cd doc
$ make html
```

The generated documentation will be in `doc/build/html/` and can be viewed with a web browser. Note that you will need to install Sphinx and a fair number of module dependencies before this will produce output.

For more on the Galaxy Project, please visit the project home page.

CHAPTER 1

Contents

1.1 Galaxy API Documentation

1.1.1 Background

In addition to being accessible through a web interface, Galaxy can also be accessed programmatically, through shell scripts and other programs. The web interface is appropriate for things like exploratory analysis, visualization, construction of workflows, and rerunning workflows on new datasets.

The web interface is less suitable for things like

- Connecting a Galaxy instance directly to your sequencer and running workflows whenever data is ready.
- Running a workflow against multiple datasets (which can be done with the web interface, but is tedious).
- When the analysis involves complex control, such as looping and branching.

The Galaxy API addresses these and other situations by exposing Galaxy internals through an additional interface, known as an Application Programming Interface, or API.

1.1.2 Quickstart

Log in as your user, navigate to the API Keys page in the User menu, and generate a new API key. Make a note of the API key, and then pull up a terminal. Now we'll use the display.py script in your galaxy/scripts/api directory for a short example:

```
% ./display.py my_key http://localhost:4096/api/histories
Collection Members
------------------
#1: /api/histories/8c49be448cfe29bc
  name: Unnamed history
  id: 8c49be448cfe29bc
#2: /api/histories/33b43b4e7093c91f
  name: output test
  id: 33b43b4e7093c91f
```

The result is a Collection of the histories of the user specified by the API key (you). To look at the details of a particular history, say #1 above, do the following:

```
% ./display.py my_key http://localhost:4096/api/histories/8c49be448cfe29bc
Member Information
------------------
state_details: {'ok': 1, 'failed_metadata': 0, 'upload': 0, 'discarded': 0, 'running': 0, 'setting_me
```

```
state: ok
contents_url: /api/histories/8c49be448cfe29bc/contents
id: 8c49be448cfe29bc
name: Unnamed history
```

This gives detailed information about the specific member in question, in this case the History. To view history contents, do the following:

```
% ./display.py my_key http://localhost:4096/api/histories/8c49be448cfe29bc/contents
Collection Members
------------------
#1: /api/histories/8c49be448cfe29bc/contents/6f91353f3eb0fa4a
  name: Pasted Entry
  type: file
  id: 6f91353f3eb0fa4a
```

What we have here is another Collection of items containing all of the datasets in this particular history. Finally, to view details of a particular dataset in this collection, execute the following:

```
% ./display.py my_key http://localhost:4096/api/histories/8c49be448cfe29bc/contents/6f91353f3eb0fa4a
Member Information
------------------
misc_blurb: 1 line
name: Pasted Entry
data_type: txt
deleted: False
file_name: /Users/yoplait/work/galaxy-stock/database/files/000/dataset_82.dat
state: ok
download_url: /datasets/6f91353f3eb0fa4a/display?to_ext=txt
visible: True
genome_build: ?
model_class: HistoryDatasetAssociation
file_size: 17
metadata_data_lines: 1
id: 6f91353f3eb0fa4a
misc_info: uploaded txt file
metadata_dbkey: ?
```

And now you've successfully used the API to request and select a history, browse the contents of that history, and then look at detailed information about a particular dataset.

For a more comprehensive Data Library example, set the following option in your galaxy.ini as well, and restart galaxy again:

```
admin_users = you@example.org
library_import_dir = /path/to/some/directory
```

In the directory you specified for 'library_import_dir', create some subdirectories, and put (or symlink) files to import into Galaxy into those subdirectories.

In Galaxy, create an account that matches the address you put in 'admin_users', then browse to that user's preferences and generate a new API Key. Copy the key to your clipboard and then use these scripts:

```
% ./display.py my_key http://localhost:4096/api/libraries
Collection Members
------------------

0 elements in collection

% ./library_create_library.py my_key http://localhost:4096/api/libraries api_test 'API Test Library'
```

```
Response
--------
/api/libraries/f3f73e481f432006
  name: api_test
  id: f3f73e481f432006

% ./display.py my_key http://localhost:4096/api/libraries
Collection Members
------------------
/api/libraries/f3f73e481f432006
  name: api_test
  id: f3f73e481f432006

% ./display.py my_key http://localhost:4096/api/libraries/f3f73e481f432006
Member Information
------------------
synopsis: None
contents_url: /api/libraries/f3f73e481f432006/contents
description: API Test Library
name: api_test

% ./display.py my_key http://localhost:4096/api/libraries/f3f73e481f432006/contents
Collection Members
------------------
/api/libraries/f3f73e481f432006/contents/28202595c0d2591f61ddda595d2c3670
  name: /
  type: folder
  id: 28202595c0d2591f61ddda595d2c3670

% ./library_create_folder.py my_key http://localhost:4096/api/libraries/f3f73e481f432006/contents 282
Response
--------
/api/libraries/f3f73e481f432006/contents/28202595c0d2591fa4f9089d2303fd89
  name: api_test_folder1
  id: 28202595c0d2591fa4f9089d2303fd89

% ./library_upload_from_import_dir.py my_key http://localhost:4096/api/libraries/f3f73e481f432006/con
Response
--------
/api/libraries/f3f73e481f432006/contents/e9ef7fdb2db87d7b
  name: 2.bed
  id: e9ef7fdb2db87d7b
/api/libraries/f3f73e481f432006/contents/3b7f6a31f80a5018
  name: 3.bed
  id: 3b7f6a31f80a5018

% ./display.py my_key http://localhost:4096/api/libraries/f3f73e481f432006/contents
Collection Members
------------------
/api/libraries/f3f73e481f432006/contents/28202595c0d2591f61ddda595d2c3670
  name: /
  type: folder
  id: 28202595c0d2591f61ddda595d2c3670
/api/libraries/f3f73e481f432006/contents/28202595c0d2591fa4f9089d2303fd89
  name: /api_test_folder1
  type: folder
  id: 28202595c0d2591fa4f9089d2303fd89
/api/libraries/f3f73e481f432006/contents/e9ef7fdb2db87d7b
```

```
  name: /api_test_folder1/2.bed
  type: file
  id: e9ef7fdb2db87d7b
/api/libraries/f3f73e481f432006/contents/3b7f6a31f80a5018
  name: /api_test_folder1/3.bed
  type: file
  id: 3b7f6a31f80a5018

% ./display.py my_key http://localhost:4096/api/libraries/f3f73e481f432006/contents/e9ef7fdb2db87d7b
Member Information
------------------
misc_blurb: 68 regions
metadata_endCol: 3
data_type: bed
metadata_columns: 6
metadata_nameCol: 4
uploaded_by: nate@...
metadata_strandCol: 6
name: 2.bed
genome_build: hg19
metadata_comment_lines: None
metadata_startCol: 2
metadata_chromCol: 1
file_size: 4272
metadata_data_lines: 68
message:
metadata_dbkey: hg19
misc_info: uploaded bed file
date_uploaded: 2010-06-22T17:01:51.266119
metadata_column_types: str, int, int, str, int, str
```

Other parameters are valid when uploading, they are the same parameters as are used in the web form, like 'link_data_only' and etc.

The request and response format should be considered alpha and are subject to change.

1.1.3 API Design Guidelines

The following section outlines guidelines related to extending and/or modifing the Galaxy API. The Galaxy API has grown in an ad-hoc fashion over time by many contributors and so clients SHOULD NOT expect the API will conform to these guidelines - but developers contributing to the Galaxy API SHOULD follow these guidelines.

- API functionality should include docstring documentation for consumption by readthedocs.org.
- Developers should familiarize themselves with the HTTP status code definitions http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html. The API responses should properly set the status code according to the result - in particular 2XX responses should be used for successful requests, 4XX for various kinds of client errors, and 5XX for the errors on the server side.
- If there is an error processing some part of request (one item in a list for instance), the status code should be set to reflect the error and the partial result may or may not be returned depending on the controller - this behavior should be documented.
- API methods should throw a finite number of exceptions (defined in exceptions Package) and these should subclass *MessageException* and not paste/wsgi HTTP exceptions. When possible, the framework itself should be responsible catching these exceptions, setting the status code, and building an error response.

- Error responses should not consist of plain text strings - they should be dictionaries describing the error and containing the following:

```
{
  "status_code": 400,
  "err_code": 400007,
  "err_msg": "Request contained invalid parameter, action could not be completed.",
  "type": "error",
  "extra_error_info": "Extra information."
}
```

  Various error conditions (once a format has been chosen and framework to enforce it in place) should be spelled out in this document.

- Backward compatibility is important and should be maintained when possible. If changing behavior in a non-backward compatibile way please ensure one of the following holds - there is a strong reason to believe no consumers depend on a behavior, the behavior is effectively broken, or the API method being modified has not been part of a tagged dist release.

The following bullet points represent good practices more than guidelines, please consider them when modifying the API.

- Functionality should not be copied and pasted between controllers - consider refactoring functionality into associated classes or short of that into Mixins (http://en.wikipedia.org/wiki/Composition_over_inheritance) or into Managers (managers Package).
- API additions are more permanent changes to Galaxy than many other potential changes and so a second opinion on API changes should be sought. (Consider a pull request!)
- New API functionality should include functional tests. These functional tests should be implemented in Python and placed in *test/functional/api*. (Once such a framework is in place - it is not right now).
- Changes to reflect modifications to the API should be pushed upstream to the BioBlend project if possible.

Longer term goals/notes.

- It would be advantageous to have a clearer separation of anonymous and admin handling functionality.
- If at some point in the future, functionality needs to be added that breaks backward compatibility in a significant way to a component used by the community - a "dev" variant of the API will be established and the community should be alerted and given a timeframe for when the old behavior will be replaced with the new behavior.
- Consistent standards for range-based requests, batch requests, filtered requests, etc... should be established and documented here.

1.1.4 API Controllers

Galaxy offers the following API controllers:

`annotations` Module

API operations on annotations.

class `galaxy.webapps.galaxy.api.annotations.`**`BaseAnnotationsController`**(*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesStored*
galaxy.model.item_attrs.UsesAnnotations

`create`(*trans*, **args*, ***kwargs*)

`delete`(*trans*, **args*, ***kwargs*)

index (*trans*, **args*, ***kwargs*)

undelete (*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.api.annotations.**HistoryAnnotationsController** (*app*)
Bases: *galaxy.webapps.galaxy.api.annotations.BaseAnnotationsController*

controller_name = 'history_annotations'

tagged_item_id = 'history_id'

class galaxy.webapps.galaxy.api.annotations.**HistoryContentAnnotationsController** (*app*)
Bases: *galaxy.webapps.galaxy.api.annotations.BaseAnnotationsController*

controller_name = 'history_content_annotations'

tagged_item_id = 'history_content_id'

class galaxy.webapps.galaxy.api.annotations.**WorkflowAnnotationsController** (*app*)
Bases: *galaxy.webapps.galaxy.api.annotations.BaseAnnotationsController*

controller_name = 'workflow_annotations'

tagged_item_id = 'workflow_id'

authenticate Module

API key retrieval through BaseAuth Sample usage:

curl –user zipzap@foo.com:password http://localhost:8080/api/authenticate/baseauth

Returns:

{ "api_key": "baa4d6e3a156d3033f05736255f195f9"

}

class galaxy.webapps.galaxy.api.authenticate.**AuthenticationController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

get_api_key (*trans*, **args*, ***kwargs*)
def get_api_key(self, trans, **kwd) * GET /api/authenticate/baseauth

returns an API key for authenticated user based on BaseAuth headers

Returns api_key in json format

Return type dict

Raises ObjectNotFound, HTTPBadRequest

configuration Module

API operations allowing clients to determine Galaxy instance's capabilities and configuration settings.

class galaxy.webapps.galaxy.api.configuration.**ConfigurationController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

dynamic_tool_confs (*trans*, **args*, ***kwargs*)

get_config_dict (*trans*, *return_admin=False*, *view=None*, *keys=None*, *default_view='all'*)
Return a dictionary with (a subset of) current Galaxy settings.

If *return_admin* also include a subset of more sensitive keys. Pass in *view* (String) and comma seperated list of keys to control which configuration settings are returned.

index (*trans*, **args*, ***kwargs*)
GET /api/configuration Return an object containing exposable configuration settings.

Note: a more complete list is returned if the user is an admin.

tool_lineages (*trans*, **args*, ***kwargs*)

version (*trans*, **args*, ***kwargs*)
GET /api/version Return a description of the major version of Galaxy (e.g. 15.03).

Return type dict

Returns dictionary with major version keyed on 'version_major'

dataset_collections Module

class galaxy.webapps.galaxy.api.dataset_collections.**DatasetCollectionsController** (*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesLibra

create (*trans*, **args*, ***kwargs*)

- **POST /api/dataset_collections:** create a new dataset collection instance.

Parameters **payload** (*dict*) – (optional) dictionary structure containing: * collection_type: dataset colltion type to create. * instance_type: Instance type - 'history' or 'library'. * name: the new dataset collections's name * datasets: object describing datasets for collection

Return type dict

Returns element view of new dataset collection

index (*trans*, **args*, ***kwargs*)

show (*trans*, **args*, ***kwargs*)

datasets Module

datatypes Module

API operations allowing clients to determine datatype supported by Galaxy.

class galaxy.webapps.galaxy.api.datatypes.**DatatypesController** (*app*)
Bases: galaxy.web.base.controller.BaseAPIController

converters (*trans*, **args*, ***kwargs*)

index (*trans*, **args*, ***kwargs*)
GET /api/datatypes Return an object containing upload datatypes.

mapping (*trans*, **args*, ***kwargs*)
GET /api/datatypes/mapping Return a dictionary of class to class mappings.

sniffers (*trans*, **args*, ***kwargs*)
GET /api/datatypes/sniffers Return a list of sniffers.

extended_metadata Module

API operations on annotations.

class galaxy.webapps.galaxy.api.extended_metadata.**BaseExtendedMetadataController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesExtend
galaxy.web.base.controller.UsesLibraryMixinItems, galaxy.web.base.controller.UsesStoredW

create(*trans*, **args*, ***kwargs*)

delete(*trans*, **args*, ***kwargs*)

index(*trans*, **args*, ***kwargs*)

undelete(*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.api.extended_metadata.**HistoryDatasetExtendMetadataController**(*app*)
Bases: galaxy.webapps.galaxy.api.extended_metadata.BaseExtendedMetadataController

controller_name = 'history_dataset_extended_metadata'

exmeta_item_id = 'history_content_id'

class galaxy.webapps.galaxy.api.extended_metadata.**LibraryDatasetExtendMetadataController**(*app*)
Bases: galaxy.webapps.galaxy.api.extended_metadata.BaseExtendedMetadataController

controller_name = 'library_dataset_extended_metadata'

exmeta_item_id = 'library_content_id'

folder_contents Module

API operations on the contents of a library folder.

class galaxy.webapps.galaxy.api.folder_contents.**FolderContentsController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesLibrar
galaxy.web.base.controller.UsesLibraryMixinItems

Class controls retrieval, creation and updating of folder contents.

build_path(*trans*, *folder*)
Search the path upwards recursively and load the whole route of names and ids for breadcrumb building purposes.

Parameters

- **folder** – current folder for navigating up
- **type** – Galaxy LibraryFolder

Returns list consisting of full path to the library

Type list

create(*self*, *trans*, *library_id*, *payload*, ***kwd*)

•**POST /api/folders/{encoded_id}/contents** create a new library file from an HDA

Parameters **payload** – dictionary structure containing:

Returns a dictionary containing the id, name, and 'show' url of the new item

Return type dict

Raises ObjectAttributeInvalidException, InsufficientPermissionsException, ItemAccessibilityException, InternalServerError

index (*trans*, **args*, ***kwargs*)
GET /api/folders/{encoded_folder_id}/contents

Displays a collection (list) of a folder's contents (files and folders). Encoded folder ID is prepended with 'F' if it is a folder as opposed to a data set which does not have it. Full path is provided in response as a separate object providing data for breadcrumb path building.

Parameters

- **folder_id** (*encoded string*) – encoded ID of the folder which contents should be library_dataset_dict
- **kwd** (*dict*) – keyword dictionary with other params

Returns dictionary containing all items and metadata

Type dict

Raises MalformedId, InconsistentDatabase, ObjectNotFound, InternalServerError

show (*trans*, **args*, ***kwargs*)
GET /api/folders/{encoded_folder_id}/

update (*trans*, **args*, ***kwargs*)
PUT /api/folders/{encoded_folder_id}/contents

folders Module

API operations on library folders.

class galaxy.webapps.galaxy.api.folders.**FoldersController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesLibrar*
galaxy.web.base.controller.UsesLibraryMixinItems

create (*self*, *trans*, *encoded_parent_folder_id*, ***kwd*)
*POST /api/folders/{encoded_parent_folder_id}

Create a new folder object underneath the one specified in the parameters.

Parameters

- **encoded_parent_folder_id** (*an encoded id string (should be prefixed by 'F')*) – the parent folder's id (required)
- **name** (*str*) – the name of the new folder (required)
- **description** (*str*) – the description of the new folder

Returns information about newly created folder, notably including ID

Return type dictionary

Raises RequestParameterMissingException

delete (*self*, *trans*, *id*, ***kwd*)

- **DELETE /api/folders/{id}** marks the folder with the given `id` as *deleted* (or removes the *deleted* mark if the *undelete* param is true)

Note: Currently, only admin users can un/delete folders.

Parameters

- **id** (*an encoded id string*) – the encoded id of the folder to un/delete
- **undelete** (bool) – (optional) flag specifying whether the item should be deleted or undeleted, defaults to false:

Returns detailed folder information

Return type dictionary

Raises ItemAccessibilityException, MalformedId, ObjectNotFound

get_permissions (*trans*, **args*, ***kwargs*)

•GET /api/folders/{id}/permissions

Load all permissions for the given folder id and return it.

Parameters

- **encoded_folder_id** (*an encoded id string*) – the encoded id of the folder
- **scope** (*string*) – either 'current' or 'available'

Returns dictionary with all applicable permissions' values

Return type dictionary

Raises ObjectNotFound, InsufficientPermissionsException

index (*trans*, **args*, ***kwargs*)

*GET /api/folders/ This would normally display a list of folders. However, that would be across multiple libraries, so it's not implemented.

set_permissions (*trans*, **args*, ***kwargs*)

def set_permissions(self, trans, encoded_folder_id, **kwd): *POST /api/folders/{encoded_folder_id}/permissions

Parameters

- **encoded_folder_id** (*an encoded id string*) – the encoded id of the folder to set the permissions of
- **action** (*string*) – (required) describes what action should be performed available actions: set_permissions
- **add_ids[]** (*string or list*) – list of Role.id defining roles that should have add item permission on the folder
- **manage_ids[]** (*string or list*) – list of Role.id defining roles that should have manage permission on the folder
- **modify_ids[]** (*string or list*) – list of Role.id defining roles that should have modify permission on the folder

Return type dictionary

Returns dict of current roles for all available permission types.

Raises RequestParameterInvalidException, ObjectNotFound, InsufficientPermissionsException, InternalServerError RequestParameterMissingException

show (*self*, *trans*, *id*, ***kwd*)
*GET /api/folders/{encoded_folder_id}

Displays information about a folder.

Parameters **id** (*an encoded id string (has to be prefixed by 'F')*) – the folder's encoded id (required)

Returns dictionary including details of the folder

Return type dict

update (*trans*, **args*, ***kwargs*)
PUT /api/folders/{encoded_folder_id}

forms Module

API operations on FormDefinition objects.

class galaxy.webapps.galaxy.api.forms.**FormDefinitionAPIController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

create (*trans*, **args*, ***kwargs*)
POST /api/forms Creates a new form.

index (*trans*, **args*, ***kwargs*)
GET /api/forms Displays a collection (list) of forms.

show (*trans*, **args*, ***kwargs*)
GET /api/forms/{encoded_form_id} Displays information about a form.

ftp_files Module

genomes Module

class galaxy.webapps.galaxy.api.genomes.**GenomesController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

RESTful controller for interactions with genome data.

index (*trans*, **args*, ***kwargs*)
GET /api/genomes: returns a list of installed genomes

show (*trans*, **args*, ***kwargs*)
GET /api/genomes/{id}

Returns information about build <id>

galaxy.webapps.galaxy.api.genomes.**get_id** (*base*, *format*)

group_roles Module

API operations on Group objects.

class galaxy.webapps.galaxy.api.group_roles.**GroupRolesAPIController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

delete (*trans*, **args*, ***kwargs*)
DELETE /api/groups/{encoded_group_id}/roles/{encoded_role_id} Removes a role from a group

index (*trans*, **args*, ***kwargs*)
GET /api/groups/{encoded_group_id}/roles Displays a collection (list) of groups.

show (*trans*, **args*, ***kwargs*)
GET /api/groups/{encoded_group_id}/roles/{encoded_role_id} Displays information about a group role.

update (*trans*, **args*, ***kwargs*)
PUT /api/groups/{encoded_group_id}/roles/{encoded_role_id} Adds a role to a group

group_users Module

API operations on Group objects.

class galaxy.webapps.galaxy.api.group_users.**GroupUsersAPIController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

delete (*trans*, **args*, ***kwargs*)
DELETE /api/groups/{encoded_group_id}/users/{encoded_user_id} Removes a user from a group

index (*trans*, **args*, ***kwargs*)
GET /api/groups/{encoded_group_id}/users Displays a collection (list) of groups.

show (*trans*, **args*, ***kwargs*)
GET /api/groups/{encoded_group_id}/users/{encoded_user_id} Displays information about a group user.

update (*trans*, **args*, ***kwargs*)
PUT /api/groups/{encoded_group_id}/users/{encoded_user_id} Adds a user to a group

groups Module

API operations on Group objects.

class galaxy.webapps.galaxy.api.groups.**GroupAPIController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

create (*trans*, **args*, ***kwargs*)
POST /api/groups Creates a new group.

index (*trans*, **args*, ***kwargs*)
GET /api/groups Displays a collection (list) of groups.

show (*trans*, **args*, ***kwargs*)
GET /api/groups/{encoded_group_id} Displays information about a group.

update (*trans*, **args*, ***kwargs*)
PUT /api/groups/{encoded_group_id} Modifies a group.

histories Module

API operations on a history.

See also:

galaxy.model.History

class galaxy.webapps.galaxy.api.histories.**HistoriesController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.ExportsHis*
galaxy.web.base.controller.ImportsHistoryMixin

archive_download (*trans*, **args*, ***kwargs*)

export_download(self, trans, id, jeha_id) * GET /api/histories/{id}/exports/{jeha_id}:

If ready and available, return raw contents of exported history. Use/poll "PUT /api/histories/{id}/exports" to initiate the creation of such an export - when ready that route will return 200 status code (instead of 202) with a JSON dictionary containing a *download_url*.

archive_export (*trans*, **args*, ***kwargs*)

export_archive(self, trans, id, payload) * PUT /api/histories/{id}/exports:

start job (if needed) to create history export for corresponding history.

Parameters **id** (*str*) – the encoded id of the history to export

Return type dict

Returns object containing url to fetch export from.

citations (*trans*, **args*, ***kwargs*)

create (*trans*, *payload*)

•**POST /api/histories:** create a new history

Parameters

- **payload** (*dict*) – (optional) dictionary structure containing: * name: the new history's name * history_id: the id of the history to copy * archive_source: the url that will generate the archive to import * archive_type: 'url' (default)
- **keys** – same as the use of *keys* in the *index* function above
- **view** – same as the use of *view* in the *index* function above

Return type dict

Returns element view of new history

delete (*self*, *trans*, *id*, ***kwd*)

•**DELETE /api/histories/{id}** delete the history with the given `id`

Note: Stops all active jobs in the history if purge is set.

Parameters

- **id** (*str*) – the encoded id of the history to delete
- **kwd** (*dict*) – (optional) dictionary structure containing extra parameters

You can purge a history, removing all it's datasets from disk (if unshared), by passing in `purge=True` in the url.

Parameters

- **keys** – same as the use of *keys* in the *index* function above
- **view** – same as the use of *view* in the *index* function above

Return type dict

Returns the deleted or purged history

index (*trans*, *deleted='False'*)

- **GET /api/histories:** return undeleted histories for the current user
- **GET /api/histories/deleted:** return deleted histories for the current user

Note: Anonymous users are allowed to get their current history

Parameters **deleted** (*boolean*) – if True, show only deleted histories, if False, non-deleted

Return type *list*

Returns list of dictionaries containing summary history information

The following are optional parameters:

view: string, one of ('summary','detailed'), defaults to 'summary' controls which set of properties to return

keys: comma separated strings, unused by default keys/names of individual properties to return

If neither keys or views are sent, the default view (set of keys) is returned. If both a view and keys are sent, the key list and the view's keys are combined. If keys are send and no view, only those properties in keys are returned.

For which properties are available see: galaxy/managers/histories/HistorySerializer

The list returned can be filtered by using two optional parameters:

q: string, generally a property name to filter by followed by an (often optional) hyphen and operator string.

qv: string, the value to filter by

..example: To filter the list to only those created after 2015-01-29, the query string would look like:

'?q=create_time-gt&qv=2015-01-29'

Multiple filters can be sent in using multiple q/qv pairs: '?q=create_time-gt&qv=2015-01-29&q=tag-has&qv=experiment-1'

The list returned can be paginated using two optional parameters:

limit: integer, defaults to no value and no limit (return all) how many items to return

offset: integer, defaults to 0 and starts at the beginning skip the first (offset - 1) items and begin returning at the Nth item

..example:

limit and offset can be combined. Skip the first two and return five: '?limit=5&offset=3'

show (*trans*, *id*, *deleted='False'*)

- **GET /api/histories/{id}:** return the history with `id`
- **GET /api/histories/deleted/{id}:** return the deleted history with `id`
- **GET /api/histories/most_recently_used:** return the most recently used history

Parameters

- **id** (*an encoded id string*) – the encoded id of the history to query or the string 'most_recently_used'
- **deleted** (*boolean*) – if True, allow information on a deleted history to be shown.

- **keys** – same as the use of *keys* in the *index* function above
- **view** – same as the use of *view* in the *index* function above

Return type dictionary

Returns detailed history information

undelete (*self*, *trans*, *id*, ***kwd*)

•**POST /api/histories/deleted/{id}/undelete:** undelete history (that hasn't been purged) with the given `id`

Parameters

- **id** (*str*) – the encoded id of the history to undelete
- **keys** – same as the use of *keys* in the *index* function above
- **view** – same as the use of *view* in the *index* function above

Return type str

Returns 'OK' if the history was undeleted

update (*self*, *trans*, *id*, *payload*, ***kwd*)

•**PUT /api/histories/{id}** updates the values for the history with the given `id`

Parameters

- **id** (*str*) – the encoded id of the history to update
- **payload** (*dict*) – a dictionary containing any or all the fields in `galaxy.model.History.to_dict()` and/or the following:
  - annotation: an annotation for the history
- **keys** – same as the use of *keys* in the *index* function above
- **view** – same as the use of *view* in the *index* function above

Return type dict

Returns an error object if an error occurred or a dictionary containing any values that were different from the original and, therefore, updated

`history_contents` Module

API operations on the contents of a history.

class `galaxy.webapps.galaxy.api.history_contents.`**HistoryContentsController** (*app*)

Bases: `galaxy.web.base.controller.BaseAPIController`, `galaxy.web.base.controller.UsesLibrar`
`galaxy.web.base.controller.UsesLibraryMixinItems`, `galaxy.web.base.controller.UsesTagsMi`

create (*self*, *trans*, *history_id*, *payload*, ***kwd*)

•**POST /api/histories/{history_id}/contents/{type}** create a new HDA by copying an accessible LibraryDataset

Parameters

- **history_id** (*str*) – encoded id string of the new HDA's History

- **type** (*str*) – Type of history content - 'dataset' (default) or 'dataset_collection'.
- **payload** (*dict*) – dictionary structure containing:: copy from library (for type 'dataset'): 'source' = 'library' 'content' = [the encoded id from the library dataset]

  copy from history dataset (for type 'dataset'): 'source' = 'hda' 'content' = [the encoded id from the HDA]

  copy from history dataset collection (for type 'dataset_collection') 'source' = 'hdca' 'content' = [the encoded id from the HDCA]

  create new history dataset collection (for type 'dataset_collection') 'source' = 'new_collection' (default 'source' if type is

  'dataset_collection' - no need to specify this)

  'collection_type' = For example, "list", "paired", "list:paired". 'name' = Name of new dataset collection. 'element_identifiers' = Recursive list structure defining collection.

  Each element must have 'src' which can be 'hda', 'ldda', 'hdca', or 'new_collection', as well as a 'name' which is the name of element (e.g. "forward" or "reverse" for paired datasets, or arbitrary sample names for instance for lists). For all src's except 'new_collection' - a encoded 'id' attribute must be included wiht element as well. 'new_collection' sources must defined a 'collection_type' and their own list of (potentially) nested 'element_identifiers'.

..note: Currently, a user can only copy an HDA from a history that the user owns.

Return type dict

Returns dictionary containing detailed information for the new HDA

delete (*self*, *trans*, *history_id*, *id*, ***kwd*)

•**DELETE /api/histories/{history_id}/contents/{id}** delete the HDA with the given `id`

Note: Currently does not stop any active jobs for which this dataset is an output.

Parameters

- **id** (*str*) – the encoded id of the history to delete
- **purge** (bool) – if True, purge the HDA
- **kwd** (*dict*) – (optional) dictionary structure containing:
  - **payload: a dictionary itself containing:**
    - purge: if True, purge the HDA

Note: that payload optionally can be placed in the query string of the request. This allows clients that strip the request body to still purge the dataset.

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

index (*self*, *trans*, *history_id*, *ids=None*, ***kwd*)

•**GET /api/histories/{history_id}/contents** return a list of HDA data for the history with the given `id`

Note: Anonymous users are allowed to get their current history contents

If Ids is not given, index returns a list of *summary* objects for every HDA associated with the given *history_id*.

If ids is given, index returns a *more complete* json object for each HDA in the ids list.

Parameters

- **`history_id`** (*str*) – encoded id string of the HDA's History
- **`ids`** (*str*) – (optional) a comma separated list of encoded *HDA* ids
- **`types`** (*str*) – (optional) kinds of contents to index (currently just dataset, but dataset_collection will be added shortly).

Return type *list*

Returns dictionaries containing summary or detailed HDA information

`show` (*self*, *trans*, *id*, *history_id*, ***kwd*)

•**GET /api/histories/{history_id}/contents/{id}** return detailed information about an HDA within a history

Note: Anonymous users are allowed to get their current history contents

Parameters

- **`ids`** – the encoded id of the HDA to return
- **`history_id`** (*str*) – encoded id string of the HDA's History

Return type dict

Returns dictionary containing detailed HDA information

`update` (*self*, *trans*, *history_id*, *id*, *payload*, ***kwd*)

•**PUT /api/histories/{history_id}/contents/{id}** updates the values for the HDA with the given `id`

Parameters

- **`history_id`** (*str*) – encoded id string of the HDA's History
- **`id`** (*str*) – the encoded id of the history to undelete
- **`payload`** (*dict*) – a dictionary containing any or all the fields in `galaxy.model.HistoryDatasetAssociation.to_dict()` and/or the following:
  - annotation: an annotation for the HDA

Return type dict

Returns an error object if an error occurred or a dictionary containing any values that were different from the original and, therefore, updated

item_tags Module

API operations related to tagging items.

class galaxy.webapps.galaxy.api.item_tags.**BaseItemTagsController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesTagsMi

create(*trans*, **args*, ***kwargs*)

delete(*trans*, **args*, ***kwargs*)

index(*trans*, **args*, ***kwargs*)

show(*trans*, **args*, ***kwargs*)

update(*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.api.item_tags.**HistoryContentTagsController**(*app*)
Bases: galaxy.webapps.galaxy.api.item_tags.BaseItemTagsController

controller_name = 'history_content_tags'

tagged_item_class = 'HistoryDatasetAssociation'

tagged_item_id = 'history_content_id'

class galaxy.webapps.galaxy.api.item_tags.**HistoryTagsController**(*app*)
Bases: galaxy.webapps.galaxy.api.item_tags.BaseItemTagsController

controller_name = 'history_tags'

tagged_item_class = 'History'

tagged_item_id = 'history_id'

class galaxy.webapps.galaxy.api.item_tags.**WorkflowTagsController**(*app*)
Bases: galaxy.webapps.galaxy.api.item_tags.BaseItemTagsController

controller_name = 'workflow_tags'

tagged_item_class = 'StoredWorkflow'

tagged_item_id = 'workflow_id'

job_files Module

API for asynchronous job running mechanisms can use to fetch or put files related to running and queued jobs.

class galaxy.webapps.galaxy.api.job_files.**JobFilesAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

This job files controller allows remote job running mechanisms to read and modify the current state of files for queued and running jobs. It is certainly not meant to represent part of Galaxy's stable, user facing API.

Furthermore, even if a user key corresponds to the user running the job, it should not be accepted for authorization - this API allows access to low-level unfiltered files and such authorization would break Galaxy's security model for tool execution.

create(*self*, *trans*, *job_id*, *payload*, ***kwargs*)

- **POST /api/jobs/{job_id}/files** Populate an output file (formal dataset, task split part, working directory file (such as those related to metadata)). This should be a multipart post with a 'file' parameter containing the contents of the actual file to create.

Parameters

- **job_id** (*str*) – encoded id string of the job
- **payload** (*dict*) – dictionary structure containing:: 'job_key' = Key authenticating 'path' = Path to file to create.

..note: This API method is intended only for consumption by job runners, not end users.

Return type dict

Returns an okay message

index (*self*, *trans*, *job_id*, ***kwargs*)

•**GET /api/jobs/{job_id}/files** Get a file required to staging a job (proper datasets, extra inputs, task-split inputs, working directory files).

Parameters

- **job_id** (*str*) – encoded id string of the job
- **path** (*str*) – Path to file.
- **job_key** (*str*) – A key used to authenticate this request as acting on behalf or a job runner for the specified job.

..note: This API method is intended only for consumption by job runners, not end users.

Return type *binary*

Returns contents of file

jobs Module

API operations on a jobs.

See also:

galaxy.model.Jobs

class galaxy.webapps.galaxy.api.jobs.**JobController** (*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesLibrar*

create (*trans*, **args*, ***kwargs*)

See the create method in tools.py in order to submit a job.

index (*trans*, *state=None*, *tool_id=None*, *history_id=None*, *date_range_min=None*, *date_range_max=None*, *user_details=False*)

•**GET /api/jobs:** return jobs for current user

!! if user is admin and user_details is True, then return jobs for all galaxy users based on filtering - this is an extended service

Parameters **state** (*string or list*) – limit listing of jobs to those that match one of the included states. If none, all are returned.

Valid Galaxy job states include: 'new', 'upload', 'waiting', 'queued', 'running', 'ok', 'error', 'paused', 'deleted', 'deleted_new'

Parameters

- **tool_id** (*string or list*) – limit listing of jobs to those that match one of the included tool_ids. If none, all are returned.
- **user_details** (*boolean*) – if true, and requestor is an admin, will return external job id and user email.
- **date_range_min** (*string '2014-01-01'*) – limit the listing of jobs to those updated on or after requested date
- **date_range_max** (*string '2014-12-31'*) – limit the listing of jobs to those updated on or before requested date
- **history_id** (*string*) – limit listing of jobs to those that match the history_id. If none, all are returned.

Return type *list*

Returns list of dictionaries containing summary job information

inputs (*trans*, **args*, ***kwargs*)

show(trans, id) * GET /api/jobs/{job_id}/inputs

returns input datasets created by job

Parameters **id** (*string*) – Encoded job id

Return type dictionary

Returns dictionary containing input dataset associations

outputs (*trans*, **args*, ***kwargs*)

show(trans, id) * GET /api/jobs/{job_id}/outputs

returns output datasets created by job

Parameters **id** (*string*) – Encoded job id

Return type dictionary

Returns dictionary containing output dataset associations

search (*trans*, *payload*)

•**POST /api/jobs/search:** return jobs for current user

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

Return type *list*

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

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

show (*trans*, *id*)

•**GET /api/jobs/{job_id}:** return jobs for current user

Parameters

- **id** (*string*) – Specific job id
- **full** (*boolean*) – whether to return extra information

Return type dictionary

Returns dictionary containing full description of job data

lda_datasets Module

API operations on the library datasets.

class galaxy.webapps.galaxy.api.lda_datasets.**LibraryDatasetsController**(*app*)

Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesVisual

delete(*trans*, **args*, ***kwargs*)

delete(self, trans, encoded_dataset_id, **kwd): * DELETE /api/libraries/datasets/{encoded_dataset_id}

Marks the dataset deleted or undeleted based on the value of the undelete flag. If the flag is not present it is considered False and the item is marked deleted.

Parameters **encoded_dataset_id** (*an encoded id string*) – the encoded id of the dataset to change

Returns dict containing information about the dataset

Return type dictionary

download(*self*, *trans*, *format*, ***kwd*)

•GET /api/libraries/datasets/download/{format}

•**POST /api/libraries/datasets/download/{format}** Downloads requested datasets (identified by encoded IDs) in requested format.

example: `GET localhost:8080/api/libraries/datasets/download/tbz?ld_ids%255B%255D=a0d8`

Note: supported format values are: 'zip', 'tgz', 'tbz', 'uncompressed'

Parameters

- **format** (*string*) – string representing requested archive format
- **ld_ids[]** (*an array*) – an array of encoded ids

Return type file

Returns either archive with the requested datasets packed inside or a single uncompressed dataset

Raises MessageException, ItemDeletionException, ItemAccessibilityException, HTTPBadRequest, OSError, IOError, ObjectNotFound

load(*trans*, **args*, ***kwargs*)

load(self, trans, **kwd): * POST /api/libraries/datasets Load dataset from the given source into the library. Source can be:

user directory - root folder specified in galaxy.ini as "$user_library_import_dir"
example path: path/to/galaxy/$user_library_import_dir/user@example.com/{user can browse everything here} the folder with the user login has to be created beforehand

(admin)import directory - root folder specified in galaxy ini as "$library_import_dir"
example path: path/to/galaxy/$library_import_dir/{admin can browse everything here}

(admin)any absolute or relative path - option allowed with "allow_library_path_paste" in galaxy.ini

Parameters

- **encoded_folder_id** (*an encoded id string*) – the encoded id of the folder to import dataset(s) to
- **source** (*str*) – source the datasets should be loaded form
- **link_data** (bool) – flag whether to link the dataset to data or copy it to Galaxy, defaults to copy while linking is set to True all symlinks will be resolved _once_
- **preserve_dirs** (bool) – flag whether to preserve the directory structure when importing dir if False only datasets will be imported
- **file_type** (*str*) – file type of the loaded datasets, defaults to 'auto' (autodetect)
- **dbkey** (*str*) – dbkey of the loaded genome, defaults to '?' (unknown)

Returns dict containing information about the created upload job

Return type dictionary

show (*self*, *trans*, *id*, ***kwd*)

•**GET /api/libraries/datasets/{encoded_dataset_id}:** Displays information about the dataset identified by the encoded ID.

Parameters **id** (*an encoded id string*) – the encoded id of the dataset to query

Returns detailed dataset information from base controller

Return type dictionary

See also:

galaxy.web.base.controller.UsesLibraryMixinItems.get_library_dataset

show_roles (*trans*, **args*, ***kwargs*)
show_roles(self, trans, id, **kwd): * GET /api/libraries/datasets/{encoded_dataset_id}/permissions

Displays information about current or available roles for a given dataset permission.

Parameters

- **encoded_dataset_id** (*an encoded id string*) – the encoded id of the dataset to query
- **scope** (*string*) – either 'current' or 'available'

Return type dictionary

Returns either dict of current roles for all permission types or dict of available roles to choose from (is the same for any permission type)

show_version (*trans*, **args*, ***kwargs*)

show_version(self, trans, encoded_dataset_id, encoded_ldda_id, **kwd): * GET /api/libraries/datasets/:encoded_dataset_id/versions/:encoded_ldda_id

Displays information about specific version of the library_dataset (i.e. ldda).

Parameters

- **encoded_dataset_id** (*an encoded id string*) – the encoded id of the dataset to query
- **encoded_ldda_id** (*an encoded id string*) – the encoded id of the ldda to query

Return type dictionary

Returns dict of ldda's details

update_permissions (*trans*, **args*, ***kwargs*)

def update(self, trans, encoded_dataset_id, **kwd): *POST /api/libraries/datasets/{encoded_dataset_id}/permissions

Parameters

- **encoded_dataset_id** (*an encoded id string*) – the encoded id of the dataset to update permissions of
- **action** (*string*) – (required) describes what action should be performed available actions: make_private, remove_restrictions, set_permissions
- **access_ids[]** (*string or list*) – list of Role.name defining roles that should have access permission on the dataset
- **manage_ids[]** (*string or list*) – list of Role.name defining roles that should have manage permission on the dataset
- **modify_ids[]** (*string or list*) – list of Role.name defining roles that should have modify permission on the library dataset item

Return type dictionary

Returns dict of current roles for all available permission types

Raises RequestParameterInvalidException, ObjectNotFound, InsufficientPermissionsException, InternalServerError RequestParameterMissingException

libraries Module

API operations on a data library.

class galaxy.webapps.galaxy.api.libraries.**LibrariesController** (*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*

create (*self*, *trans*, *payload*, ***kwd*)

•**POST /api/libraries:** Creates a new library. Only name parameter is required.

Note: Currently, only admin users can create libraries.

Parameters **payload** (*dict*) – dictionary structure containing:: 'name': the new library's name (required) 'description': the new library's description (optional) 'synopsis': the new library's synopsis (optional)

Returns detailed library information

Return type dict

Raises ItemAccessibilityException, RequestParameterMissingException

delete (*self*, *trans*, *id*, ***kwd*)

•**DELETE /api/libraries/{id}** marks the library with the given `id` as *deleted* (or removes the *deleted* mark if the *undelete* param is true)

Note: Currently, only admin users can un/delete libraries.

Parameters

- **id** (*an encoded id string*) – the encoded id of the library to un/delete
- **undelete** (bool) – (optional) flag specifying whether the item should be deleted or undeleted, defaults to false:

Returns detailed library information

Return type dictionary

See also:

galaxy.model.Library.dict_element_visible_keys

Raises ItemAccessibilityException, MalformedId, ObjectNotFound

get_permissions (*trans*, **args*, ***kwargs*)

•GET /api/libraries/{id}/permissions

Load all permissions for the given library id and return it.

Parameters

- **encoded_library_id** (*an encoded id string*) – the encoded id of the library
- **scope** (*string*) – either 'current' or 'available'
- **is_library_access** (bool) – indicates whether the roles available for the library access are requested

Returns dictionary with all applicable permissions' values

Return type dictionary

Raises ObjectNotFound, InsufficientPermissionsException

index (*self*, *trans*, ***kwd*)

•**GET /api/libraries:** Returns a list of summary data for all libraries.

Parameters **deleted** (*boolean (optional)*) – if True, show only `deleted` libraries, if False show only `non-deleted`

Returns list of dictionaries containing library information

Return type *list*

See also:

galaxy.model.Library.dict_collection_visible_keys

set_permissions(*trans*, **args*, ***kwargs*)

**def set_permissions(self, trans, encoded_dataset_id, **kwd): ***POST /api/libraries/{encoded_library_id}/permissions

Parameters

- **encoded_library_id** (*an encoded id string*) – the encoded id of the library to set the permissions of
- **action** (*string*) – (required) describes what action should be performed available actions: remove_restrictions, set_permissions
- **access_ids[]** (*string or list*) – list of Role.id defining roles that should have access permission on the library
- **add_ids[]** (*string or list*) – list of Role.id defining roles that should have add item permission on the library
- **manage_ids[]** (*string or list*) – list of Role.id defining roles that should have manage permission on the library
- **modify_ids[]** (*string or list*) – list of Role.id defining roles that should have modify permission on the library

Return type dictionary

Returns dict of current roles for all available permission types

Raises RequestParameterInvalidException, ObjectNotFound, InsufficientPermissionsException, InternalServerError RequestParameterMissingException

set_permissions_old(*trans*, *library*, *payload*, ***kwd*)

*** old implementation for backward compatibility ***

POST /api/libraries/{encoded_library_id}/permissions Updates the library permissions.

show(*self*, *trans*, *id*, *deleted='False'*, ***kwd*)

•**GET /api/libraries/{encoded_id}:** returns detailed information about a library

•**GET /api/libraries/deleted/{encoded_id}:** returns detailed information about a `deleted` library

Parameters

- **id** (*an encoded id string*) – the encoded id of the library
- **deleted** (*boolean*) – if True, allow information on a `deleted` library

Returns detailed library information

Return type dictionary

See also:

galaxy.model.Library.dict_element_visible_keys

Raises MalformedId, ObjectNotFound

update(*trans*, **args*, ***kwargs*)

•**PATCH /api/libraries/{encoded_id}** Updates the library defined by an `encoded_id` with the data in the payload.

Note: Currently, only admin users can update libraries. Also the library must not be *deleted*.

param id the encoded id of the library

type id an encoded id string

param payload (required) dictionary structure containing:: 'name': new library's name, cannot be empty 'description': new library's description 'synopsis': new library's synopsis

type payload dict

returns detailed library information

rtype dict

raises ItemAccessibilityException, MalformedId, ObjectNotFound, RequestParameterInvalidException, RequestParameterMissingException

`library_contents` Module

API operations on the contents of a data library.

class `galaxy.webapps.galaxy.api.library_contents.`**`LibraryContentsController`**(*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesLibrar*
galaxy.web.base.controller.UsesLibraryMixinItems

`create`(*self*, *trans*, *library_id*, *payload*, ***kwd*)

•**POST /api/libraries/{library_id}/contents:** create a new library file or folder

To copy an HDA into a library send `create_type` of 'file' and the HDA's encoded id in `from_hda_id` (and optionally `ldda_message`).

Parameters

- **`library_id`** (*str*) – the encoded id of the library where to create the new item
- **`payload`** (*dict*) – dictionary structure containing:
  - folder_id: the encoded id of the parent folder of the new item
  - create_type: the type of item to create ('file', 'folder' or 'collection')
  - **from_hda_id: (optional, only if create_type is 'file') the** encoded id of an accessible HDA to copy into the library
  - ldda_message: (optional) the new message attribute of the LDDA created
  - **extended_metadata: (optional) dub-dictionary containing any extended** metadata to associate with the item
  - upload_option: (optional) one of 'upload_file' (default), 'upload_directory' or 'upload_paths'
  - **server_dir: (optional, only if upload_option is** 'upload_directory') relative path of the subdirectory of Galaxy `library_import_dir` to upload. All and only the files (i.e. no subdirectories) contained in the specified directory will be uploaded.

- **filesystem_paths: (optional, only if upload_option is** 'upload_paths' and the user is an admin) file paths on the Galaxy server to upload to the library, one file per line
- **link_data_only: (optional, only when upload_option is** 'upload_directory' or 'upload_paths') either 'copy_files' (default) or 'link_to_files'. Setting to 'link_to_files' symlinks instead of copying the files
- **name: (optional, only if create_type is 'folder') name of the** folder to create
- **description: (optional, only if create_type is 'folder')** description of the folder to create

Return type dict

Returns a dictionary containing the id, name, and 'show' url of the new item

delete (*self*, *trans*, *library_id*, *id*, ***kwd*)

•**DELETE /api/libraries/{library_id}/contents/{id}** delete the LibraryDataset with the given `id`

Parameters

- **id** (*str*) – the encoded id of the library dataset to delete
- **kwd** (*dict*) – (optional) dictionary structure containing:
  - **payload: a dictionary itself containing:**
    - purge: if True, purge the LD

Return type dict

Returns an error object if an error occurred or a dictionary containing: * id: the encoded id of the library dataset, * deleted: if the library dataset was marked as deleted, * purged: if the library dataset was purged

index (*self*, *trans*, *library_id*, ***kwd*)

•**GET /api/libraries/{library_id}/contents:** Returns a list of library files and folders.

Note: May be slow! Returns all content traversing recursively through all folders.

See also:

`galaxy.webapps.galaxy.api.FolderContentsController.index` for a non-recursive solution

Parameters **library_id** (*str*) – the encoded id of the library

Returns

list of dictionaries of the form: * id: the encoded id of the library item * name: the 'library path'

or relationship of the library item to the root

- type: 'file' or 'folder'
- url: the url to get detailed information on the library item

Return type *list*

Raises MalformedId, InconsistentDatabase, RequestParameterInvalidException, InternalServerError

show (*self*, *trans*, *id*, *library_id*, ***kwd*)

•**GET /api/libraries/{library_id}/contents/{id}** Returns information about library file or folder.

Parameters

- **id** (*str*) – the encoded id of the library item to return
- **library_id** (*str*) – the encoded id of the library that contains this item

Returns detailed library item information

Return type dict

See also:

`galaxy.model.LibraryDataset.to_dict()` and `galaxy.model.LibraryFolder.dict_element_vi`

update (*self*, *trans*, *id*, *library_id*, *payload*, ***kwd*)

•**PUT /api/libraries/{library_id}/contents/{id}** create a ImplicitlyConvertedDatasetAssociation

See also:

`galaxy.model.ImplicitlyConvertedDatasetAssociation`

Parameters

- **id** (*str*) – the encoded id of the library item to return
- **library_id** (*str*) – the encoded id of the library that contains this item
- **payload** (*dict*) – dictionary structure containing:: 'converted_dataset_id':

Return type None

Returns None

metrics Module

API operations for for querying and recording user metrics from some client (typically a user's browser).

class galaxy.webapps.galaxy.api.metrics.**MetricsController** (*app*)
Bases: `galaxy.web.base.controller.BaseAPIController`

create (*trans*, *payload*)

•**POST /api/metrics:** record any metrics sent and return some status object

Note: Anonymous users can post metrics

Parameters **payload** (*dict*) – (optional) dictionary structure containing: * metrics: a list containing dictionaries of the form:

** namespace: label indicating the source of the metric ** time: isoformat datetime when the metric was recorded ** level: an integer representing the metric's log level ** args: a json string containing an array of extra data

Return type dict

Returns status object

debugging = None
set to true to send additional debugging info to the log

page_revisions Module

API for updating Galaxy Pages

class galaxy.webapps.galaxy.api.page_revisions.**PageRevisionsController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.SharableIt
galaxy.model.item_attrs.UsesAnnotations, galaxy.web.base.controller.SharableMixin

create(*self*, *trans*, *page_id*, *payload* **kwd*)

•**POST /api/pages/{page_id}/revisions** Create a new revision for a page

Parameters

- **page_id** – Add revision to Page with ID=page_id
- **payload** – A dictionary containing:: 'title' = New title of the page 'content' = New content of the page

Return type dictionary

Returns Dictionary with 'success' or 'error' element to indicate the result of the request

index(*self*, *trans*, *page_id*, **kwd*)

•**GET /api/pages/{page_id}/revisions** return a list of Page revisions

Parameters **page_id** – Display the revisions of Page with ID=page_id

Return type *list*

Returns dictionaries containing different revisions of the page

pages Module

API for updating Galaxy Pages

class galaxy.webapps.galaxy.api.pages.**PagesController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.SharableIt
galaxy.model.item_attrs.UsesAnnotations, galaxy.web.base.controller.SharableMixin

create(*self*, *trans*, *payload*, **kwd*)

•**POST /api/pages** Create a page and return dictionary containing Page summary

Parameters **payload** – dictionary structure containing:: 'slug' = The title slug for the page URL, must be unique 'title' = Title of the page 'content' = HTML contents of the page 'annotation' = Annotation that will be attached to the page

Return type dict

Returns Dictionary return of the Page.to_dict call

delete (*self*, *trans*, *id*, ***kwd*)

- **DELETE /api/pages/{id}** Create a page and return dictionary containing Page summary

Parameters **id** – ID of page to be deleted

Return type dict

Returns Dictionary with 'success' or 'error' element to indicate the result of the request

index (*self*, *trans*, *deleted=False*, ***kwd*)

- **GET /api/pages** return a list of Pages viewable by the user

Parameters **deleted** – Display deleted pages

Return type *list*

Returns dictionaries containing summary or detailed Page information

show (*self*, *trans*, *id*, ***kwd*)

- **GET /api/pages/{id}** View a page summary and the content of the latest revision

Parameters **id** – ID of page to be displayed

Return type dict

Returns Dictionary return of the Page.to_dict call with the 'content' field populated by the most recent revision

provenance Module

API operations provenance

class galaxy.webapps.galaxy.api.provenance.**BaseProvenanceController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

create (*trans*, **args*, ***kwargs*)

delete (*trans*, **args*, ***kwargs*)

index (*trans*, **args*, ***kwargs*)

show (*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.api.provenance.**HDAProvenanceController** (*app*)
Bases: *galaxy.webapps.galaxy.api.provenance.BaseProvenanceController*

controller_name = 'history_content_provenance'

provenance_item_class = 'HistoryDatasetAssociation'

provenance_item_id = 'history_content_id'

class galaxy.webapps.galaxy.api.provenance.**LDDAProvenanceController** (*app*)
Bases: *galaxy.webapps.galaxy.api.provenance.BaseProvenanceController*

controller_name = 'ldda_provenance'

provenance_item_class = 'LibraryDatasetDatasetAssociation'

provenance_item_id = 'library_content_id'

quotas Module

API operations on Quota objects.

class galaxy.webapps.galaxy.api.quotas.**QuotaAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controllers.admin.Adm galaxy.actions.admin.AdminActions, galaxy.web.base.controller.UsesQuotaMixin, galaxy.web.params.QuotaParamParser

create(*trans*, **args*, ***kwargs*)
POST /api/quotas Creates a new quota.

delete(*trans*, **args*, ***kwargs*)
DELETE /api/quotas/{encoded_quota_id} Deletes a quota

index(*trans*, **args*, ***kwargs*)
GET /api/quotas GET /api/quotas/deleted Displays a collection (list) of quotas.

show(*trans*, **args*, ***kwargs*)
GET /api/quotas/{encoded_quota_id} GET /api/quotas/deleted/{encoded_quota_id} Displays information about a quota.

undelete(*trans*, **args*, ***kwargs*)
POST /api/quotas/deleted/{encoded_quota_id}/undelete Undeletes a quota

update(*trans*, **args*, ***kwargs*)
PUT /api/quotas/{encoded_quota_id} Modifies a quota.

request_types Module

API operations on RequestType objects.

class galaxy.webapps.galaxy.api.request_types.**RequestTypeAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

create(*trans*, **args*, ***kwargs*)
POST /api/request_types Creates a new request type (external_service configuration).

index(*trans*, **args*, ***kwargs*)
GET /api/request_types Displays a collection (list) of request_types.

show(*trans*, **args*, ***kwargs*)
GET /api/request_types/{encoded_request_type_id} Displays information about a request_type.

requests Module

API operations on a sample tracking system.

class galaxy.webapps.galaxy.api.requests.**RequestsAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

index(*trans*, **args*, ***kwargs*)
GET /api/requests Displays a collection (list) of sequencing requests.

show(*trans*, **args*, ***kwargs*)
GET /api/requests/{encoded_request_id} Displays details of a sequencing request.

update(*trans*, **args*, ***kwargs*)
PUT /api/requests/{encoded_request_id} Updates a request state, sample state or sample dataset transfer status depending on the update_type

v = ('REQUEST', 'request_state')

roles Module

API operations on Role objects.

class galaxy.webapps.galaxy.api.roles.**RoleAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

create(*trans*, **args*, ***kwargs*)
POST /api/roles Creates a new role.

index(*trans*, **args*, ***kwargs*)
GET /api/roles Displays a collection (list) of roles.

show(*trans*, **args*, ***kwargs*)
GET /api/roles/{encoded_role_id} Displays information about a role.

samples Module

API operations for samples in the Galaxy sample tracking system.

class galaxy.webapps.galaxy.api.samples.**SamplesAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

index(*trans*, **args*, ***kwargs*)
GET /api/requests/{encoded_request_id}/samples Displays a collection (list) of sample of a sequencing request.

k = 'SAMPLE_DATASET'

update(*trans*, **args*, ***kwargs*)
PUT /api/samples/{encoded_sample_id} Updates a sample or objects related (mapped) to a sample.

update_type_values = ['sample_state', 'run_details', 'sample_dataset_transfer_status']

update_types = <galaxy.util.bunch.Bunch object>

v = ['sample_dataset_transfer_status']

search Module

API for searching Galaxy Datasets

class galaxy.webapps.galaxy.api.search.**SearchController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.SharableIt

create(*trans*, **args*, ***kwargs*)
POST /api/search Do a search of the various elements of Galaxy.

tool_data Module

class galaxy.webapps.galaxy.api.tool_data.**ToolData**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

RESTful controller for interactions with tool data

delete (*trans*, **args*, ***kwargs*)
DELETE /api/tool_data/{id} Removes an item from a data table

Parameters

- **id** (*str*) – the id of the data table containing the item to delete
- **kwd** (*dict*) – (required) dictionary structure containing:
  - **payload: a dictionary itself containing:**
    - values: <TAB> separated list of column contents, there must be a value for all the columns of the data table

download_field_file (*trans*, **args*, ***kwargs*)

index (*trans*, **args*, ***kwargs*)
GET /api/tool_data: returns a list tool_data tables:

reload (*trans*, **args*, ***kwargs*)
GET /api/tool_data/{id}/reload

Reloads a tool_data table.

show (*trans*, **args*, ***kwargs*)

show_field (*trans*, **args*, ***kwargs*)
GET /api/tool_data/<id>/fields/<value>

Get information about a partiular field in a tool_data table

tool_shed_repositories Module

class galaxy.webapps.galaxy.api.tool_shed_repositories.**ToolShedRepositoriesController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

RESTful controller for interactions with tool shed repositories.

exported_workflows (*trans*, **args*, ***kwargs*)
GET /api/tool_shed_repositories/{encoded_tool_shed_repository_id}/exported_workflows

Display a list of dictionaries containing information about this tool shed repository's exported workflows.

Parameters **id** – the encoded id of the ToolShedRepository object

get_latest_installable_revision (*trans*, **args*, ***kwargs*)
POST /api/tool_shed_repositories/get_latest_installable_revision Get the latest installable revision of a specified repository from a specified Tool Shed.

Parameters **key** – the current Galaxy admin user's API key

The following parameters are included in the payload. :param tool_shed_url (required): the base URL of the Tool Shed from which to retrieve the Repository revision. :param name (required): the name of the Repository :param owner (required): the owner of the Repository

import_workflow (*trans*, **args*, ***kwargs*)
POST /api/tool_shed_repositories/import_workflow

Import the specified exported workflow contained in the specified installed tool shed repository into Galaxy.

Parameters

- **key** – the API key of the Galaxy user with which the imported workflow will be associated.

- **id** – the encoded id of the ToolShedRepository object

The following parameters are included in the payload. :param index: the index location of the workflow tuple in the list of exported workflows stored in the metadata for the specified repository

import_workflows (*trans*, **args*, ***kwargs*)
POST /api/tool_shed_repositories/import_workflows

Import all of the exported workflows contained in the specified installed tool shed repository into Galaxy.

Parameters

- **key** – the API key of the Galaxy user with which the imported workflows will be associated.
- **id** – the encoded id of the ToolShedRepository object

index (*trans*, **args*, ***kwargs*)
GET /api/tool_shed_repositories Display a list of dictionaries containing information about installed tool shed repositories.

install_repository_revision (*trans*, **args*, ***kwargs*)
POST /api/tool_shed_repositories/install_repository_revision Install a specified repository revision from a specified tool shed into Galaxy.

Parameters **key** – the current Galaxy admin user's API key

The following parameters are included in the payload. :param tool_shed_url (required): the base URL of the Tool Shed from which to install the Repository :param name (required): the name of the Repository :param owner (required): the owner of the Repository :param changeset_revision (required): the changeset_revision of the RepositoryMetadata object associated with the Repository :param new_tool_panel_section_label (optional): label of a new section to be added to the Galaxy tool panel in which to load

tools contained in the Repository. Either this parameter must be an empty string or the tool_panel_section_id parameter must be an empty string or both must be an empty string (both cannot be used simultaneously).

Parameters

- **(optional)** (*shed_tool_conf*) – id of the Galaxy tool panel section in which to load tools contained in the Repository. If this parameter is an empty string and the above new_tool_panel_section_label parameter is an empty string, tools will be loaded outside of any sections in the tool panel. Either this parameter must be an empty string or the tool_panel_section_id parameter must be an empty string of both must be an empty string (both cannot be used simultaneously).
- **(optional)** – Set to True if you want to install repository dependencies defined for the specified repository being installed. The default setting is False.
- **(optional)** – Set to True if you want to install tool dependencies defined for the specified repository being installed. The default setting is False.
- **(optional)** – The shed-related tool panel configuration file configured in the "tool_config_file" setting in the Galaxy config file (e.g., galaxy.ini). At least one shed-related tool panel config file is required to be configured. Setting this parameter to a specific file enables you to choose where the specified repository will be installed because the tool_path attribute of the <toolbox> from the specified file is used as the installation location (e.g., <toolbox tool_path="../shed_tools">). If this parameter is not set, a shed-related tool panel configuration file will be selected automatically.

install_repository_revisions (*trans*, **args*, ***kwargs*)

POST /api/tool_shed_repositories/install_repository_revisions Install one or more specified repository revisions from one or more specified tool sheds into Galaxy. The received parameters must be ordered lists so that positional values in tool_shed_urls, names, owners and changeset_revisions are associated.

It's questionable whether this method is needed as the above method for installing a single repository can probably cover all desired scenarios. We'll keep this one around just in case...

Parameters **key** – the current Galaxy admin user's API key

The following parameters are included in the payload. :param tool_shed_urls: the base URLs of the Tool Sheds from which to install a specified Repository :param names: the names of the Repositories to be installed :param owners: the owners of the Repositories to be installed :param changeset_revisions: the changeset_revisions of each RepositoryMetadata object associated with each Repository to be installed :param new_tool_panel_section_label: optional label of a new section to be added to the Galaxy tool panel in which to load

tools contained in the Repository. Either this parameter must be an empty string or the tool_panel_section_id parameter must be an empty string, as both cannot be used.

Parameters

- **tool_panel_section_id** – optional id of the Galaxy tool panel section in which to load tools contained in the Repository. If not set, tools will be loaded outside of any sections in the tool panel. Either this parameter must be an empty string or the tool_panel_section_id parameter must be an empty string, as both cannot be used.
- **(optional)** (*shed_tool_conf*) – Set to True if you want to install repository dependencies defined for the specified repository being installed. The default setting is False.
- **(optional)** – Set to True if you want to install tool dependencies defined for the specified repository being installed. The default setting is False.
- **(optional)** – The shed-related tool panel configuration file configured in the "tool_config_file" setting in the Galaxy config file (e.g., galaxy.ini). At least one shed-related tool panel config file is required to be configured. Setting this parameter to a specific file enables you to choose where the specified repository will be installed because the tool_path attribute of the <toolbox> from the specified file is used as the installation location (e.g., <toolbox tool_path="../shed_tools">). If this parameter is not set, a shed-related tool panel configuration file will be selected automatically.

repair_repository_revision (*trans*, **args*, ***kwargs*)

POST /api/tool_shed_repositories/repair_repository_revision Repair a specified repository revision previously installed into Galaxy.

Parameters **key** – the current Galaxy admin user's API key

The following parameters are included in the payload. :param tool_shed_url (required): the base URL of the Tool Shed from which the Repository was installed :param name (required): the name of the Repository :param owner (required): the owner of the Repository :param changeset_revision (required): the changeset_revision of the RepositoryMetadata object associated with the Repository

reset_metadata_on_installed_repositories (*trans*, **args*, ***kwargs*)

PUT /api/tool_shed_repositories/reset_metadata_on_installed_repositories

Resets all metadata on all repositories installed into Galaxy in an "orderly fashion".

Parameters **key** – the API key of the Galaxy admin user.

show(*trans*, **args*, ***kwargs*)
GET /api/tool_shed_repositories/{encoded_tool_shed_repsository_id} Display a dictionary containing information about a specified tool_shed_repository.

Parameters **id** – the encoded id of the ToolShedRepository object

galaxy.webapps.galaxy.api.tool_shed_repositories.**get_message_for_no_shed_tool_config**()

tools Module

users Module

API operations on User objects.

class galaxy.webapps.galaxy.api.users.**UserAPIController**(*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesTagsMi*
galaxy.web.base.controller.CreatesUsersMixin, *galaxy.web.base.controller.CreatesApiKeysI*

anon_user_api_value(*trans*)
Returns data for an anonymous user, truncated to only usage and quota_percent

api_key(*trans*, **args*, ***kwargs*)
POST /api/users/{encoded_user_id}/api_key Creates a new API key for specified user.

create(*trans*, **args*, ***kwargs*)
POST /api/users Creates a new Galaxy user.

delete(*trans*, **args*, ***kwargs*)

index(*trans*, **args*, ***kwargs*)
GET /api/users GET /api/users/deleted Displays a collection (list) of users.

show(*trans*, **args*, ***kwargs*)
GET /api/users/{encoded_user_id} GET /api/users/deleted/{encoded_user_id} GET /api/users/current Displays information about a user.

undelete(*trans*, **args*, ***kwargs*)

update(*trans*, **args*, ***kwargs*)

visualizations Module

Visualizations resource control over the API.

NOTE!: this is a work in progress and functionality and data structures may change often.

class galaxy.webapps.galaxy.api.visualizations.**VisualizationsController**(*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesVisual*
galaxy.web.base.controller.SharableMixin, *galaxy.model.item_attrs.UsesAnnotations*

RESTful controller for interactions with visualizations.

create(*trans*, **args*, ***kwargs*)
POST /api/visualizations creates a new visualization using the given payload

POST /api/visualizations?import_id={encoded_visualization_id} imports a copy of an existing visualization into the user's workspace

index (*trans*, **args*, ***kwargs*)
GET /api/visualizations:

show (*trans*, **args*, ***kwargs*)
GET /api/visualizations/{viz_id}

update (*trans*, **args*, ***kwargs*)
PUT /api/visualizations/{encoded_visualization_id}

workflows Module

API operations for Workflows

class galaxy.webapps.galaxy.api.workflows.**WorkflowsAPIController** (*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesStored galaxy.model.item_attrs.UsesAnnotations, galaxy.web.base.controller.SharableMixin

build_module (*trans*, **args*, ***kwargs*)
POST /api/workflows/build_module Builds module details including a tool model for the workflow editor.

cancel_invocation (*trans*, **args*, ***kwargs*)
DELETE /api/workflows/{workflow_id}/invocation/{invocation_id} Cancel the specified workflow invocation.

Parameters

- **workflow_id** (*str*) – the workflow id (required)
- **invocation_id** (*str*) – the usage id (required)

Raises exceptions.MessageException, exceptions.ObjectNotFound

create (*trans*, **args*, ***kwargs*)
POST /api/workflows

Run or create workflows from the api.

If installed_repository_file or from_history_id is specified a new workflow will be created for this user. Otherwise, workflow_id must be specified and this API method will cause a workflow to execute.

:param installed_repository_file The path of a workflow to import. Either workflow_id, installed_repository_file or from_history_id must be specified :type installed_repository_file str

Parameters

- **workflow_id** (*str*) – An existing workflow id. Either workflow_id, installed_repository_file or from_history_id must be specified
- **parameters** (*dict*) – If workflow_id is set - see _update_step_parameters()
- **ds_map** (*dict*) – If workflow_id is set - a dictionary mapping each input step id to a dictionary with 2 keys: 'src' (which can be 'ldda', 'ld' or 'hda') and 'id' (which should be the id of a LibraryDatasetDatasetAssociation, LibraryDataset or HistoryDatasetAssociation respectively)
- **no_add_to_history** (*str*) – If workflow_id is set - if present in the payload with any value, the input datasets will not be added to the selected history
- **history** (*str*) – If workflow_id is set - optional history where to run the workflow, either the name of a new history or "hist_id=HIST_ID" where HIST_ID is the id of an existing history. If not specified, the workflow will be run a new unnamed history

- **replacement_params** (*dict*) – If workflow_id is set - an optional dictionary used when renaming datasets
- **from_history_id** (*str*) – Id of history to extract a workflow from. Either workflow_id, installed_repository_file or from_history_id must be specified
- **job_ids** (*str*) – If from_history_id is set - optional list of jobs to include when extracting a workflow from history
- **dataset_ids** (*str*) – If from_history_id is set - optional list of HDA 'hid's corresponding to workflow inputs when extracting a workflow from history
- **dataset_collection_ids** (*str*) – If from_history_id is set - optional list of HDCA 'hid's corresponding to workflow inputs when extracting a workflow from history
- **workflow_name** (*str*) – If from_history_id is set - name of the workflow to create when extracting a workflow from history

delete (*trans*, **args*, ***kwargs*)

DELETE /api/workflows/{encoded_workflow_id} Deletes a specified workflow Author: rpark

copied from galaxy.web.controllers.workflows.py (delete)

import_new_workflow_deprecated (*trans*, **args*, ***kwargs*)

POST /api/workflows/upload Importing dynamic workflows from the api. Return newly generated workflow id. Author: rpark

currently assumes payload['workflow'] is a json representation of a workflow to be inserted into the database

Deprecated in favor to POST /api/workflows with encoded 'workflow' in payload the same way.

import_shared_workflow_deprecated (*trans*, **args*, ***kwargs*)

POST /api/workflows/import Import a workflow shared by other users.

Parameters **workflow_id** (*str*) – the workflow id (required)

Raises exceptions.MessageException, exceptions.ObjectNotFound

index (*trans*, **args*, ***kwargs*)

GET /api/workflows

Displays a collection of workflows.

Parameters **show_published** (*boolean*) – if True, show also published workflows

index_invocations (*trans*, **args*, ***kwargs*)

GET /api/workflows/{workflow_id}/invocations

Get the list of the workflow invocations

Parameters **workflow_id** (*str*) – the workflow id (required)

Raises exceptions.MessageException, exceptions.ObjectNotFound

invocation_step (*trans*, **args*, ***kwargs*)

GET /api/workflows/{workflow_id}/invocation/{invocation_id}/steps/{step_id}

Parameters

- **workflow_id** (*str*) – the workflow id (required)
- **invocation_id** (*str*) – the invocation id (required)
- **step_id** (*str*) – encoded id of the WorkflowInvocationStep (required)

- **payload** – payload containing update action information for running workflow.

Raises exceptions.MessageException, exceptions.ObjectNotFound

invoke (*trans*, **args*, ***kwargs*)

POST /api/workflows/{encoded_workflow_id}/invocations

Schedule the workflow specified by *workflow_id* to run.

show (*trans*, **args*, ***kwargs*)

GET /api/workflows/{encoded_workflow_id}

Displays information needed to run a workflow from the command line.

show_invocation (*trans*, **args*, ***kwargs*)

GET /api/workflows/{workflow_id}/invocation/{invocation_id} Get detailed description of workflow invocation

Parameters

- **workflow_id** (*str*) – the workflow id (required)
- **invocation_id** (*str*) – the invocation id (required)

Raises exceptions.MessageException, exceptions.ObjectNotFound

update (*trans*, **args*, ***kwargs*)

•**PUT /api/workflows/{id}** updates the workflow stored with `id`

Parameters

- **id** (*str*) – the encoded id of the workflow to update
- **payload** (*dict*) – a dictionary containing any or all the * workflow the json description of the workflow as would be

  produced by GET workflows/<id>/download or given to *POST workflows*

  The workflow contents will be updated to target this.

Return type dict

Returns serialized version of the workflow

update_invocation_step (*trans*, **args*, ***kwargs*)

PUT /api/workflows/{workflow_id}/invocation/{invocation_id}/steps/{step_id} Update state of running workflow step invocation - still very nebulous but this would be for stuff like confirming paused steps can proceed etc....

Parameters

- **workflow_id** (*str*) – the workflow id (required)
- **invocation_id** (*str*) – the usage id (required)
- **step_id** (*str*) – encoded id of the WorkflowInvocationStep (required)

Raises exceptions.MessageException, exceptions.ObjectNotFound

workflow_dict (*trans*, **args*, ***kwargs*)

GET /api/workflows/{encoded_workflow_id}/download Returns a selected workflow as a json dictionary.

1.2 lib

1.2.1 fpconst Module

1.2.2 galaxy Package

galaxy Package

Galaxy root package – this is a namespace package.

app Module

config Module

Universe configuration builder.

class galaxy.config.**Configuration**(***kwargs*)

Bases: object

check()

deprecated_options = ('database_file',)

ensure_tempdir()

get(*key*, *default*)

get_bool(*key*, *default*)

guess_galaxy_port()

is_admin_user(*user*)

Determine if the provided user is listed in *admin_users*.

NOTE: This is temporary, admin users will likely be specified in the database in the future.

resolve_path(*path*)

Resolve a path relative to Galaxy's root.

sentry_dsn_public

Sentry URL with private key removed for use in client side scripts, sentry server will need to be configured to accept events

class galaxy.config.**ConfiguresGalaxyMixin**

Shared code for configuring Galaxy-like app objects.

reindex_tool_search()

galaxy.config.**configure_logging**(*config*)

Allow some basic logging configuration to be read from ini file.

galaxy.config.**get_database_engine_options**(*kwargs*, *model_prefix='*')

Allow options for the SQLAlchemy database engine to be passed by using the prefix "database_engine_option".

galaxy.config.**resolve_path**(*path*, *root*)

If 'path' is relative make absolute by prepending 'root'

Subpackages

actions Package

admin Module Contains administrative functions

class galaxy.actions.admin.**AdminActions**
Bases: object

Mixin for controllers that provide administrative functionality.

datatypes Package

assembly Module velvet datatypes James E Johnson - University of Minnesota for velvet assembler tool in galaxy

class galaxy.datatypes.assembly.**Amos**(***kwd*)
Bases: *galaxy.datatypes.data.Text*

Class describing the AMOS assembly file

file_ext = 'afg'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff(*filename*)
Determines whether the file is an amos assembly file format Example:

```
{CTG
iid:1
eid:1
seq:
CCTCTCCTGTAGAGTTCAACCGA-GCCGGTAGAGTTTTATCA
.
qlt:
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
.
{TLE
src:1027
off:0
clr:618,0
gap:
250 612
.
}
}
```

class galaxy.datatypes.assembly.**Roadmaps**(***kwd*)
Bases: *galaxy.datatypes.data.Text*

Class describing the Sequences file generated by velveth

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff(*filename*)

Determines whether the file is a velveth produced RoadMap:: 142858 21 1 ROADMAP 1 ROADMAP 2 ...

class galaxy.datatypes.assembly.**Sequences**(***kwd*)
Bases: *galaxy.datatypes.sequence.Fasta*

Class describing the Sequences file generated by velveth

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff (*filename*)
Determines whether the file is a velveth produced fasta format The id line has 3 fields separated by tabs: sequence_name sequence_index cataegory:

```
>SEQUENCE_0_length_35   1       1
GGATATAGGGCCAACCCAACTCAACGGCCTGTCTT
>SEQUENCE_1_length_35   2       1
CGACGAATGACAGGTCACGAATTTGGCGGGGATTA
```

class galaxy.datatypes.assembly.**Velvet** (***kwd*)
Bases: *galaxy.datatypes.images.Html*

allow_datatype_change = False

composite_type = 'auto_primary_file'

file_ext = 'html'

generate_primary_file (*dataset=None*)

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

regenerate_primary_file (*dataset*)
cannot do this until we are setting metadata

set_meta (*dataset*, ***kwd*)

binary Module Binary classes

class galaxy.datatypes.binary.**Ab1** (***kwd*)
Bases: *galaxy.datatypes.binary.Binary*

Class describing an ab1 binary sequence file

display_peek (*dataset*)

file_ext = 'ab1'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek (*dataset*, *is_multi_byte=False*)

class galaxy.datatypes.binary.**Bam** (***kwd*)
Bases: *galaxy.datatypes.binary.Binary*

Class describing a BAM binary file

column_dataprovider (**args*, ***kwargs*)

data_sources = {'index': 'bigwig', 'data': 'bai'}

dataproviders = {'chunk64': <function chunk64_dataprovider at 0x7f497a522b18>, 'id-seq-qual': <function id_seq_q

dataset_content_needs_grooming (*file_name*)
See if file_name is a sorted BAM file

dict_dataprovider (**args*, ***kwargs*)

display_peek (*dataset*)

file_ext = 'bam'

genomic_region_dataprovider (**args*, ***kwargs*)

genomic_region_dict_dataprovider (**args*, ***kwargs*)

groom_dataset_content (*file_name*)
Ensures that the Bam file contents are sorted. This function is called on an output dataset after the content is initially generated.

header_dataprovider (**args*, ***kwargs*)

id_seq_qual_dataprovider (**args*, ***kwargs*)

init_meta (*dataset*, *copy_from=None*)

line_dataprovider (**args*, ***kwargs*)

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', bam_index (FileParameter): BAM Index

regex_line_dataprovider (**args*, ***kwargs*)

samtools_dataprovider (**args*, ***kwargs*)
Generic samtools interface - all options available through settings.

set_meta (*dataset*, *overwrite=True*, ***kwd*)
Creates the index for the BAM file.

set_peek (*dataset*, *is_multi_byte=False*)

sniff (*filename*)

track_type = 'ReadTrack'

class galaxy.datatypes.binary.**Bcf** (***kwd*)
Bases: *galaxy.datatypes.binary.Binary*

Class describing a BCF file

file_ext = 'bcf'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff (*filename*)

class galaxy.datatypes.binary.**BigBed** (***kwd*)
Bases: *galaxy.datatypes.binary.BigWig*

BigBed support from UCSC.

data_sources = {'data_standalone': 'bigbed'}

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

class galaxy.datatypes.binary.**BigWig** (***kwd*)
Bases: *galaxy.datatypes.binary.Binary*

Accessing binary BigWig files from UCSC. The supplemental info in the paper has the binary details: http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq351v1

data_sources = {'data_standalone': 'bigwig'}

display_peek (*dataset*)

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek (*dataset*, *is_multi_byte=False*)

sniff (*filename*)

track_type = 'LineTrack'

class galaxy.datatypes.binary.**Binary**(***kwd*)
Bases: galaxy.datatypes.data.Data

Binary data

display_data(*trans*, *dataset*, *preview=False*, *filename=None*, *to_ext=None*, *size=None*, *offset=None*, ***kwd*)

get_mime()
Returns the mime type of the datatype

static is_ext_unsniffable(*ext*)

static is_sniffable_binary(*filename*)

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

static register_sniffable_binary_format(*data_type*, *ext*, *type_class*)

static register_unsniffable_binary_ext(*ext*)

set_peek(*dataset*, *is_multi_byte=False*)
Set the peek and blurb text

sniffable_binary_formats = [{'ext': 'bam', 'type': 'bam', 'class': <class 'galaxy.datatypes.binary.Bam'>}, {'ext':

unsniffable_binary_formats = ['ab1', 'compressed_archive', 'asn1-binary', 'h5', 'scf']

class galaxy.datatypes.binary.**CompressedArchive**(***kwd*)
Bases: galaxy.datatypes.binary.Binary

Class describing an compressed binary file This class can be sublass'ed to implement archive filetypes that will not be unpacked by upload.py.

compressed = True

display_peek(*dataset*)

file_ext = 'compressed_archive'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek(*dataset*, *is_multi_byte=False*)

class galaxy.datatypes.binary.**GeminiSQLite**(***kwd*)
Bases: galaxy.datatypes.binary.SQlite

Class describing a Gemini Sqlite database

display_peek(*dataset*)

file_ext = 'gemini.sqlite'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', tables (ListParameter): Database Tables,

set_meta(*dataset*, *overwrite=True*, ***kwd*)

set_peek(*dataset*, *is_multi_byte=False*)

sniff(*filename*)

class galaxy.datatypes.binary.**GenericAsn1Binary**(***kwd*)
Bases: galaxy.datatypes.binary.Binary

Class for generic ASN.1 binary format

file_ext = 'asn1-binary'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

class galaxy.datatypes.binary.**H5**(***kwd*)
Bases: galaxy.datatypes.binary.Binary

Class describing an HDF5 file

display_peek(*dataset*)

file_ext = 'h5'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek(*dataset*, *is_multi_byte=False*)

class galaxy.datatypes.binary.**SQlite**(***kwd*)
Bases: galaxy.datatypes.binary.Binary

Class describing a Sqlite database

dataproviders = {'chunk64': <function chunk64_dataprovider at 0x7f497a522b18>, 'chunk': <function chunk_datap

display_peek(*dataset*)

file_ext = 'sqlite'

init_meta(*dataset*, *copy_from=None*)

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', tables (ListParameter): Database Tables,

set_meta(*dataset*, *overwrite=True*, ***kwd*)

set_peek(*dataset*, *is_multi_byte=False*)

sniff(*filename*)

sqlite_datadictprovider(**args*, ***kwargs*)

sqlite_dataprovider(**args*, ***kwargs*)

sqlite_datatableprovider(**args*, ***kwargs*)

class galaxy.datatypes.binary.**Scf**(***kwd*)
Bases: galaxy.datatypes.binary.Binary

Class describing an scf binary sequence file

display_peek(*dataset*)

file_ext = 'scf'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek(*dataset*, *is_multi_byte=False*)

class galaxy.datatypes.binary.**Sff**(***kwd*)
Bases: galaxy.datatypes.binary.Binary

Standard Flowgram Format (SFF)

display_peek(*dataset*)

file_ext = 'sff'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek(*dataset*, *is_multi_byte=False*)

sniff(*filename*)

class galaxy.datatypes.binary.**Sra**(**kwd)
Bases: galaxy.datatypes.binary.Binary

Sequence Read Archive (SRA) datatype originally from mdshw5/sra-tools-galaxy

display_peek(*dataset*)

file_ext = 'sra'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek(*dataset*, *is_multi_byte=False*)

sniff(*filename*)
The first 8 bytes of any NCBI sra file is 'NCBI.sra', and the file is binary. For details about the format, see http://www.ncbi.nlm.nih.gov/books/n/helpsra/SRA_Overview_BK/#SRA_Overview_BK.4_SRA_Data_Structure

class galaxy.datatypes.binary.**TwoBit**(**kwd)
Bases: galaxy.datatypes.binary.Binary

Class describing a TwoBit format nucleotide file

display_peek(*dataset*)

file_ext = 'twobit'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek(*dataset*, *is_multi_byte=False*)

sniff(*filename*)

class galaxy.datatypes.binary.**Xlsx**(**kwd)
Bases: galaxy.datatypes.binary.Binary

Class for Excel 2007 (xlsx) files

file_ext = 'xlsx'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*)

checkers Module

galaxy.datatypes.checkers.**check_binary**(*name*, *file_path=True*)
galaxy.datatypes.checkers.**check_bz2**(*file_path*)

galaxy.datatypes.checkers.**check_gzip**(*file_path*)

galaxy.datatypes.checkers.**check_html**(*file_path*, *chunk=None*)

galaxy.datatypes.checkers.**check_image**(*file_path*)

galaxy.datatypes.checkers.**check_zip**(*file_path*)

galaxy.datatypes.checkers.**is_bz2**(*file_path*)

galaxy.datatypes.checkers.**is_gzip**(*file_path*)

chrominfo Module

class galaxy.datatypes.chrominfo.**ChromInfo**(**kwd)
Bases: galaxy.datatypes.tabular.Tabular

file_ext = 'len'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

coverage Module Coverage datatypes

class galaxy.datatypes.coverage.**LastzCoverage**(***kwd*)

Bases: *galaxy.datatypes.tabular.Tabular*

file_ext = 'coverage'

get_track_resolution(*dataset*, *start*, *end*)

get_track_window(*dataset*, *data*, *start*, *end*)

Assumes we have a numpy file.

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

data Module

class galaxy.datatypes.data.**Data**(***kwd*)

Bases: object

Base class for all datatypes. Implements basic interfaces as well as class methods for metadata.

```
>>> class DataTest( Data ):
...     MetadataElement( name="test" )
...
>>> DataTest.metadata_spec.test.name
'test'
>>> DataTest.metadata_spec.test.desc
'test'
>>> type( DataTest.metadata_spec.test.param )
<class 'galaxy.datatypes.metadata.MetadataParameter'>
```

CHUNKABLE = False

add_composite_file(*name*, ***kwds*)

add_display_app(*app_id*, *label*, *file_function*, *links_function*)

Adds a display app to the datatype. app_id is a unique id label is the primary display label, e.g., display at 'UCSC' file_function is a string containing the name of the function that returns a properly formatted display links_function is a string containing the name of the function that returns a list of (link_name,link)

add_display_application(*display_application*)

New style display applications

after_setting_metadata(*dataset*)

This function is called on the dataset after metadata is set.

allow_datatype_change = True

as_display_type(*dataset*, *type*, ***kwd*)

Returns modified file contents for a particular display type

base_dataprovider(**args*, ***kwargs*)

before_setting_metadata(*dataset*)

This function is called on the dataset before metadata is set.

chunk64_dataprovider(**args*, ***kwargs*)

chunk_dataprovider(**args*, ***kwargs*)

clear_display_apps()

composite_files = {}

composite_type = None

convert_dataset(*trans*, *original_dataset*, *target_type*, *return_output=False*, *visible=True*, *deps=None*, *set_output_history=True*)
This function adds a job to the queue to convert a dataset to another type. Returns a message about success/failure.

copy_safe_peek = True

data_sources = {}

dataprovider(*dataset*, *data_format*, ***settings*)
Base dataprovider factory for all datatypes that returns the proper provider for the given *data_format* or raises a *NoProviderAvailable*.

dataproviders = {'chunk64': <function chunk64_dataprovider at 0x7f497a522b18>, 'base': <function base_dataprovi

dataset_content_needs_grooming(*file_name*)
This function is called on an output dataset file after the content is initially generated.

display_data(*trans*, *data*, *preview=False*, *filename=None*, *to_ext=None*, *size=None*, *offset=None*, ***kwd*)
Old display method, for transition - though still used by API and test framework. Datatypes should be very careful if overridding this method and this interface between datatypes and Galaxy will likely change.

TOOD: Document alternatives to overridding this method (data providers?).

display_info(*dataset*)
Returns formatted html of dataset info

display_name(*dataset*)
Returns formatted html of dataset name

display_peek(*dataset*)
Create HTML table, used for displaying peek

find_conversion_destination(*dataset*, *accepted_formats*, *datatypes_registry*, ***kwd*)
Returns (target_ext, existing converted dataset)

generate_auto_primary_file(*dataset=None*)

get_composite_files(*dataset=None*)

get_converter_types(*original_dataset*, *datatypes_registry*)
Returns available converters by type for this dataset

get_display_application(*key*, *default=None*)

get_display_applications_by_dataset(*dataset*, *trans*)

get_display_label(*type*)
Returns primary label for display app

get_display_links(*dataset*, *type*, *app*, *base_url*, *target_frame='_blank'*, ***kwd*)
Returns a list of tuples of (name, link) for a particular display type. No check on 'access' permissions is done here - if you can view the dataset, you can also save it or send it to a destination outside of Galaxy, so Galaxy security restrictions do not apply anyway.

get_display_types()
Returns display types available

get_max_optional_metadata_filesize()

get_mime()
Returns the mime type of the datatype

get_raw_data (*dataset*)
Returns the full data. To stream it open the file_name and read/write as needed

get_visualizations (*dataset*)
Returns a list of visualizations for datatype.

groom_dataset_content (*file_name*)
This function is called on an output dataset file if dataset_content_needs_grooming returns True.

has_dataprovider (*data_format*)
Returns True if *data_format* is available in *dataproviders*.

has_resolution

init_meta (*dataset*, *copy_from=None*)

is_binary = True

matches_any (*target_datatypes*)
Check if this datatype is of any of the target_datatypes or is a subtype thereof.

max_optional_metadata_filesize

static merge (*split_files*, *output_file*)
Merge files with copy.copyfileobj() will not hit the max argument limitation of cat. gz and bz2 files are also working.

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'
dictionary of metadata fields for this datatype:

missing_meta (*dataset*, *check=[]*, *skip=[]*)
Checks for empty metadata values, Returns True if non-optional metadata is missing Specifying a list of 'check' values will only check those names provided; when used, optionality is ignored Specifying a list of 'skip' items will return True even when a named metadata value is missing

primary_file_name = 'index'

remove_display_app (*app_id*)
Removes a display app from the datatype

repair_methods (*dataset*)
Unimplemented method, returns dict with method/option for repairing errors

set_max_optional_metadata_filesize (*max_value*)

set_meta (*dataset*, *overwrite=True*, ***kwd*)
Unimplemented method, allows guessing of metadata from contents of file

set_peek (*dataset*, *is_multi_byte=False*)
Set the peek and blurb text

set_raw_data (*dataset*, *data*)
Saves the data on the disc

supported_display_apps = {}

track_type = None

validate (*dataset*)
Unimplemented validate, return no exceptions

writable_files

write_from_stream (*dataset*, *stream*)
Writes data from a stream

class galaxy.datatypes.data.**DataMeta**(*name*, *bases*, *dict_*)
Bases: type

Metaclass for Data class. Sets up metadata spec.

class galaxy.datatypes.data.**GenericAsn1**(**kwd)
Bases: galaxy.datatypes.data.Text

Class for generic ASN.1 text format

file_ext = 'asn1'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.data.**LineCount**(**kwd)
Bases: galaxy.datatypes.data.Text

Dataset contains a single line with a single integer that denotes the line count for a related dataset. Used for custom builds.

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.data.**Newick**(**kwd)
Bases: galaxy.datatypes.data.Text

New Hampshire/Newick Format

file_ext = 'nhx'

get_visualizations(*dataset*)
Returns a list of visualizations for datatype.

init_meta(*dataset*, *copy_from=None*)

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff(*filename*)
Returning false as the newick format is too general and cannot be sniffed.

class galaxy.datatypes.data.**Nexus**(**kwd)
Bases: galaxy.datatypes.data.Text

Nexus format as used By Paup, Mr Bayes, etc

file_ext = 'nex'

get_visualizations(*dataset*)
Returns a list of visualizations for datatype.

init_meta(*dataset*, *copy_from=None*)

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff(*filename*)
All Nexus Files Simply puts a '#NEXUS' in its first line

class galaxy.datatypes.data.**Text**(**kwd)
Bases: galaxy.datatypes.data.Data

count_data_lines(*dataset*)
Count the number of lines of data in dataset, skipping all blank lines and comments.

dataproviders = {'chunk64': <function chunk64_dataprovider at 0x7f497a522b18>, 'base': <function base_dataprovi

estimate_file_lines(*dataset*)
Perform a rough estimate by extrapolating number of lines from a small read.

file_ext = 'txt'

get_mime()
Returns the mime type of the datatype

line_class = 'line'
Add metadata elements

line_dataprovider(**args*, ***kwargs*)
Returns an iterator over the dataset's lines (that have been 'strip'ed) optionally excluding blank lines and lines that start with a comment character.

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

regex_line_dataprovider(**args*, ***kwargs*)
Returns an iterator over the dataset's lines optionally including/excluding lines that match one or more regex filters.

set_meta(*dataset*, ***kwd*)
Set the number of lines of data in dataset.

set_peek(*dataset*, *line_count=None*, *is_multi_byte=False*, *WIDTH=256*, *skipchars=[]*)
Set the peek. This method is used by various subclasses of Text.

set_raw_data(*dataset*, *data*)
Saves the data on the disc

classmethod split(*input_datasets*, *subdir_generator_function*, *split_params*)
Split the input files by line.

write_from_stream(*dataset*, *stream*)
Writes data from a stream

galaxy.datatypes.data.**get_file_peek**(*file_name*, *is_multi_byte=False*, *WIDTH=256*, *LINE_COUNT=5*, *skipchars=[]*)
Returns the first LINE_COUNT lines wrapped to WIDTH

>>> fname = get_test_fname('4.bed') ## >>> get_file_peek(fname) ## 'chr22 30128507 31828507 uc003bnx.1_cds_2_0_chr22_29227_f 0 +
'

galaxy.datatypes.data.**get_test_fname**(*fname*)
Returns test data filename

genetics Module rgenetics datatypes Use at your peril Ross Lazarus for the rgenetics and galaxy projects

genome graphs datatypes derived from Interval datatypes genome graphs datasets have a header row with appropriate columnames The first column is always the marker - eg columname = rs, first row= rs12345 if the rows are snps subsequent row values are all numeric ! Will fail if any non numeric (eg '+' or 'NA') values ross lazarus for rgenetics august 20 2007

class galaxy.datatypes.genetics.**Affybatch**(***kwd*)
Bases: *galaxy.datatypes.genetics.RexpBase*

derived class for BioC data structures in Galaxy

file_ext = 'affybatch'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**Eigenstratgeno**(***kwd*)
Bases: *galaxy.datatypes.genetics.Rgenetics*

Eigenstrat format - may be able to get rid of this if we move to shellfish Rgenetics data collections

file_ext = 'eigenstratgeno'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**Eigenstratpca**(***kwd*)
Bases: *galaxy.datatypes.genetics.Rgenetics*

Eigenstrat PCA file for case control adjustment Rgenetics data collections

file_ext = 'eigenstratpca'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**Eset**(***kwd*)
Bases: *galaxy.datatypes.genetics.RexpBase*

derived class for BioC data structures in Galaxy

file_ext = 'eset'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**Fped**(***kwd*)
Bases: *galaxy.datatypes.genetics.Rgenetics*

FBAT pedigree format - single file, map is header row of rs numbers. Strange. Rgenetics data collections

file_ext = 'fped'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**Fphe**(***kwd*)
Bases: *galaxy.datatypes.genetics.Rgenetics*

fbat pedigree file - mad format with ! as first char on header row Rgenetics data collections

file_ext = 'fphe'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**GenomeGraphs**(***kwd*)
Bases: *galaxy.datatypes.tabular.Tabular*

Tab delimited data containing a marker id and any number of numeric values

as_ucsc_display_file(*dataset*, ***kwd*)
Returns file

file_ext = 'gg'

get_mime()
Returns the mime type of the datatype

make_html_table(*dataset*, *skipchars=[]*)
Create HTML table, used for displaying peek

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta(*dataset*, ***kwd*)

sniff(*filename*)
Determines whether the file is in gg format

ucsc_links(*dataset*, *type*, *app*, *base_url*)
from the ever-helpful angie hinrichs angie@soe.ucsc.edu a genome graphs call looks like this

http://genome.ucsc.edu/cgi-bin/hgGenome?clade=mammal&org=Human&db=hg18&hgGenome_dataSetName=dname
&hgGenome_dataSetDescription=test&hgGenome_formatType=best%20guess&hgGenome_markerType=best%20guess
&hgGenome_columnLabels=best%20guess&hgGenome_maxVal=&hgGenome_labelVals=
&hgGenome_maxGapToFill=25000000&hgGenome_uploadFile=http://galaxy.esphealth.org/datasets/333/display/index
&hgGenome_doSubmitUpload=submit

Galaxy gives this for an interval file

http://genome.ucsc.edu/cgi-bin/hgTracks?db=hg18&position=chr1:1-1000&hgt.customText=
http%3A%2F%2Fgalaxy.esphealth.org%2Fdisplay_as%3Fid%3D339%26display_app%3Ducsc

validate(*dataset*)
Validate a gg file - all numeric after header row

class galaxy.datatypes.genetics.**Lped**(***kwd*)
Bases: *galaxy.datatypes.genetics.Rgenetics*

linkage pedigree (ped,map) Rgenetics data collections

file_ext = 'lped'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**MAlist**(***kwd*)
Bases: *galaxy.datatypes.genetics.RexpBase*

derived class for BioC data structures in Galaxy

file_ext = 'malist'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**Pbed**(***kwd*)
Bases: *galaxy.datatypes.genetics.Rgenetics*

Plink Binary compressed 2bit/geno Rgenetics data collections

file_ext = 'pbed'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**Phe**(***kwd*)
Bases: *galaxy.datatypes.genetics.Rgenetics*

Phenotype file

file_ext = 'phe'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**Pheno**(***kwd*)
Bases: *galaxy.datatypes.tabular.Tabular*

base class for pheno files

file_ext = 'pheno'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**Pphe**(***kwd*)
Bases: *galaxy.datatypes.genetics.Rgenetics*

Plink phenotype file - header must have FID IID... Rgenetics data collections

file_ext = 'pphe'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**RexpBase**(***kwd*)

Bases: galaxy.datatypes.images.Html

base class for BioC data structures in Galaxy must be constructed with the pheno data in place since that goes into the metadata for each instance

allow_datatype_change = False

composite_type = 'auto_primary_file'

display_peek(*dataset*)

Returns formatted html of peek

file_ext = 'rexpbase'

generate_primary_file(*dataset=None*)

This is called only at upload to write the html file cannot rename the datasets here - they come with the default unfortunately

get_file_peek(*filename*)

can't really peek at a filename - need the extra_files_path and such?

get_mime()

Returns the mime type of the datatype

get_peek(*dataset*)

expects a .pheno file in the extra_files_dir - ugh

get_phecols(*phenolist=[], maxConc=20*)

sept 2009: cannot use whitespace to split - make a more complex structure here and adjust the methods that rely on this structure return interesting phenotype column names for an rexpression eset or affybatch to use in array subsetting and so on. Returns a data structure for a dynamic Galaxy select parameter. A column with only 1 value doesn't change, so is not interesting for analysis. A column with a different value in every row is equivalent to a unique identifier so is also not interesting for anova or limma analysis - both these are removed after the concordance (count of unique terms) is constructed for each column. Then a complication - each remaining pair of columns is tested for redundancy - if two columns are always paired, then only one is needed :)

get_pheno(*dataset*)

expects a .pheno file in the extra_files_dir - ugh note that R is wierd and adds the row.name in the header so the columns are all wrong - unless you tell it not to. A file can be written as write.table(file='foo.pheno',pData(foo),sep=' ',quote=F,row.names=F)

html_table = None

init_meta(*dataset, copy_from=None*)

is_binary = True

make_html_table(*pp='nothing supplied from peek\n'*)

Create HTML table, used for displaying peek

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

regenerate_primary_file(*dataset*)

cannot do this until we are setting metadata

set_meta(*dataset, **kwd*)

NOTE we apply the tabular machinary to the phenodata extracted from a BioC eSet or affybatch.

set_peek(*dataset, **kwd*)

expects a .pheno file in the extra_files_dir - ugh note that R is weird and does not include the row.name in the header. why?

class galaxy.datatypes.genetics.**Rgenetics**(***kwd*)
Bases: galaxy.datatypes.images.Html

base class to use for rgenetics datatypes derived from html - composite datatype elements stored in extra files path

allow_datatype_change = False

composite_type = 'auto_primary_file'

file_ext = 'rgenetics'

generate_primary_file(*dataset=None*)

get_mime()
Returns the mime type of the datatype

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

regenerate_primary_file(*dataset*)
cannot do this until we are setting metadata

set_meta(*dataset*, ***kwd*)
for lped/pbed eg

class galaxy.datatypes.genetics.**SNPMatrix**(***kwd*)
Bases: galaxy.datatypes.genetics.Rgenetics

BioC SNPMatrix Rgenetics data collections

file_ext = 'snpmatrix'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_peek(*dataset*, ***kwd*)

sniff(*filename*)
need to check the file header hex code

class galaxy.datatypes.genetics.**Snptest**(***kwd*)
Bases: galaxy.datatypes.genetics.Rgenetics

BioC snptest Rgenetics data collections

file_ext = 'snptest'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**ldIndep**(***kwd*)
Bases: galaxy.datatypes.genetics.Rgenetics

LD (a good measure of redundancy of information) depleted Plink Binary compressed 2bit/geno This is really a plink binary, but some tools work better with less redundancy so are constrained to these files

file_ext = 'ldreduced'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**rgFeatureList**(***kwd*)
Bases: galaxy.datatypes.genetics.rgTabList

for featureid lists of exclusions or inclusions in the clean tool output from QC eg low maf, high missingness, bad hwe in controls, excess mendel errors,... featureid subsets on statistical criteria -> specialized display such as gg same infrastructure for expression?

file_ext = 'rgFList'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.genetics.**rgSampleList**(***kwd*)
Bases: galaxy.datatypes.genetics.rgTabList

for sampleid exclusions or inclusions in the clean tool output from QC eg excess het, gender error, ibd pair member,eigen outlier,excess mendel errors,... since they can be uploaded, should be flexible but they are persistent at least same infrastructure for expression?

file_ext = 'rgSList'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff(*filename*)

class galaxy.datatypes.genetics.**rgTabList**(***kwd*)
Bases: galaxy.datatypes.tabular.Tabular

for sampleid and for featureid lists of exclusions or inclusions in the clean tool featureid subsets on statistical criteria -> specialized display such as gg

display_peek(*dataset*)
Returns formated html of peek

file_ext = 'rgTList'

get_mime()
Returns the mime type of the datatype

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

images Module Image classes

class galaxy.datatypes.images.**Bmp**(***kwd*)
Bases: galaxy.datatypes.images.Image

file_ext = 'bmp'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in bmp format.

class galaxy.datatypes.images.**Eps**(***kwd*)
Bases: galaxy.datatypes.images.Image

file_ext = 'eps'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in eps format.

class galaxy.datatypes.images.**Gif**(***kwd*)
Bases: galaxy.datatypes.images.Image

file_ext = 'gif'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in gif format.

class galaxy.datatypes.images.**Gmaj**(**kwd)
Bases: galaxy.datatypes.data.Data

Class describing a GMAJ Applet

copy_safe_peek = False

display_peek(*dataset*)

file_ext = 'gmaj.zip'

get_mime()
Returns the mime type of the datatype

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek(*dataset*, *is_multi_byte=False*)

sniff(*filename*)
NOTE: the sniff.convert_newlines() call in the upload utility will keep Gmaj data types from being correctly sniffed, but the files can be uploaded (they'll be sniffed as 'txt'). This sniff function is here to provide an example of a sniffer for a zip file.

class galaxy.datatypes.images.**Html**(**kwd)
Bases: galaxy.datatypes.data.Text

Class describing an html file

file_ext = 'html'

get_mime()
Returns the mime type of the datatype

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_peek(*dataset*, *is_multi_byte=False*)

sniff(*filename*)
Determines whether the file is in html format

```
>>> fname = get_test_fname( 'complete.bed' )
>>> Html().sniff( fname )
False
>>> fname = get_test_fname( 'file.html' )
>>> Html().sniff( fname )
True
```

class galaxy.datatypes.images.**Im**(**kwd)
Bases: galaxy.datatypes.images.Image

file_ext = 'im'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in im format.

class galaxy.datatypes.images.**Image**(**kwd)
Bases: galaxy.datatypes.data.Data

Class describing an image

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek(*dataset*, *is_multi_byte=False*)

sniff(*filename*)

class galaxy.datatypes.images.**Jpg**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'jpg'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in jpg format.

class galaxy.datatypes.images.**Laj**(***kwd*)
Bases: *galaxy.datatypes.data.Text*

Class describing a LAJ Applet

copy_safe_peek = False

display_peek(*dataset*)

file_ext = 'laj'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_peek(*dataset*, *is_multi_byte=False*)

class galaxy.datatypes.images.**Pbm**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'pbm'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in PBM format

class galaxy.datatypes.images.**Pcd**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'pcd'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in pcd format.

class galaxy.datatypes.images.**Pcx**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'pcx'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in pcx format.

class galaxy.datatypes.images.**Pdf**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'pdf'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*)
Determine if the file is in pdf format.

class galaxy.datatypes.images.**Pgm**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'pgm'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in PGM format

class galaxy.datatypes.images.**Png**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'png'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in png format.

class galaxy.datatypes.images.**Ppm**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'ppm'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in ppm format.

class galaxy.datatypes.images.**Psd**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'psd'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in psd format.

class galaxy.datatypes.images.**Rast**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'rast'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in rast format

class galaxy.datatypes.images.**Rgb**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'rgb'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in RGB format.

class galaxy.datatypes.images.**Tiff**(***kwd*)
Bases: *galaxy.datatypes.images.Image*

file_ext = 'tiff'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in tiff format.

class galaxy.datatypes.images.**Xbm**(***kwd*)
Bases: galaxy.datatypes.images.Image

file_ext = 'xbm'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in XBM format.

class galaxy.datatypes.images.**Xpm**(***kwd*)
Bases: galaxy.datatypes.images.Image

file_ext = 'xpm'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

sniff(*filename*, *image=None*)
Determine if the file is in XPM format.

galaxy.datatypes.images.**create_applet_tag_peek**(*class_name*, *archive*, *params*)

interval Module Interval datatypes

class galaxy.datatypes.interval.**Bed**(***kwd*)
Bases: galaxy.datatypes.interval.Interval

Tab delimited data in BED format

as_ucsc_display_file(*dataset*, ***kwd*)
Returns file contents with only the bed data. If bed 6+, treat as interval.

data_sources = {'index': 'bigwig', 'data': 'tabix', 'feature_search': 'fli'}

file_ext = 'bed'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta(*dataset*, *overwrite=True*, ***kwd*)
Sets the metadata information for datasets previously determined to be in bed format.

sniff(*filename*)
Checks for 'bedness'

BED lines have three required fields and nine additional optional fields. The number of fields per line must be consistent throughout any single set of data in an annotation track. The order of the optional fields is binding: lower-numbered fields must always be populated if higher-numbered fields are used. The data type of all 12 columns is: 1-str, 2-int, 3-int, 4-str, 5-int, 6-str, 7-int, 8-int, 9-int or list, 10-int, 11-list, 12-list

For complete details see http://genome.ucsc.edu/FAQ/FAQformat#format1

```
>>> fname = get_test_fname( 'test_tab.bed' )
>>> Bed().sniff( fname )
True
>>> fname = get_test_fname( 'interval1.bed' )
>>> Bed().sniff( fname )
True
>>> fname = get_test_fname( 'complete.bed' )
>>> Bed().sniff( fname )
True
```

track_type = 'FeatureTrack'
Add metadata elements

class galaxy.datatypes.interval.**Bed12**(***kwd*)
Bases: galaxy.datatypes.interval.BedStrict

Tab delimited data in strict BED format - no non-standard columns allowed; column count forced to 12

file_ext = 'bed12'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.interval.**Bed6**(***kwd*)
Bases: galaxy.datatypes.interval.BedStrict

Tab delimited data in strict BED format - no non-standard columns allowed; column count forced to 6

file_ext = 'bed6'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.interval.**BedGraph**(***kwd*)
Bases: galaxy.datatypes.interval.Interval

Tab delimited chrom/start/end/datavalue dataset

as_ucsc_display_file(*dataset*, ***kwd*)
Returns file contents as is with no modifications. TODO: this is a functional stub and will need to be enhanced moving forward to provide additional support for bedgraph.

data_sources = {'index': 'bigwig', 'data': 'bigwig'}

file_ext = 'bedgraph'

get_estimated_display_viewport(*dataset*, *chrom_col=0*, *start_col=1*, *end_col=2*)
Set viewport based on dataset's first 100 lines.

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

track_type = 'LineTrack'

class galaxy.datatypes.interval.**BedStrict**(***kwd*)
Bases: galaxy.datatypes.interval.Bed

Tab delimited data in strict BED format - no non-standard columns allowed

allow_datatype_change = False

file_ext = 'bedstrict'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta(*dataset*, *overwrite=True*, ***kwd*)

sniff(*filename*)

class galaxy.datatypes.interval.**ChromatinInteractions**(***kwd*)
Bases: galaxy.datatypes.interval.Interval

Chromatin interactions obtained from 3C/5C/Hi-C experiments.

column_names = ['Chrom1', 'Start1', 'End1', 'Chrom2', 'Start2', 'End2', 'Value']
Add metadata elements

data_sources = {'index': 'bigwig', 'data': 'tabix'}

file_ext = 'chrint'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff(*filename*)

track_type = 'DiagonalHeatmapTrack'

class galaxy.datatypes.interval.**CustomTrack**(***kwd*)

Bases: *galaxy.datatypes.tabular.Tabular*

UCSC CustomTrack

display_peek(*dataset*)

Returns formated html of peek

file_ext = 'customtrack'

get_estimated_display_viewport(*dataset*, *chrom_col=None*, *start_col=None*, *end_col=None*)

Return a chrom, start, stop tuple for viewing a file.

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta(*dataset*, *overwrite=True*, ***kwd*)

sniff(*filename*)

Determines whether the file is in customtrack format.

CustomTrack files are built within Galaxy and are basically bed or interval files with the first line looking something like this.

track name="User Track" description="User Supplied Track (from Galaxy)" color=0,0,0 visibility=1

```
>>> fname = get_test_fname( 'complete.bed' )
>>> CustomTrack().sniff( fname )
False
>>> fname = get_test_fname( 'ucsc.customtrack' )
>>> CustomTrack().sniff( fname )
True
```

ucsc_links(*dataset*, *type*, *app*, *base_url*)

class galaxy.datatypes.interval.**ENCODEPeak**(***kwd*)

Bases: *galaxy.datatypes.interval.Interval*

Human ENCODE peak format. There are both broad and narrow peak formats. Formats are very similar; narrow peak has an additional column, though.

Broad peak (http://genome.ucsc.edu/FAQ/FAQformat#format13): This format is used to provide called regions of signal enrichment based on pooled, normalized (interpreted) data. It is a BED 6+3 format.

Narrow peak http://genome.ucsc.edu/FAQ/FAQformat#format12 and : This format is used to provide called peaks of signal enrichment based on pooled, normalized (interpreted) data. It is a BED6+4 format.

column_names = ['Chrom', 'Start', 'End', 'Name', 'Score', 'Strand', 'SignalValue', 'pValue', 'qValue', 'Peak']

data_sources = {'index': 'bigwig', 'data': 'tabix'}

Add metadata elements

file_ext = 'encodepeak'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff(*filename*)

class galaxy.datatypes.interval.**Gff**(***kwd*)

Bases: *galaxy.datatypes.tabular.Tabular*, galaxy.datatypes.interval._RemoteCallMixin

Tab delimited data in Gff format

column_names = ['Seqname', 'Source', 'Feature', 'Start', 'End', 'Score', 'Strand', 'Frame', 'Group']

data_sources = {'index': 'bigwig', 'data': 'interval_index', 'feature_search': 'fli'}

dataproviders = {'dataset-column': <function dataset_column_dataprovider at 0x7f497b0bd320>, 'chunk64': <functi

display_peek (*dataset*)
Returns formated html of peek

file_ext = 'gff'

gbrowse_links (*dataset*, *type*, *app*, *base_url*)

genomic_region_dataprovider (**args*, ***kwargs*)

genomic_region_dict_dataprovider (**args*, ***kwargs*)

get_estimated_display_viewport (*dataset*)
Return a chrom, start, stop tuple for viewing a file. There are slight differences between gff 2 and gff 3 formats. This function should correctly handle both...

interval_dataprovider (**args*, ***kwargs*)

interval_dict_dataprovider (**args*, ***kwargs*)

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_attribute_metadata (*dataset*)
Sets metadata elements for dataset's attributes.

set_meta (*dataset*, *overwrite=True*, ***kwd*)

sniff (*filename*)
Determines whether the file is in gff format

GFF lines have nine required fields that must be tab-separated.

For complete details see http://genome.ucsc.edu/FAQ/FAQformat#format3

```
>>> fname = get_test_fname( 'gff_version_3.gff' )
>>> Gff().sniff( fname )
False
>>> fname = get_test_fname( 'test.gff' )
>>> Gff().sniff( fname )
True
```

track_type = 'FeatureTrack'
Add metadata elements

ucsc_links (*dataset*, *type*, *app*, *base_url*)

class galaxy.datatypes.interval.**Gff3** (***kwd*)
Bases: *galaxy.datatypes.interval.Gff*

Tab delimited data in Gff3 format

column_names = ['Seqid', 'Source', 'Type', 'Start', 'End', 'Score', 'Strand', 'Phase', 'Attributes']

file_ext = 'gff3'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta (*dataset*, *overwrite=True*, ***kwd*)

sniff (*filename*)
Determines whether the file is in gff version 3 format

GFF 3 format:

1. adds a mechanism for representing more than one level of hierarchical grouping of features and subfeatures.
2. separates the ideas of group membership and feature name/id
3. constrains the feature type field to be taken from a controlled vocabulary.
4. allows a single feature, such as an exon, to belong to more than one group at a time.
5. provides an explicit convention for pairwise alignments
6. provides an explicit convention for features that occupy disjunct regions

The format consists of 9 columns, separated by tabs (NOT spaces).

Undefined fields are replaced with the "." character, as described in the original GFF spec.

For complete details see http://song.sourceforge.net/gff3.shtml

```
>>> fname = get_test_fname( 'test.gff' )
>>> Gff3().sniff( fname )
False
>>> fname = get_test_fname('gff_version_3.gff')
>>> Gff3().sniff( fname )
True
```

track_type = 'FeatureTrack'
Add metadata elements

valid_gff3_phase = ['.', '0', '1', '2']

valid_gff3_strand = ['+', '-', '.', '?']

class galaxy.datatypes.interval.**Gtf**(***kwd*)
Bases: *galaxy.datatypes.interval.Gff*

Tab delimited data in Gtf format

column_names = ['Seqname', 'Source', 'Feature', 'Start', 'End', 'Score', 'Strand', 'Frame', 'Attributes']

file_ext = 'gtf'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff(*filename*)
Determines whether the file is in gtf format

GTF lines have nine required fields that must be tab-separated. The first eight GTF fields are the same as GFF. The group field has been expanded into a list of attributes. Each attribute consists of a type/value pair. Attributes must end in a semi-colon, and be separated from any following attribute by exactly one space. The attribute list must begin with the two mandatory attributes:

gene_id value - A globally unique identifier for the genomic source of the sequence. transcript_id value - A globally unique identifier for the predicted transcript.

For complete details see http://genome.ucsc.edu/FAQ/FAQformat#format4

```
>>> fname = get_test_fname( '1.bed' )
>>> Gtf().sniff( fname )
False
>>> fname = get_test_fname( 'test.gff' )
>>> Gtf().sniff( fname )
False
>>> fname = get_test_fname( 'test.gtf' )
```

```
>>> Gtf().sniff( fname )
True
```

track_type = 'FeatureTrack'
Add metadata elements

class galaxy.datatypes.interval.**Interval**(***kwd*)
Bases: *galaxy.datatypes.tabular.Tabular*

Tab delimited data containing interval information

as_ucsc_display_file(*dataset*, ***kwd*)
Returns file contents with only the bed data

data_sources = {'index': 'bigwig', 'data': 'tabix'}
Add metadata elements

dataproviders = {'dataset-column': <function dataset_column_dataprovider at 0x7f497b0bd320>, 'chunk64': <functi

display_peek(*dataset*)
Returns formated html of peek

displayable(*dataset*)

file_ext = 'interval'

genomic_region_dataprovider(**args*, ***kwargs*)

genomic_region_dict_dataprovider(**args*, ***kwargs*)

get_estimated_display_viewport(*dataset*, *chrom_col=None*, *start_col=None*, *end_col=None*)
Return a chrom, start, stop tuple for viewing a file.

get_track_resolution(*dataset*, *start*, *end*)

get_track_window(*dataset*, *data*, *start*, *end*)
Assumes the incoming track data is sorted already.

init_meta(*dataset*, *copy_from=None*)

interval_dataprovider(**args*, ***kwargs*)

interval_dict_dataprovider(**args*, ***kwargs*)

line_class = 'region'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

repair_methods(*dataset*)
Return options for removing errors along with a description

set_meta(*dataset*, *overwrite=True*, *first_line_is_header=False*, ***kwd*)
Tries to guess from the line the location number of the column for the chromosome, region start-end and strand

sniff(*filename*)
Checks for 'intervalness'

This format is mostly used by galaxy itself. Valid interval files should include a valid header comment, but this seems to be loosely regulated.

```
>>> fname = get_test_fname( 'test_space.txt' )
>>> Interval().sniff( fname )
False
>>> fname = get_test_fname( 'interval.interval' )
```

```
>>> Interval().sniff( fname )
True
```

track_type = **'FeatureTrack'**

ucsc_links (*dataset*, *type*, *app*, *base_url*)
Generate links to UCSC genome browser sites based on the dbkey and content of dataset.

validate (*dataset*)
Validate an interval file using the bx GenomicIntervalReader

class galaxy.datatypes.interval.**Wiggle** (***kwd*)
Bases: *galaxy.datatypes.tabular.Tabular*, galaxy.datatypes.interval._RemoteCallMixin

Tab delimited data in wiggle format

data_sources = **{'index': 'bigwig', 'data': 'bigwig'}**

dataproviders = **{'dataset-column': <function dataset_column_dataprovider at 0x7f497b0bd320>, 'chunk64': <functi**

display_peek (*dataset*)
Returns formated html of peek

file_ext = **'wig'**

gbrowse_links (*dataset*, *type*, *app*, *base_url*)

get_estimated_display_viewport (*dataset*)
Return a chrom, start, stop tuple for viewing a file.

get_track_resolution (*dataset*, *start*, *end*)

get_track_window (*dataset*, *data*, *start*, *end*)
Assumes we have a numpy file.

metadata_spec = **dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe**

set_meta (*dataset*, *overwrite=True*, ***kwd*)

sniff (*filename*)
Determines wether the file is in wiggle format

The .wig format is line-oriented. Wiggle data is preceeded by a track definition line, which adds a number of options for controlling the default display of this track. Following the track definition line is the track data, which can be entered in several different formats.

The track definition line begins with the word 'track' followed by the track type. The track type with version is REQUIRED, and it currently must be wiggle_0. For example, track type=wiggle_0...

For complete details see http://genome.ucsc.edu/goldenPath/help/wiggle.html

```
>>> fname = get_test_fname( 'interval1.bed' )
>>> Wiggle().sniff( fname )
False
>>> fname = get_test_fname( 'wiggle.wig' )
>>> Wiggle().sniff( fname )
True
```

track_type = **'LineTrack'**

ucsc_links (*dataset*, *type*, *app*, *base_url*)

wiggle_dataprovider (**args*, ***kwargs*)

wiggle_dict_dataprovider (**args*, ***kwargs*)

metadata Module Galaxy Metadata

class galaxy.datatypes.metadata.**ColumnParameter**(*spec*)
Bases: galaxy.datatypes.metadata.RangeParameter

get_html(*value*, *context=None*, *other_values=None*, *values=None*, ***kwd*)

get_html_field(*value=None*, *context=None*, *other_values=None*, *values=None*, ***kwd*)

class galaxy.datatypes.metadata.**ColumnTypesParameter**(*spec*)
Bases: galaxy.datatypes.metadata.MetadataParameter

to_string(*value*)

class galaxy.datatypes.metadata.**DBKeyParameter**(*spec*)
Bases: galaxy.datatypes.metadata.SelectParameter

get_html(*value=None*, *context=None*, *other_values=None*, *values=None*, ***kwd*)

get_html_field(*value=None*, *context=None*, *other_values=None*, *values=None*, ***kwd*)

class galaxy.datatypes.metadata.**DictParameter**(*spec*)
Bases: galaxy.datatypes.metadata.MetadataParameter

to_safe_string(*value*)

to_string(*value*)

class galaxy.datatypes.metadata.**FileParameter**(*spec*)
Bases: galaxy.datatypes.metadata.MetadataParameter

from_external_value(*value*, *parent*, *path_rewriter=None*)
Turns a value read from a external dict into its value to be pushed directly into the metadata dict.

get_html(*value=None*, *context=None*, *other_values=None*, ***kwd*)

get_html_field(*value=None*, *context=None*, *other_values=None*, ***kwd*)

make_copy(*value*, *target_context*, *source_context*)

classmethod marshal(*value*)

new_file(*dataset=None*, ***kwds*)

to_external_value(*value*)
Turns a value read from a metadata into its value to be pushed directly into the external dict.

to_safe_string(*value*)

to_string(*value*)

wrap(*value*, *session*)

class galaxy.datatypes.metadata.**JobExternalOutputMetadataWrapper**(*job*)
Bases: object

Class with methods allowing set_meta() to be called externally to the Galaxy head. This class allows access to external metadata filenames for all outputs associated with a job. We will use JSON as the medium of exchange of information, except for the DatasetInstance object which will use pickle (in the future this could be JSONified as well)

cleanup_external_metadata(*sa_session*)

external_metadata_set_successfully(*dataset*, *sa_session*)

get_dataset_metadata_key(*dataset*)

get_output_filenames_by_dataset(*dataset*, *sa_session*)

invalidate_external_metadata(*datasets*, *sa_session*)

set_job_runner_external_pid(*pid*, *sa_session*)

setup_external_metadata(*datasets*, *sa_session*, *exec_dir=None*, *tmp_dir=None*, *dataset_files_path=None*, *output_fnames=None*, *config_root=None*, *config_file=None*, *datatypes_config=None*, *job_metadata=None*, *compute_tmp_dir=None*, *include_command=True*, *kwds=None*)

class galaxy.datatypes.metadata.**ListParameter**(*spec*)
Bases: galaxy.datatypes.metadata.MetadataParameter

to_string(*value*)

class galaxy.datatypes.metadata.**MetadataCollection**(*parent*)
Bases: object

MetadataCollection is not a collection at all, but rather a proxy to the real metadata which is stored as a Dictionary. This class handles processing the metadata elements when they are set and retrieved, returning default values in cases when metadata is not set.

element_is_set(*name*)

from_JSON_dict(*filename=None*, *path_rewriter=None*, *json_dict=None*)

get(*key*, *default=None*)

get_html_by_name(*name*, ***kwd*)

get_parent()

items()

make_dict_copy(*to_copy*)
Makes a deep copy of input iterable to_copy according to self.spec

parent

set_parent(*parent*)

spec

to_JSON_dict(*filename=None*)

galaxy.datatypes.metadata.**MetadataElement = <galaxy.datatypes.metadata.Statement object>**
MetadataParameter sub-classes.

class galaxy.datatypes.metadata.**MetadataElementSpec**(*datatype*, *name=None*, *desc=None*, *param=<class 'galaxy.datatypes.metadata.MetadataParameter'>*, *default=None*, *no_value=None*, *visible=True*, *set_in_upload=False*, ***kwargs*)
Bases: object

Defines a metadata element and adds it to the metadata_spec (which is a MetadataSpecCollection) of datatype.

get(*name*, *default=None*)

unwrap(*value*)
Turns an incoming value into its storable form.

wrap(*value*, *session*)
Turns a stored value into its usable form.

class galaxy.datatypes.metadata.**MetadataParameter**(*spec*)
Bases: object

from_external_value(*value*, *parent*)
Turns a value read from an external dict into its value to be pushed directly into the metadata dict.

get_html(*value*, *context=None*, *other_values=None*, ***kwd*)
The "context" is simply the metadata collection/bunch holding this piece of metadata. This is passed in to allow for metadata to validate against each other (note: this could turn into a huge, recursive mess if not done with care). For example, a column assignment should validate against the number of columns in the dataset.

get_html_field(*value=None*, *context=None*, *other_values=None*, ***kwd*)

make_copy(*value*, *target_context=None*, *source_context=None*)

classmethod marshal(*value*)
This method should/can be overridden to convert the incoming value to whatever type it is supposed to be.

to_external_value(*value*)
Turns a value read from a metadata into its value to be pushed directly into the external dict.

to_safe_string(*value*)

to_string(*value*)

unwrap(*form_value*)
Turns a value into its storable form.

validate(*value*)
Throw an exception if the value is invalid.

wrap(*value*, *session*)
Turns a value into its usable form.

class galaxy.datatypes.metadata.**MetadataSpecCollection**(*dict=None*)
Bases: *galaxy.util.odict.odict*

A simple extension of dict which allows cleaner access to items and allows the values to be iterated over directly as if it were a list. append() is also implemented for simplicity and does not "append".

append(*item*)

iter()

class galaxy.datatypes.metadata.**MetadataTempFile**(***kwds*)
Bases: object

classmethod cleanup_from_JSON_dict_filename(*filename*)

file_name

classmethod from_JSON(*json_dict*)

classmethod is_JSONified_value(*value*)

tmp_dir = 'database/tmp'

to_JSON()

class galaxy.datatypes.metadata.**PythonObjectParameter**(*spec*)
Bases: *galaxy.datatypes.metadata.MetadataParameter*

get_html(*value=None*, *context=None*, *other_values=None*, ***kwd*)

get_html_field(*value=None*, *context=None*, *other_values=None*, ***kwd*)

classmethod marshal(*value*)

to_string(*value*)

class galaxy.datatypes.metadata.**RangeParameter**(*spec*)
Bases: galaxy.datatypes.metadata.SelectParameter

get_html(*value*, *context=None*, *other_values=None*, *values=None*, ***kwd*)

get_html_field(*value=None*, *context=None*, *other_values=None*, *values=None*, ***kwd*)

classmethod marshal(*value*)

class galaxy.datatypes.metadata.**SelectParameter**(*spec*)
Bases: galaxy.datatypes.metadata.MetadataParameter

get_html(*value*, *context=None*, *other_values=None*, *values=None*, ***kwd*)

get_html_field(*value=None*, *context=None*, *other_values=None*, *values=None*, ***kwd*)

classmethod marshal(*value*)

to_string(*value*)

wrap(*value*, *session*)

class galaxy.datatypes.metadata.**Statement**(*target*)
Bases: object

This class inserts its target into a list in the surrounding class. the data.Data class has a metaclass which executes these statements. This is how we shove the metadata element spec into the class.

classmethod process(*element*)

ngsindex Module NGS indexes

class galaxy.datatypes.ngsindex.**BowtieBaseIndex**(***kwd*)
Bases: galaxy.datatypes.ngsindex.BowtieIndex

Bowtie base space index

file_ext = 'bowtie_base_index'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.ngsindex.**BowtieColorIndex**(***kwd*)
Bases: galaxy.datatypes.ngsindex.BowtieIndex

Bowtie color space index

file_ext = 'bowtie_color_index'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.ngsindex.**BowtieIndex**(***kwd*)
Bases: galaxy.datatypes.images.Html

base class for BowtieIndex is subclassed by BowtieColorIndex and BowtieBaseIndex

allow_datatype_change = False

composite_type = 'auto_primary_file'

display_peek(*dataset*)

file_ext = 'bowtie_index'

generate_primary_file (*dataset=None*)
This is called only at upload to write the html file cannot rename the datasets here - they come with the default unfortunately

is_binary = True

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

regenerate_primary_file (*dataset*)
cannot do this until we are setting metadata

set_peek (*dataset*, *is_multi_byte=False*)

sniff (*filename*)

qualityscore Module Qualityscore class

class galaxy.datatypes.qualityscore.**QualityScore** (***kwd*)
Bases: *galaxy.datatypes.data.Text*

until we know more about quality score formats

file_ext = 'qual'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.qualityscore.**QualityScore454** (***kwd*)
Bases: *galaxy.datatypes.qualityscore.QualityScore*

until we know more about quality score formats

file_ext = 'qual454'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff (*filename*)

```
>>> fname = get_test_fname( 'sequence.fasta' )
>>> QualityScore454().sniff( fname )
False
>>> fname = get_test_fname( 'sequence.qual454' )
>>> QualityScore454().sniff( fname )
True
```

class galaxy.datatypes.qualityscore.**QualityScoreIllumina** (***kwd*)
Bases: *galaxy.datatypes.qualityscore.QualityScore*

until we know more about quality score formats

file_ext = 'qualillumina'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.qualityscore.**QualityScoreSOLiD** (***kwd*)
Bases: *galaxy.datatypes.qualityscore.QualityScore*

until we know more about quality score formats

file_ext = 'qualsolid'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta (*dataset*, ***kwd*)

sniff (*filename*)

```
>>> fname = get_test_fname( 'sequence.fasta' )
>>> QualityScoreSOLiD().sniff( fname )
False
>>> fname = get_test_fname( 'sequence.qualsolid' )
>>> QualityScoreSOLiD().sniff( fname )
True
```

class galaxy.datatypes.qualityscore.**QualityScoreSolexa** (***kwd*)

Bases: *galaxy.datatypes.qualityscore.QualityScore*

until we know more about quality score formats

file_ext = 'qualsolexa'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

registry Module Provides mapping between extensions and datatypes, mime-types, etc.

exception galaxy.datatypes.registry.**ConfigurationError**

Bases: exceptions.Exception

class galaxy.datatypes.registry.**Registry**

Bases: object

change_datatype (*data*, *ext*)

find_conversion_destination_for_dataset_by_extensions (*dataset*, *accepted_formats*, *converter_safe=True*)

Returns (target_ext, existing converted dataset)

get_available_tracks ()

get_composite_extensions ()

get_converter_by_target_type (*source_ext*, *target_ext*)

Returns a converter based on source and target datatypes

get_converters_by_datatype (*ext*)

Returns available converters by source type

get_datatype_by_extension (*ext*)

Returns a datatype based on an extension

get_datatype_class_by_name (*name*)

Return the datatype class where the datatype's *type* attribute (as defined in the datatype_conf.xml file) contains *name*.

get_display_sites (*site_type*)

get_legacy_sites_by_build (*site_type*, *build*)

get_mimetype_by_extension (*ext*, *default='application/octet-stream'*)

Returns a mimetype based on an extension

get_upload_metadata_params (*context*, *group*, *tool*)

Returns dict of case value:inputs for metadata conditional for upload tool

integrated_datatypes_configs

load_build_sites (*root*)

load_datatype_converters(*toolbox*, *installed_repository_dict=None*, *deactivate=False*)
If deactivate is False, add datatype converters from self.converters or self.proprietary_converters to the calling app's toolbox. If deactivate is True, eliminates relevant converters from the calling app's toolbox.

load_datatype_sniffers(*root*, *deactivate=False*, *handling_proprietary_datatypes=False*, *override=False*)
Process the sniffers element from a parsed a datatypes XML file located at root_dir/config (if processing the Galaxy distributed config) or contained within an installed Tool Shed repository. If deactivate is True, an installed Tool Shed repository that includes custom sniffers is being deactivated or uninstalled, so appropriate loaded sniffers will be removed from the registry. The value of override will be False when a Tool Shed repository is being installed. Since installation is occurring after the datatypes registry has been initialized at server startup, its contents cannot be overridden by newly introduced conflicting sniffers.

load_datatypes(*root_dir=None*, *config=None*, *deactivate=False*, *override=True*)
Parse a datatypes XML file located at root_dir/config (if processing the Galaxy distributed config) or contained within an installed Tool Shed repository. If deactivate is True, an installed Tool Shed repository that includes custom datatypes is being deactivated or uninstalled, so appropriate loaded datatypes will be removed from the registry. The value of override will be False when a Tool Shed repository is being installed. Since installation is occurring after the datatypes registry has been initialized at server startup, its contents cannot be overridden by newly introduced conflicting data types.

load_display_applications(*app*, *installed_repository_dict=None*, *deactivate=False*)
If deactivate is False, add display applications from self.display_app_containers or self.proprietary_display_app_containers to appropriate datatypes. If deactivate is True, eliminates relevant display applications from appropriate datatypes.

load_external_metadata_tool(*toolbox*)
Adds a tool which is used to set external metadata

reload_display_applications(*display_application_ids=None*)
Reloads display applications: by id, or all if no ids provided Returns tuple([reloaded_ids], [failed_ids])

set_default_values()

to_xml_file()

sequence Module Sequence classes

class galaxy.datatypes.sequence.**Alignment**(***kwd*)
Bases: *galaxy.datatypes.data.Text*

Class describing an alignment

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

split(*input_datasets*, *subdir_generator_function*, *split_params*)
Split a generic alignment file (not sensible or possible, see subclasses).

class galaxy.datatypes.sequence.**Axt**(***kwd*)
Bases: *galaxy.datatypes.data.Text*

Class describing an axt alignment

file_ext = 'axt'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff(*filename*)
Determines whether the file is in axt format

axt alignment files are produced from Blastz, an alignment tool available from Webb Miller's lab at Penn State University.

Each alignment block in an axt file contains three lines: a summary line and 2 sequence lines. Blocks are separated from one another by blank lines.

The summary line contains chromosomal position and size information about the alignment. It consists of 9 required fields.

The sequence lines contain the sequence of the primary assembly (line 2) and aligning assembly (line 3) with inserts. Repeats are indicated by lower-case letters.

For complete details see http://genome.ucsc.edu/goldenPath/help/axt.html

```
>>> fname = get_test_fname( 'alignment.axt' )
>>> Axt().sniff( fname )
True
>>> fname = get_test_fname( 'alignment.lav' )
>>> Axt().sniff( fname )
False
```

class `galaxy.datatypes.sequence.`**`Fasta`**(***kwd*)

Bases: *`galaxy.datatypes.sequence.Sequence`*

Class representing a FASTA sequence

`file_ext` = 'fasta'

`metadata_spec` = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

`sniff`(*filename*)

Determines whether the file is in fasta format

A sequence in FASTA format consists of a single-line description, followed by lines of sequence data. The first character of the description line is a greater-than (">") symbol in the first column. All lines should be shorter than 80 characters

For complete details see http://www.ncbi.nlm.nih.gov/blast/fasta.shtml

Rules for sniffing as True:

We don't care about line length (other than empty lines).

The first non-empty line must start with '>' and the Very Next line.strip() must have sequence data and not be a header.

'sequence data' here is loosely defined as non-empty lines which do not start with '>'

This will cause Color Space FASTA (csfasta) to be detected as True (they are, after all, still FASTA files - they have a header line followed by sequence data)

Previously this method did some checking to determine if the sequence data had integers (presumably to differentiate between fasta and csfasta)

This should be done through sniff order, where csfasta (currently has a null sniff function) is detected for first (stricter definition) followed sometime after by fasta

We will only check that the first purported sequence is correctly formatted.

```
>>> fname = get_test_fname( 'sequence.maf' )
>>> Fasta().sniff( fname )
False
>>> fname = get_test_fname( 'sequence.fasta' )
>>> Fasta().sniff( fname )
True
```

classmethod split (*input_datasets*, *subdir_generator_function*, *split_params*)
Split a FASTA file sequence by sequence.

Note that even if split_mode="number_of_parts", the actual number of sub-files produced may not match that requested by split_size.

If split_mode="to_size" then split_size is treated as the number of FASTA records to put in each sub-file (not size in bytes).

class galaxy.datatypes.sequence.**Fastq** (***kwd*)
Bases: *galaxy.datatypes.sequence.Sequence*

Class representing a generic FASTQ sequence

file_ext = 'fastq'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

static process_split_file (*data*)
This is called in the context of an external process launched by a Task (possibly not on the Galaxy machine) to create the input files for the Task. The parameters: data - a dict containing the contents of the split file

set_meta (*dataset*, ***kwd*)
Set the number of sequences and the number of data lines in dataset. FIXME: This does not properly handle line wrapping

sniff (*filename*)
Determines whether the file is in generic fastq format For details, see http://maq.sourceforge.net/fastq.shtml

Note: There are three kinds of FASTQ files, known as "Sanger" (sometimes called "Standard"), Solexa, and Illumin
These differ in the representation of the quality scores

```
>>> fname = get_test_fname( '1.fastqsanger' )
>>> Fastq().sniff( fname )
True
>>> fname = get_test_fname( '2.fastqsanger' )
>>> Fastq().sniff( fname )
True
```

classmethod split (*input_datasets*, *subdir_generator_function*, *split_params*)
FASTQ files are split on cluster boundaries, in increments of 4 lines

class galaxy.datatypes.sequence.**FastqCSSanger** (***kwd*)
Bases: *galaxy.datatypes.sequence.Fastq*

Class representing a Color Space FASTQ sequence (e.g a SOLiD variant)

file_ext = 'fastqcssanger'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.sequence.**FastqIllumina** (***kwd*)
Bases: *galaxy.datatypes.sequence.Fastq*

Class representing a FASTQ sequence (the Illumina 1.3+ variant)

file_ext = 'fastqillumina'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.sequence.**FastqSanger** (***kwd*)
Bases: *galaxy.datatypes.sequence.Fastq*

Class representing a FASTQ sequence (the Sanger variant)

file_ext = 'fastqsanger'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.sequence.**FastqSolexa**(***kwd*)
Bases: *galaxy.datatypes.sequence.Fastq*

Class representing a FASTQ sequence (the Solexa variant)

file_ext = 'fastqsolexa'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.sequence.**Lav**(***kwd*)
Bases: *galaxy.datatypes.data.Text*

Class describing a LAV alignment

file_ext = 'lav'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff(*filename*)
Determines whether the file is in lav format

LAV is an alignment format developed by Webb Miller's group. It is the primary output format for BLASTZ. The first line of a .lav file begins with #:lav.

For complete details see http://www.bioperl.org/wiki/LAV_alignment_format

```
>>> fname = get_test_fname( 'alignment.lav' )
>>> Lav().sniff( fname )
True
>>> fname = get_test_fname( 'alignment.axt' )
>>> Lav().sniff( fname )
False
```

class galaxy.datatypes.sequence.**Maf**(***kwd*)
Bases: *galaxy.datatypes.sequence.Alignment*

Class describing a Maf alignment

display_peek(*dataset*)
Returns formated html of peek

file_ext = 'maf'

init_meta(*dataset*, *copy_from=None*)

make_html_table(*dataset*, *skipchars=[]*)
Create HTML table, used for displaying peek

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta(*dataset*, *overwrite=True*, ***kwd*)
Parses and sets species, chromosomes, index from MAF file.

set_peek(*dataset*, *is_multi_byte=False*)

sniff(*filename*)
Determines wether the file is in maf format

The .maf format is line-oriented. Each multiple alignment ends with a blank line. Each sequence in an alignment is on a single line, which can get quite long, but there is no length limit. Words in a line are

delimited by any white space. Lines starting with # are considered to be comments. Lines starting with ## can be ignored by most programs, but contain meta-data of one form or another.

The first line of a .maf file begins with ##maf. This word is followed by white-space-separated variable=value pairs. There should be no white space surrounding the "=".

For complete details see http://genome.ucsc.edu/FAQ/FAQformat#format5

```
>>> fname = get_test_fname( 'sequence.maf' )
>>> Maf().sniff( fname )
True
>>> fname = get_test_fname( 'sequence.fasta' )
>>> Maf().sniff( fname )
False
```

class galaxy.datatypes.sequence.**MafCustomTrack**(***kwd*)

Bases: *galaxy.datatypes.data.Text*

file_ext = 'mafcustomtrack'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta(*dataset*, *overwrite=True*, ***kwd*)

Parses and sets viewport metadata from MAF file.

class galaxy.datatypes.sequence.**RNADotPlotMatrix**(***kwd*)

Bases: *galaxy.datatypes.data.Data*

file_ext = 'rna_eps'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

set_peek(*dataset*, *is_multi_byte=False*)

sniff(*filename*)

Determine if the file is in RNA dot plot format.

class galaxy.datatypes.sequence.**Sequence**(***kwd*)

Bases: *galaxy.datatypes.data.Text*

Class describing a sequence

classmethod do_fast_split(*input_datasets*, *toc_file_datasets*, *subdir_generator_function*, *split_params*)

classmethod do_slow_split(*input_datasets*, *subdir_generator_function*, *split_params*)

static get_sequences_per_file(*total_sequences*, *split_params*)

static get_split_commands_sequential(*is_compressed*, *input_name*, *output_name*, *start_sequence*, *sequence_count*)

Does a brain-dead sequential scan & extract of certain sequences >>> Sequence.get_split_commands_sequential(True, './input.gz', './output.gz', start_sequence=0, sequence_count=10) ['zcat "./input.gz" | (tail -n +1 2> /dev/null) | head -40 | gzip -c > "./output.gz"'] >>> Sequence.get_split_commands_sequential(False, './input.fastq', './output.fastq', start_sequence=10, sequence_count=10) ['tail -n +41 "./input.fastq" 2> /dev/null | head -40 > "./output.fastq"']

static get_split_commands_with_toc(*input_name*, *output_name*, *toc_file*, *start_sequence*, *sequence_count*)

Uses a Table of Contents dict, parsed from an FQTOC file, to come up with a set of shell commands that will extract the parts necessary >>> three_sections=[dict(start=0, end=74, sequences=10), dict(start=74, end=148, sequences=10), dict(start=148, end=148+76, sequences=10)] >>> Sequence.get_split_commands_with_toc('./input.gz', './output.gz', dict(sections=three_sections), start_sequence=0, sequence_count=10) ['dd bs=1 skip=0 count=74 if=./input.gz 2> /dev/null

>> ./output.gz'] >>> Sequence.get_split_commands_with_toc('./input.gz', './output.gz', dict(sections=three_sections), start_sequence=1, sequence_count=5) ['(dd bs=1 skip=0 count=74 if=./input.gz 2> /dev/null)| zcat | (tail -n +5 2> /dev/null) | head -20 | gzip -c >> ./output.gz'] >>> Sequence.get_split_commands_with_toc('./input.gz', './output.gz', dict(sections=three_sections), start_sequence=0, sequence_count=20) ['dd bs=1 skip=0 count=148 if=./input.gz 2> /dev/null >> ./output.gz'] >>> Sequence.get_split_commands_with_toc('./input.gz', './output.gz', dict(sections=three_sections), start_sequence=5, sequence_count=10) ['(dd bs=1 skip=0 count=74 if=./input.gz 2> /dev/null)| zcat | (tail -n +21 2> /dev/null) | head -20 | gzip -c >> ./output.gz', '(dd bs=1 skip=74 count=74 if=./input.gz 2> /dev/null)| zcat | (tail -n +1 2> /dev/null) | head -20 | gzip -c >> ./output.gz'] >>> Sequence.get_split_commands_with_toc('./input.gz', './output.gz', dict(sections=three_sections), start_sequence=10, sequence_count=10) ['dd bs=1 skip=74 count=74 if=./input.gz 2> /dev/null >> ./output.gz'] >>> Sequence.get_split_commands_with_toc('./input.gz', './output.gz', dict(sections=three_sections), start_sequence=5, sequence_count=20) ['(dd bs=1 skip=0 count=74 if=./input.gz 2> /dev/null)| zcat | (tail -n +21 2> /dev/null) | head -20 | gzip -c >> ./output.gz', 'dd bs=1 skip=74 count=74 if=./input.gz 2> /dev/null >> ./output.gz', '(dd bs=1 skip=148 count=76 if=./input.gz 2> /dev/null)| zcat | (tail -n +1 2> /dev/null) | head -20 | gzip -c >> ./output.gz']

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta (*dataset*, ***kwd*)
Set the number of sequences and the number of data lines in dataset.

set_peek (*dataset*, *is_multi_byte=False*)

split (*input_datasets*, *subdir_generator_function*, *split_params*)
Split a generic sequence file (not sensible or possible, see subclasses).

classmethod write_split_files (*input_datasets*, *toc_file_datasets*, *subdir_generator_function*, *sequences_per_file*)

class galaxy.datatypes.sequence.**SequenceSplitLocations** (***kwd*)
Bases: *galaxy.datatypes.data.Text*

Class storing information about a sequence file composed of multiple gzip files concatenated as one OR an uncompressed file. In the GZIP case, each sub-file's location is stored in start and end.

The format of the file is JSON:

```
{ "sections" : [
        { "start" : "x", "end" : "y", "sequences" : "z" },
        ...
]}
```

file_ext = 'fqtoc'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_peek (*dataset*, *is_multi_byte=False*)

sniff (*filename*)

class galaxy.datatypes.sequence.**csFasta** (***kwd*)
Bases: *galaxy.datatypes.sequence.Sequence*

Class representing the SOLID Color-Space sequence (csfasta)

file_ext = 'csfasta'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta (*dataset*, ***kwd*)

sniff (*filename*)

Color-space sequence: >2_15_85_F3 T2130210130123030023322120121122212221122122222

```
>>> fname = get_test_fname( 'sequence.fasta' )
>>> csFasta().sniff( fname )
False
>>> fname = get_test_fname( 'sequence.csfasta' )
>>> csFasta().sniff( fname )
True
```

sniff Module File format detector

exception galaxy.datatypes.sniff.**InappropriateDatasetContentError**

Bases: exceptions.Exception

galaxy.datatypes.sniff.**check_newlines**(*fname*, *bytes_to_read=52428800*)

Determines if there are any non-POSIX newlines in the first number_of_bytes (by default, 50MB) of the file.

galaxy.datatypes.sniff.**convert_newlines**(*fname*, *in_place=True*, *tmp_dir=None*, *tmp_prefix=None*)

Converts in place a file from universal line endings to Posix line endings.

```
>>> fname = get_test_fname('temp.txt')
>>> file(fname, 'wt').write("1 2\r3 4")
>>> convert_newlines(fname, tmp_prefix="gxtest", tmp_dir=tempfile.gettempdir())
(2, None)
>>> file(fname).read()
'1 2\n3 4\n'
```

galaxy.datatypes.sniff.**convert_newlines_sep2tabs**(*fname*, *in_place=True*, *patt='\\s+'*, *tmp_dir=None*, *tmp_prefix=None*)

Combines above methods: convert_newlines() and sep2tabs() so that files do not need to be read twice

```
>>> fname = get_test_fname('temp.txt')
>>> file(fname, 'wt').write("1 2\r3 4")
>>> convert_newlines_sep2tabs(fname, tmp_prefix="gxtest", tmp_dir=tempfile.gettempdir())
(2, None)
>>> file(fname).read()
'1\t2\n3\t4\n'
```

galaxy.datatypes.sniff.**get_headers**(*fname*, *sep*, *count=60*, *is_multi_byte=False*)

Returns a list with the first 'count' lines split by 'sep'

```
>>> fname = get_test_fname('complete.bed')
>>> get_headers(fname,'\t')
[['chr7', '127475281', '127491632', 'NM_000230', '0', '+', '127486022', '127488767', '0', '3', '
```

galaxy.datatypes.sniff.**get_test_fname**(*fname*)

Returns test data filename

galaxy.datatypes.sniff.**guess_ext**(*fname*, *sniff_order=None*, *is_multi_byte=False*)

Returns an extension that can be used in the datatype factory to generate a data for the 'fname' file

```
>>> fname = get_test_fname('megablast_xml_parser_test1.blastxml')
>>> guess_ext(fname)
'xml'
>>> fname = get_test_fname('interval.interval')
>>> guess_ext(fname)
```

```
    'interval'
    >>> fname = get_test_fname('interval1.bed')
    >>> guess_ext(fname)
    'bed'
    >>> fname = get_test_fname('test_tab.bed')
    >>> guess_ext(fname)
    'bed'
    >>> fname = get_test_fname('sequence.maf')
    >>> guess_ext(fname)
    'maf'
    >>> fname = get_test_fname('sequence.fasta')
    >>> guess_ext(fname)
    'fasta'
    >>> fname = get_test_fname('file.html')
    >>> guess_ext(fname)
    'html'
    >>> fname = get_test_fname('test.gtf')
    >>> guess_ext(fname)
    'gtf'
    >>> fname = get_test_fname('test.gff')
    >>> guess_ext(fname)
    'gff'
    >>> fname = get_test_fname('gff_version_3.gff')
    >>> guess_ext(fname)
    'gff3'
    >>> fname = get_test_fname('temp.txt')
    >>> file(fname, 'wt').write("a\t2\nc\t1\nd\t0")
    >>> guess_ext(fname)
    'tabular'
    >>> fname = get_test_fname('temp.txt')
    >>> file(fname, 'wt').write("a 1 2 x\nb 3 4 y\nc 5 6 z")
    >>> guess_ext(fname)
    'txt'
    >>> fname = get_test_fname('test_tab1.tabular')
    >>> guess_ext(fname)
    'tabular'
    >>> fname = get_test_fname('alignment.lav')
    >>> guess_ext(fname)
    'lav'
    >>> fname = get_test_fname('1.sff')
    >>> guess_ext(fname)
    'sff'
    >>> fname = get_test_fname('1.bam')
    >>> guess_ext(fname)
    'bam'
    >>> fname = get_test_fname('3unsorted.bam')
    >>> guess_ext(fname)
    'bam'
```

galaxy.datatypes.sniff.**handle_compressed_file**(*filename*, *datatypes_registry*, *ext='auto'*)

galaxy.datatypes.sniff.**handle_uploaded_dataset_file**(*filename*, *datatypes_registry*, *ext='auto'*, *is_multi_byte=False*)

galaxy.datatypes.sniff.**is_column_based**(*fname*, *sep='\t'*, *skip=0*, *is_multi_byte=False*)

Checks whether the file is column based with respect to a separator (defaults to tab separator).

```
>>> fname = get_test_fname('test.gff')
>>> is_column_based(fname)
True
>>> fname = get_test_fname('test_tab.bed')
>>> is_column_based(fname)
True
>>> is_column_based(fname, sep=' ')
False
>>> fname = get_test_fname('test_space.txt')
>>> is_column_based(fname)
False
>>> is_column_based(fname, sep=' ')
True
>>> fname = get_test_fname('test_ensembl.tab')
>>> is_column_based(fname)
True
>>> fname = get_test_fname('test_tab1.tabular')
>>> is_column_based(fname, sep=' ', skip=0)
False
>>> fname = get_test_fname('test_tab1.tabular')
>>> is_column_based(fname)
True
```

galaxy.datatypes.sniff.**sep2tabs**(*fname*, *in_place=True*, *patt='\\s+'*)

Transforms in place a 'sep' separated file to a tab separated one

```
>>> fname = get_test_fname('temp.txt')
>>> file(fname, 'wt').write("1 2\n3 4\n")
>>> sep2tabs(fname)
(2, None)
>>> file(fname).read()
'1\t2\n3\t4\n'
```

galaxy.datatypes.sniff.**stream_to_file**(*stream*, *suffix=''*, *prefix=''*, *dir=None*, *text=False*, ***kwd*)

Writes a stream to a temporary file, returns the temporary file's name

galaxy.datatypes.sniff.**stream_to_open_named_file**(*stream*, *fd*, *filename*, *source_encoding=None*, *source_error='strict'*, *target_encoding=None*, *target_error='strict'*)

Writes a stream to the provided file descriptor, returns the file's name and bool(is_multi_byte). Closes file descriptor

tabular Module Tabular datatype

class galaxy.datatypes.tabular.**Eland**(***kwd*)

Bases: *galaxy.datatypes.tabular.Tabular*

Support for the export.txt.gz file used by Illumina's ELANDv2e aligner

file_ext = '_export.txt.gz'

make_html_table(*dataset*, *skipchars=None*)

Create HTML table, used for displaying peek

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta (*dataset*, *overwrite=True*, *skip=None*, *max_data_lines=5*, ***kwd*)

sniff (*filename*)

Determines whether the file is in ELAND export format

A file in ELAND export format consists of lines of tab-separated data. There is no header.

Rules for sniffing as True:

```
- There must be 22 columns on each line
- LANE, TILEm X, Y, INDEX, READ_NO, SEQ, QUAL, POSITION, *STRAND, FILT must be correct
- We will only check that up to the first 5 alignments are correctly formatted.
```

class galaxy.datatypes.tabular.**ElandMulti** (***kwd*)

Bases: *galaxy.datatypes.tabular.Tabular*

file_ext = 'elandmulti'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

sniff (*filename*)

class galaxy.datatypes.tabular.**FeatureLocationIndex** (***kwd*)

Bases: *galaxy.datatypes.tabular.Tabular*

An index that stores feature locations in tabular format.

file_ext = 'fli'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.tabular.**Pileup** (***kwd*)

Bases: *galaxy.datatypes.tabular.Tabular*

Tab delimited data in pileup (6- or 10-column) format

data_sources = {'data': 'tabix'}

Add metadata elements

dataproviders = {'dataset-column': <function dataset_column_dataprovider at 0x7f497b0bd320>, 'chunk64': <functi

display_peek (*dataset*)

Returns formated html of peek

file_ext = 'pileup'

genomic_region_dataprovider (**args*, ***kwargs*)

genomic_region_dict_dataprovider (**args*, ***kwargs*)

init_meta (*dataset*, *copy_from=None*)

line_class = 'genomic coordinate'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

repair_methods (*dataset*)

Return options for removing errors along with a description

sniff (*filename*)

Checks for 'pileup-ness'

There are two main types of pileup: 6-column and 10-column. For both, the first three and last two columns are the same. We only check the first three to allow for some personalization of the format.

```
>>> fname = get_test_fname( 'interval.interval' )
>>> Pileup().sniff( fname )
False
>>> fname = get_test_fname( '6col.pileup' )
>>> Pileup().sniff( fname )
True
>>> fname = get_test_fname( '10col.pileup' )
>>> Pileup().sniff( fname )
True
```

class galaxy.datatypes.tabular.**Sam**(***kwd*)

Bases: *galaxy.datatypes.tabular.Tabular*

column_dataprovider(**args*, ***kwargs*)

data_sources = {'index': 'bigwig', 'data': 'bam'}

dataproviders = {'dataset-column': <function dataset_column_dataprovider at 0x7f497b0bded8>, 'chunk64': <functi

dataset_column_dataprovider(**args*, ***kwargs*)

dataset_dict_dataprovider(**args*, ***kwargs*)

dict_dataprovider(**args*, ***kwargs*)

display_peek(*dataset*)

Returns formated html of peek

file_ext = 'sam'

genomic_region_dataprovider(**args*, ***kwargs*)

genomic_region_dict_dataprovider(**args*, ***kwargs*)

header_dataprovider(**args*, ***kwargs*)

id_seq_qual_dataprovider(**args*, ***kwargs*)

line_dataprovider(**args*, ***kwargs*)

static **merge**(*split_files*, *output_file*)

Multiple SAM files may each have headers. Since the headers should all be the same, remove the headers from files 1-n, keeping them in the first file only

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

regex_line_dataprovider(**args*, ***kwargs*)

set_meta(*dataset*, *overwrite=True*, *skip=None*, *max_data_lines=5*, ***kwd*)

sniff(*filename*)

Determines whether the file is in SAM format

A file in SAM format consists of lines of tab-separated data. The following header line may be the first line:

```
@QNAME  FLAG    RNAME   POS     MAPQ    CIGAR   MRNM    MPOS    ISIZE   SEQ     QUAL
or
@QNAME  FLAG    RNAME   POS     MAPQ    CIGAR   MRNM    MPOS    ISIZE   SEQ     QUAL    OPT
```

Data in the OPT column is optional and can consist of tab-separated data

For complete details see http://samtools.sourceforge.net/SAM1.pdf

Rules for sniffing as True:

```
There must be 11 or more columns of data on each line
Columns 2 (FLAG), 4(POS), 5 (MAPQ), 8 (MPOS), and 9 (ISIZE) must be numbers (9 can be negat
We will only check that up to the first 5 alignments are correctly formatted.
```

```
>>> fname = get_test_fname( 'sequence.maf' )
>>> Sam().sniff( fname )
False
>>> fname = get_test_fname( '1.sam' )
>>> Sam().sniff( fname )
True
```

track_type = 'ReadTrack'

class galaxy.datatypes.tabular.**Tabular**(***kwd*)

Bases: *galaxy.datatypes.data.Text*

Tab delimited data

CHUNKABLE = True

Add metadata elements

as_gbrowse_display_file(*dataset*, ***kwd*)

as_ucsc_display_file(*dataset*, ***kwd*)

column_dataprovider(**args*, ***kwargs*)

Uses column settings that are passed in

dataproviders = {'dataset-column': <function dataset_column_dataprovider at 0x7f497b0bd320>, 'chunk64': <functi

dataset_column_dataprovider(**args*, ***kwargs*)

Attempts to get column settings from dataset.metadata

dataset_dict_dataprovider(**args*, ***kwargs*)

Attempts to get column settings from dataset.metadata

dict_dataprovider(**args*, ***kwargs*)

Uses column settings that are passed in

display_data(*trans*, *dataset*, *preview=False*, *filename=None*, *to_ext=None*, *chunk=None*, ***kwd*)

display_peek(*dataset*)

Returns formatted html of peek

displayable(*dataset*)

get_chunk(*trans*, *dataset*, *chunk*)

make_html_peek_header(*dataset*, *skipchars=None*, *column_names=None*, *column_number_format='%s'*, *column_parameter_alias=None*, ***kwargs*)

make_html_peek_rows(*dataset*, *skipchars=None*, ***kwargs*)

make_html_table(*dataset*, ***kwargs*)

Create HTML table, used for displaying peek

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta(*dataset*, *overwrite=True*, *skip=None*, *max_data_lines=100000*, *max_guess_type_data_lines=None*, ***kwd*)

Tries to determine the number of columns as well as those columns that contain numerical values in the dataset. A skip parameter is used because various tabular data types reuse this function, and their data type classes are responsible to determine how many invalid comment lines should be skipped. Using None for

skip will cause skip to be zero, but the first line will be processed as a header. A max_data_lines parameter is used because various tabular data types reuse this function, and their data type classes are responsible to determine how many data lines should be processed to ensure that the non-optional metadata parameters are properly set; if used, optional metadata parameters will be set to None, unless the entire file has already been read. Using None for max_data_lines will process all data lines.

Items of interest:

1. We treat 'overwrite' as always True (we always want to set tabular metadata when called).
2. If a tabular file has no data, it will have one column of type 'str'.
3. We used to check only the first 100 lines when setting metadata and this class's set_peek() method read the entire file to determine the number of lines in the file. Since metadata can now be processed on cluster nodes, we've merged the line count portion of the set_peek() processing here, and we now check the entire contents of the file.

set_peek (*dataset*, *line_count=None*, *is_multi_byte=False*)

class galaxy.datatypes.tabular.**Taxonomy** (***kwd*)

Bases: galaxy.datatypes.tabular.Tabular

display_peek (*dataset*)

Returns formated html of peek

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

class galaxy.datatypes.tabular.**Vcf** (***kwd*)

Bases: galaxy.datatypes.tabular.Tabular

Variant Call Format for describing SNPs and other simple genome variations.

column_names = ['Chrom', 'Pos', 'ID', 'Ref', 'Alt', 'Qual', 'Filter', 'Info', 'Format', 'data']

data_sources = {'index': 'bigwig', 'data': 'tabix'}

dataproviders = {'dataset-column': <function dataset_column_dataprovider at 0x7f497b0bd320>, 'chunk64': <functi

display_peek (*dataset*)

Returns formated html of peek

file_ext = 'vcf'

genomic_region_dataprovider (**args*, ***kwargs*)

genomic_region_dict_dataprovider (**args*, ***kwargs*)

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_meta (*dataset*, ***kwd*)

sniff (*filename*)

track_type = 'VariantTrack'

tracks Module Datatype classes for tracks/track views within galaxy.

class galaxy.datatypes.tracks.**GeneTrack** (***kwargs*)

Bases: galaxy.datatypes.binary.Binary

file_ext = 'genetrack'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?'

xml Module XML format classes

class galaxy.datatypes.xml.**CisML**(**kwd)
Bases: galaxy.datatypes.xml.GenericXml

CisML XML data

file_ext = 'cisml'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_peek(*dataset*, *is_multi_byte=False*)
Set the peek and blurb text

sniff(*filename*)

class galaxy.datatypes.xml.**GenericXml**(**kwd)
Bases: galaxy.datatypes.data.Text

Base format class for any XML file.

dataproviders = {'xml': <function xml_dataprovider at 0x7f497a42f578>, 'chunk64': <function chunk64_dataprovide

file_ext = 'xml'

static merge(*split_files*, *output_file*)
Merging multiple XML files is non-trivial and must be done in subclasses.

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_peek(*dataset*, *is_multi_byte=False*)
Set the peek and blurb text

sniff(*filename*)
Determines whether the file is XML or not

```
>>> fname = get_test_fname( 'megablast_xml_parser_test1.blastxml' )
>>> GenericXml().sniff( fname )
True
>>> fname = get_test_fname( 'interval.interval' )
>>> GenericXml().sniff( fname )
False
```

xml_dataprovider(**args*, ***kwargs*)

class galaxy.datatypes.xml.**MEMEXml**(**kwd)
Bases: galaxy.datatypes.xml.GenericXml

MEME XML Output data

file_ext = 'memexml'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_peek(*dataset*, *is_multi_byte=False*)
Set the peek and blurb text

sniff(*filename*)

class galaxy.datatypes.xml.**Owl**(**kwd)
Bases: galaxy.datatypes.xml.GenericXml

Web Ontology Language OWL format description http://www.w3.org/TR/owl-ref/

file_ext = 'owl'

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_peek (*dataset*, *is_multi_byte=False*)

sniff (*filename*)
Checking for keyword - '<owl' in the first 200 lines.

class galaxy.datatypes.xml.**Phyloxml** (**kwd*)
Bases: *galaxy.datatypes.xml.GenericXml*

Format for defining phyloxml data http://www.phyloxml.org/

file_ext = 'phyloxml'

get_visualizations (*dataset*)
Returns a list of visualizations for datatype.

metadata_spec = dbkey (DBKeyParameter): Database/Build, defaults to '?', data_lines (MetadataParameter): Numbe

set_peek (*dataset*, *is_multi_byte=False*)
Set the peek and blurb text

sniff (*filename*)
"Checking for keyword - 'phyloxml' always in lowercase in the first few lines

Subpackages

converters Package

bed_to_genetrack_converter Module

bed_to_gff_converter Module

bedgraph_to_array_tree_converter Module

bgzip Module

Uses pysam to bgzip a file

usage: %prog in_file out_file

galaxy.datatypes.converters.bgzip.**main**()

fasta_to_len Module

Input: fasta, int Output: tabular Return titles with lengths of corresponding seq

galaxy.datatypes.converters.fasta_to_len.**compute_fasta_length** (*fasta_file*, *out_file*, *keep_first_char*, *keep_first_word=False*)

fasta_to_tabular_converter Module

Input: fasta Output: tabular

fastq_to_fqtoc Module

galaxy.datatypes.converters.fastq_to_fqtoc.**main**()
The format of the file is JSON:

```
{ "sections" : [
        { "start" : "x", "end" : "y", "sequences" : "z" },
        ...
]}
```

This works only for UNCOMPRESSED fastq files. The Python GzipFile does not provide seekable offsets via tell(), so clients just have to split the slow way

fastqsolexa_to_fasta_converter Module convert fastqsolexa file to separated sequence and quality files.

assume each sequence and quality score are contained in one line the order should be: 1st line: @title_of_seq 2nd line: nucleotides 3rd line: +title_of_qualityscore (might be skipped) 4th line: quality scores (in three forms: a. digits, b. ASCII codes, the first char as the coding base, c. ASCII codes without the first char.)

Usage: %python fastqsolexa_to_fasta_converter.py <your_fastqsolexa_filename> <output_seq_filename> <output_score_filename>

galaxy.datatypes.converters.fastqsolexa_to_fasta_converter.**stop_err**(*msg*)

fastqsolexa_to_qual_converter Module convert fastqsolexa file to separated sequence and quality files.

assume each sequence and quality score are contained in one line the order should be: 1st line: @title_of_seq 2nd line: nucleotides 3rd line: +title_of_qualityscore (might be skipped) 4th line: quality scores (in three forms: a. digits, b. ASCII codes, the first char as the coding base, c. ASCII codes without the first char.)

Usage: %python fastqsolexa_to_qual_converter.py <your_fastqsolexa_filename> <output_seq_filename> <output_score_filename>

galaxy.datatypes.converters.fastqsolexa_to_qual_converter.**stop_err**(*msg*)

gff_to_bed_converter Module

gff_to_interval_index_converter Module Convert from GFF file to interval index file.

usage: python gff_to_interval_index_converter.py [input] [output]

galaxy.datatypes.converters.gff_to_interval_index_converter.**main**()

interval_to_bed_converter Module

galaxy.datatypes.converters.interval_to_bed_converter.**stop_err**(*msg*)

interval_to_bedstrict_converter Module

galaxy.datatypes.converters.interval_to_bedstrict_converter.**force_bed_field_count**(*fields*, *region_count*, *force_num_c*

galaxy.datatypes.converters.interval_to_bedstrict_converter.**stop_err**(*msg*)

interval_to_coverage Module Converter to generate 3 (or 4) column base-pair coverage from an interval file.

usage: %prog bed_file out_file -1, –cols1=N,N,N,N: Columns for chrom, start, end, strand in interval file -2, –cols2=N,N,N,N: Columns for chrom, start, end, strand in coverage file

class galaxy.datatypes.converters.interval_to_coverage.**CoverageWriter**(*out_stream=None*, *chromCol=0*, *positionCol=1*, *forwardCol=2*, *reverseCol=3*)

Bases: object

close()

write(***kwargs*)

galaxy.datatypes.converters.interval_to_coverage.**main**(*interval*, *coverage*)

Uses a sliding window of partitions to count coverages. Every interval record adds its start and end to the partitions. The result is a list of partitions, or every position that has a (maybe) different number of basepairs covered. We don't worry about merging because we pop as the sorted intervals are read in. As the input start positions exceed the partition positions in partitions, coverages are kicked out in bulk.

interval_to_fli Module Creates a feature location index (FLI) for a given BED/GFF file. FLI index has the form:

```
[line_length]
<symbol1_in_lowercase><tab><symbol1><tab><location>
<symbol2_in_lowercase><tab><symbol2><tab><location>
...
```

where location is formatted as:

contig:start-end

and symbols are sorted in lexigraphical order.

galaxy.datatypes.converters.interval_to_fli.**main**()

interval_to_interval_index_converter Module Convert from interval file to interval index file.

usage: %prog <options> in_file out_file -c, –chr-col: chromosome column, default=1 -s, –start-col: start column, default=2 -e, –end-col: end column, default=3

galaxy.datatypes.converters.interval_to_interval_index_converter.**main**()

interval_to_summary_tree_converter Module

interval_to_tabix_converter Module Uses pysam to index a bgzipped interval file with tabix Supported presets: bed, gff, vcf

usage: %prog in_file out_file

galaxy.datatypes.converters.interval_to_tabix_converter.**main**()

lped_to_fped_converter Module

galaxy.datatypes.converters.lped_to_fped_converter.**main**()

call fbater need to work with rgenetics composite datatypes so in and out are html files with data in extrafiles path <command interpreter="python">rg_convert_lped_fped.py '$input1/$input1.metadata.base_name' '$output1' '$output1.extra_files_path' </command>

galaxy.datatypes.converters.lped_to_fped_converter.**rgConv**(*inpedfilepath*, *outhtmlname*, *outfilepath*)

convert linkage ped/map to fbat

galaxy.datatypes.converters.lped_to_fped_converter.**timenow**()

return current time as a string

lped_to_pbed_converter Module

galaxy.datatypes.converters.lped_to_pbed_converter.**getMissval**(*inped=''*)

read some lines...ugly hack - try to guess missing value should be N or 0 but might be . or -

galaxy.datatypes.converters.lped_to_pbed_converter.**main**()

need to work with rgenetics composite datatypes so in and out are html files with data in extrafiles path <command interpreter="python">lped_to_pbed_converter.py '$input1/$input1.metadata.base_name' '$output1' '$output1.extra_files_path' '${GALAXY_DATA_INDEX_DIR}/rg/bin/plink' </command>

galaxy.datatypes.converters.lped_to_pbed_converter.**rgConv**(*inpedfilepath*, *outhtmlname*, *outfilepath*, *plink*)

galaxy.datatypes.converters.lped_to_pbed_converter.**timenow**()

return current time as a string

maf_to_fasta_converter Module

maf_to_interval_converter Module

pbed_ldreduced_converter Module

galaxy.datatypes.converters.pbed_ldreduced_converter.**main**()

need to work with rgenetics composite datatypes so in and out are html files with data in extrafiles path

galaxy.datatypes.converters.pbed_ldreduced_converter.**makeLDreduced**(*basename*, *infpath=None*, *outfpath=None*, *plinke='plink'*, *forcerebuild=False*, *returnFname=False*, *winsize='60'*, *winmove='40'*, *r2thresh='0.1'*)

not there so make and leave in output dir for post job hook to copy back into input extra files path for next time

galaxy.datatypes.converters.pbed_ldreduced_converter.**pruneLD**(*plinktasks=[]*, *cd='./'*, *vclbase=[]*)

galaxy.datatypes.converters.pbed_ldreduced_converter.**timenow**()
return current time as a string

pbed_to_lped_converter Module

galaxy.datatypes.converters.pbed_to_lped_converter.**main**()
need to work with rgenetics composite datatypes so in and out are html files with data in extrafiles path <command interpreter="python">pbed_to_lped_converter.py '$input1/$input1.metadata.base_name' '$output1' '$output1.extra_files_path' '${GALAXY_DATA_INDEX_DIR}/rg/bin/plink' </command>

galaxy.datatypes.converters.pbed_to_lped_converter.**rgConv**(*inpedfilepath*, *outhtmlname*, *outfilepath*, *plink*)

galaxy.datatypes.converters.pbed_to_lped_converter.**timenow**()
return current time as a string

picard_interval_list_to_bed6_converter Module

sam_or_bam_to_summary_tree_converter Module

sam_to_bam Module A wrapper script for converting SAM to BAM, with sorting. %prog input_filename.sam output_filename.bam

galaxy.datatypes.converters.sam_to_bam.**cleanup_before_exit**(*tmp_dir*)

vcf_to_interval_index_converter Module Convert from VCF file to interval index file.

galaxy.datatypes.converters.vcf_to_interval_index_converter.**main**()

vcf_to_summary_tree_converter Module

vcf_to_vcf_bgzip Module Uses pysam to bgzip a vcf file as-is. Headers, which are important, are kept. Original ordering, which may be specifically needed by tools or external display applications, is also maintained.

usage: %prog in_file out_file

galaxy.datatypes.converters.vcf_to_vcf_bgzip.**main**()

wiggle_to_array_tree_converter Module

galaxy.datatypes.converters.wiggle_to_array_tree_converter.**main**()

wiggle_to_simple_converter Module Read a wiggle track and print out a series of lines containing "chrom position score". Ignores track lines, handles bed, variableStep and fixedStep wiggle lines.

galaxy.datatypes.converters.wiggle_to_simple_converter.**main**()

galaxy.datatypes.converters.wiggle_to_simple_converter.**stop_err**(*msg*)

display_applications Package

application Module

class galaxy.datatypes.display_applications.application.**DisplayApplication**(*display_id*, *name*, *app*, *version=None*, *filename=None*, *elem=None*)

Bases: object

add_data_table_watch(*table_name*, *version=None*)

filter_by_dataset(*data*, *trans*)

classmethod from_elem(*elem*, *app*, *filename=None*)

classmethod from_file(*filename*, *app*)

get_link(*link_name*, *data*, *dataset_hash*, *user_hash*, *trans*, *app_kwds*)

reload()

class galaxy.datatypes.display_applications.application.**DisplayApplicationLink**(*display_application*)

Bases: object

build_parameter_dict(*data*, *dataset_hash*, *user_hash*, *trans*, *app_kwds*)

filter_by_dataset(*data*, *trans*)

classmethod from_elem(*elem*, *display_application*, *other_values=None*)

get_display_url(*data*, *trans*)

get_inital_values(*data*, *trans*)

class galaxy.datatypes.display_applications.application.**DynamicDisplayApplicationBuilder**(*elem*, *display_*, *build_*

Bases: object

class galaxy.datatypes.display_applications.application.**PopulatedDisplayApplicationLink**(*display*, *data*, *dataset*, *user_h*, *trans*, *app_kw*

Bases: object

display_ready()

display_url()

get_param_name_by_url(*url*)

get_param_value(*name*)

get_prepare_steps(*datasets_only=True*)

prepare_display()

preparing_display()

parameters Module

class galaxy.datatypes.display_applications.parameters.**DisplayApplicationDataParameter**(*elem*, *link*)

Bases: galaxy.datatypes.display_applications.parameters.DisplayApplicationParameter

Parameter that returns a file_name containing the requested content

formats

get_value(*other_values*, *dataset_hash*, *user_hash*, *trans*)

is_preparing(*other_values*)

prepare(*other_values*, *dataset_hash*, *user_hash*, *trans*)

ready(*other_values*)

type = 'data'

class galaxy.datatypes.display_applications.parameters.**DisplayApplicationParameter**(*elem*, *link*)

Bases: object

Abstract Class for Display Application Parameters

build_url(*other_values*)

classmethod from_elem(*elem*, *link*)

get_value(*other_values*, *dataset_hash*, *user_hash*, *trans*)

is_preparing(*other_values*)

prepare(*other_values*, *dataset_hash*, *user_hash*, *trans*)

ready(*other_values*)

type = None

class galaxy.datatypes.display_applications.parameters.**DisplayApplicationTemplateParameter**(*ele*
lin

Bases: galaxy.datatypes.display_applications.parameters.DisplayApplicationParameter

Parameter that returns a string containing the requested content

get_value(*other_values*, *dataset_hash*, *user_hash*, *trans*)

type = 'template'

class galaxy.datatypes.display_applications.parameters.**DisplayDataValueWrapper**(*value*, *parameter*, *other_values*, *dataset_hash*, *user_hash*, *trans*)

Bases: galaxy.datatypes.display_applications.parameters.DisplayParameterValueWrapper

ACTION_NAME = 'data'

action_name

mime_type()

qp

class galaxy.datatypes.display_applications.parameters.**DisplayParameterValueWrapper**(*value*, *parameter*, *other_values*, *dataset_hash*, *user_hash*, *trans*)

Bases: object

ACTION_NAME = 'param'

action_name

mime_type()

qp

url

util Module

galaxy.datatypes.display_applications.util.**decode_dataset_user**(*trans*, *dataset_hash*, *user_hash*)

galaxy.datatypes.display_applications.util.**encode_dataset_user**(*trans*, *dataset*, *user*)

util Package

util Package Utilities for Galaxy datatypes.

gff_util Module Provides utilities for working with GFF files.

class galaxy.datatypes.util.gff_util.**GFFFeature**(*reader*, *chrom_col=0*, *feature_col=2*, *start_col=3*, *end_col=4*, *strand_col=6*, *score_col=5*, *default_strand='.'*, *fix_strand=False*, *intervals=[]*, *raw_size=0*)

Bases: *galaxy.datatypes.util.gff_util.GFFInterval*

A GFF feature, which can include multiple intervals.

copy()

lines()

name()

Returns feature's name.

class galaxy.datatypes.util.gff_util.**GFFInterval**(*reader*, *fields*, *chrom_col=0*, *feature_col=2*, *start_col=3*, *end_col=4*, *strand_col=6*, *score_col=5*, *default_strand='.'*, *fix_strand=False*)

Bases: bx.intervals.io.GenomicInterval

A GFF interval, including attributes. If file is strictly a GFF file, only attribute is 'group.'

copy()

class galaxy.datatypes.util.gff_util.**GFFIntervalToBEDReaderWrapper**(*reader*, ***kwargs*)

Bases: bx.intervals.io.NiceReaderWrapper

Reader wrapper that reads GFF intervals/lines and automatically converts them to BED format.

parse_row(*line*)

class galaxy.datatypes.util.gff_util.**GFFReaderWrapper**(*reader*, *chrom_col=0*, *feature_col=2*, *start_col=3*, *end_col=4*, *strand_col=6*, *score_col=5*, *fix_strand=False*, *convert_to_bed_coord=False*, ***kwargs*)

Bases: bx.intervals.io.NiceReaderWrapper

Reader wrapper for GFF files.

Wrapper has two major functions:

1. group entries for GFF file (via group column), GFF3 (via id attribute), or GTF (via gene_id/transcript id);
2. convert coordinates from GFF format–starting and ending coordinates are 1-based, closed–to the 'traditional'/BED interval format–0 based, half-open. This is useful when using GFF files as inputs to tools that expect traditional interval format.

next()

Returns next GFFFeature.

parse_row(*line*)

galaxy.datatypes.util.gff_util.**convert_bed_coords_to_gff**(*interval*)

Converts an interval object's coordinates from BED format to GFF format. Accepted object types include GenomicInterval and list (where the first element in the list is the interval's start, and the second element is the interval's end).

galaxy.datatypes.util.gff_util.**convert_gff_coords_to_bed**(*interval*)

Converts an interval object's coordinates from GFF format to BED format. Accepted object types include GFFFeature, GenomicInterval, and list (where the first element in the list is the interval's start, and the second element is the interval's end).

galaxy.datatypes.util.gff_util.**gff_attributes_to_str**(*attrs*, *gff_format*)

Convert GFF attributes to string. Supported formats are GFF3, GTF.

galaxy.datatypes.util.gff_util.**parse_gff_attributes**(*attr_str*)

Parses a GFF/GTF attribute string and returns a dictionary of name-value pairs. The general format for a GFF3 attributes string is

name1=value1;name2=value2

The general format for a GTF attribute string is

name1 "value1" ; name2 "value2"

The general format for a GFF attribute string is a single string that denotes the interval's group; in this case, method returns a dictionary with a single key-value pair, and key name is 'group'

galaxy.datatypes.util.gff_util.**read_unordered_gtf**(*iterator*, *strict=False*)

Returns GTF features found in an iterator. GTF lines need not be ordered or clustered for reader to work. Reader returns GFFFeature objects sorted by transcript_id, chrom, and start position.

image_util Module Provides utilities for working with image files.

galaxy.datatypes.util.image_util.**check_image_type**(*filename*, *types*, *image=None*)

galaxy.datatypes.util.image_util.**get_image_ext**(*file_path*, *image*)

galaxy.datatypes.util.image_util.**image_type**(*filename*, *image=None*)

eggs Package

eggs Package Manage Galaxy eggs

class galaxy.eggs.**CaseSensitiveConfigParser**(*defaults=None*, *dict_type=<class 'collections.OrderedDict'>*, *allow_no_value=False*)

Bases: ConfigParser.SafeConfigParser

optionxform(*optionstr*)

class galaxy.eggs.**Crate**(*galaxy_config_file=None*, *platform=None*)

Bases: object

Reads the eggs.ini file for use with checking and fetching.

all_eggs
Return a list of all eggs in the crate.

all_missing
Return true if any eggs in the eggs config file are missing.

all_names
Return a list of names of all eggs in the crate.

config_eggs
Return a list of all eggs in the crate that are needed based on the options set in the Galaxy config file.

config_file = '/var/build/user_builds/jmchilton-galaxy/checkouts/latest/eggs.ini'

config_missing
Return true if any eggs are missing, conditional on options set in the Galaxy config file.

config_names
Return a list of names of all eggs in the crate that are needed based on the options set in the Galaxy config file.

parse()

parse_egg_section(*eggs*, *tags*, *full_platform=False*, *egg_class=<class 'galaxy.eggs.Egg'>*)

resolve(*all=False*)
Try to resolve (e.g. fetch) all eggs in the crate.

class galaxy.eggs.**Egg**(*name=None*, *version=None*, *tag=None*, *url=None*, *platform=None*, *crate=None*)

Bases: object

Contains information about locating and downloading eggs.

fetch() *serves as the install method to pkg_resources.working_set.resolve()*

path
Return the path of the egg, if it exists, or None

remove_doppelgangers()

require()

resolve()

set_dir()

set_distribution()
Stores a pkg_resources Distribution object for reference later

unpack_if_needed()

version_conflict(*conflict_dist*, *conflict_req*)

exception galaxy.eggs.**EggNotFetchable**(*eggs*)
Bases: exceptions.Exception

class galaxy.eggs.**GalaxyConfig**(*config_file*)
Bases: object

always_conditional = ('pysam', 'ctypes', 'python_daemon')

check_conditional(*egg_name*)

class galaxy.eggs.**URLRetriever**(**args*, ***kwargs*)
Bases: urllib.FancyURLopener

http_error_default(**args*)

galaxy.eggs.**get_env**()

galaxy.eggs.**remove_file_or_path**(*f*)

galaxy.eggs.**require**(*req_str*)

galaxy.eggs.**string_as_bool**(*string*)

galaxy.eggs.**unpack_zipfile**(*filename*, *extract_dir*, *ignores=[]*)

dist Module Manage Galaxy eggs

class galaxy.eggs.dist.**DistScrambleCrate**(*galaxy_config_file*, *build_on='all'*)
Bases: *galaxy.eggs.scramble.ScrambleCrate*

Holds eggs with info on how to build them for distribution.

dist_config_file = '/var/build/user_builds/jmchilton-galaxy/checkouts/latest/dist-eggs.ini'

get_platforms(*wanted*)

parse()

parse_egg_section(*eggs*, *tags*, *full_platform=False*)

class galaxy.eggs.dist.**DistScrambleEgg**(**args*, ***kwargs*)
Bases: *galaxy.eggs.scramble.ScrambleEgg*

path

run_scramble_script()

set_dir()

unpack_if_needed()

scramble Module Manage Galaxy eggs

class galaxy.eggs.scramble.**ScrambleCrate**(*galaxy_config_file=None*, *platform=None*)
Bases: galaxy.eggs.__init__.Crate

Reads the eggs.ini file for use with scrambling eggs.

parse()

parse_egg_section(**args*, ***kwargs*)

scramble(*all=False*)

class galaxy.eggs.scramble.**ScrambleEgg**(**args*, ***kwargs*)

Bases: galaxy.eggs.__init__.Egg

Contains information about scrambling eggs.

archive_dir = '/var/build/user_builds/jmchilton-galaxy/checkouts/latest/scripts/scramble/archives'

build_dir = '/var/build/user_builds/jmchilton-galaxy/checkouts/latest/scripts/scramble/build'

copy_build_script()

ez_setup = '/var/build/user_builds/jmchilton-galaxy/checkouts/latest/scripts/scramble/lib/ez_setup.py'

ez_setup_url = 'http://peak.telecommunity.com/dist/ez_setup.py'

fetch_one(*urls*)

Fetches the first available archive out of a list.

fetch_source()

Get egg (and dependent) source

get_tld(*names*)

run_scramble_script()

scramble()

scramble_dir = '/var/build/user_builds/jmchilton-galaxy/checkouts/latest/scripts/scramble'

script_dir = '/var/build/user_builds/jmchilton-galaxy/checkouts/latest/scripts/scramble/scripts'

unpack_source()

unpack_tar()

unpack_zip()

exception galaxy.eggs.scramble.**ScrambleFailure**(*eggs*, *msg=None*)

Bases: exceptions.Exception

exceptions Package

exceptions Package Custom exceptions for Galaxy

exception galaxy.exceptions.**ActionInputError**(*err_msg*, *type='error'*)

Bases: *galaxy.exceptions.MessageException*

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

exception galaxy.exceptions.**AdminRequiredException**(*err_msg=None*, *type='info'*, ***extra_error_info*)

Bases: *galaxy.exceptions.MessageException*

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 403

exception galaxy.exceptions.**AuthenticationFailed**(*err_msg=None*, *type='info'*, ***extra_error_info*)

Bases: *galaxy.exceptions.MessageException*

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 401

exception galaxy.exceptions.**AuthenticationRequired**(*err_msg=None*, *type='info'*, ***extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 403

exception galaxy.exceptions.**ConfigDoesNotAllowException**(*err_msg=None*, *type='info'*, ***extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 403

exception galaxy.exceptions.**ConfigurationError**

Bases: exceptions.Exception

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 500

exception galaxy.exceptions.**Conflict**(*err_msg=None*, *type='info'*, ***extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 409

exception galaxy.exceptions.**DeprecatedMethod**(*err_msg=None*, *type='info'*, ***extra_error_info*)

Bases: galaxy.exceptions.MessageException

Method (or a particular form/arg signature) has been removed and won't be available later

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 404

exception galaxy.exceptions.**DuplicatedIdentifierException**(*err_msg=None*, *type='info'*, ***extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

exception galaxy.exceptions.**DuplicatedSlugException**(*err_msg=None*, *type='info'*, ***extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

exception galaxy.exceptions.**InconsistentDatabase**(*err_msg=None*, *type='info'*, ***extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 500

exception galaxy.exceptions.**InsufficientPermissionsException**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 403

exception galaxy.exceptions.**InternalServerError**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 500

exception galaxy.exceptions.**ItemAccessibilityException**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 403

exception galaxy.exceptions.**ItemDeletionException**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

exception galaxy.exceptions.**ItemOwnershipException**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 403

exception galaxy.exceptions.**MalformedId**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

exception galaxy.exceptions.**MessageException**(*err_msg=None, type='info', **extra_error_info*)

Bases: exceptions.Exception

Exception to make throwing errors from deep in controllers easier.

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

exception galaxy.exceptions.**NotImplemented**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 501

exception galaxy.exceptions.**ObjectAttributeInvalidException**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

exception galaxy.exceptions.**ObjectAttributeMissingException**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

exception galaxy.exceptions.**ObjectInvalid**

Bases: exceptions.Exception

Accessed object store ID is invalid

exception galaxy.exceptions.**ObjectNotFound**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

Accessed object was not found

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 404

exception galaxy.exceptions.**RequestParameterInvalidException**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

exception galaxy.exceptions.**RequestParameterMissingException**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

exception galaxy.exceptions.**ToolMetaParameterException**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

exception galaxy.exceptions.**UnknownContentsType**(*err_msg=None, type='info', **extra_error_info*)

Bases: galaxy.exceptions.MessageException

err_code = <galaxy.exceptions.error_codes.ErrorCode object>

status_code = 400

external_services Package

actions Module

class galaxy.external_services.actions.**ExternalServiceAction**(*elem*, *parent*)
Bases: object

Abstract Class for External Service Actions

classmethod from_elem(*elem*, *parent*)

get_action_access_link(*trans*, *param_dict*)

handle_action(*completed_action*, *param_dict*, *trans*)

perform_action(*param_dict*)

populate_action(*param_dict*)

type = None

class galaxy.external_services.actions.**ExternalServiceResult**(*name*, *param_dict*)
Bases: object

content

class galaxy.external_services.actions.**ExternalServiceTemplateAction**(*elem*, *parent*)
Bases: *galaxy.external_services.actions.ExternalServiceAction*

Action that redirects to an external URL

perform_action(*param_dict*)

type = 'template'

class galaxy.external_services.actions.**ExternalServiceValueResult**(*name*, *param_dict*, *value*)
Bases: *galaxy.external_services.actions.ExternalServiceResult*

content

class galaxy.external_services.actions.**ExternalServiceWebAPIAction**(*elem*, *parent*)
Bases: *galaxy.external_services.actions.ExternalServiceAction*

Action that accesses an external Web API and provides handlers for the requested content

class ExternalServiceWebAPIActionRequest(*elem*, *parent*)
Bases: object

get_web_api_action(*param_dict*)

ExternalServiceWebAPIAction.**perform_action**(*param_dict*)

ExternalServiceWebAPIAction.**type = 'web_api'**

class galaxy.external_services.actions.**ExternalServiceWebAPIActionResult**(*name*, *param_dict*, *url*, *method*, *target*)
Bases: *galaxy.external_services.actions.ExternalServiceResult*

content

class galaxy.external_services.actions.**ExternalServiceWebAction**(*elem*, *parent*)
Bases: *galaxy.external_services.actions.ExternalServiceAction*

Action that accesses an external web application

get_action_access_link(*trans*, *param_dict*)

type = 'web'

class galaxy.external_services.actions.**PopulatedExternalServiceAction**(*action*, *param_dict*)

Bases: object

get_action_access_link(*trans*)

handle_results(*trans*)

perform_action()

class galaxy.external_services.actions.**Template**(*elem*, *parent*)

Bases: object

build_template(*param_dict*)

parameters Module

class galaxy.external_services.parameters.**ExternalServiceParameter**(*elem*, *parent*)

Bases: object

Abstract Class for External Service Parameters

classmethod from_elem(*elem*, *parent*)

get_value(*param_dict*)

requires_user_input = False

type = None

class galaxy.external_services.parameters.**ExternalServiceTemplateParameter**(*elem*, *parent*)

Bases: *galaxy.external_services.parameters.ExternalServiceParameter*

Parameter that returns a string containing the requested content

get_value(*param_dict*)

type = 'template'

service Module

class galaxy.external_services.service.**ActionSection**(*name*, *label*)

Bases: list

has_action()

class galaxy.external_services.service.**BooleanExternalServiceActionsGroupWhen**(*parent*, *name*, *value*, *label=None*)

Bases: *galaxy.external_services.service.ExternalServiceActionsGroupWhen*

classmethod from_elem(*parent*, *elem*)

Returns an instance of this when

is_case(*param_dict*)

type = 'boolean'

class galaxy.external_services.service.**ExternalServiceActionsConditional**(*elem*, *parent*)

Bases: object

get_current_cases(*param_dict*)

type = 'conditional'

class galaxy.external_services.service.**ExternalServiceActionsGroup**(*parent*, *name*, *label=None*)

Bases: object

add_item(*item*)

classmethod from_elem(*elem*, *parent=None*)

Return ExternalServiceActionsGroup created from an xml element.

load_sub_elems(*elem*)

populate(*service_instance*, *item=None*, *param_dict=None*)

prepare_actions(*param_dict*, *parent_dict*, *parent_section*)

class galaxy.external_services.service.**ExternalServiceActionsGroupWhen**(*parent*, *name*, *label=None*)

Bases: galaxy.external_services.service.ExternalServiceActionsGroup

classmethod from_elem(*parent*, *elem*)

Loads the proper when by attributes of elem

get_ref(*param_dict*)

is_case(*param_dict*)

type = 'when'

class galaxy.external_services.service.**ItemIsInstanceExternalServiceActionsGroupWhen**(*parent*, *name*, *value*, *label=None*)

Bases: galaxy.external_services.service.ExternalServiceActionsGroupWhen

classmethod from_elem(*parent*, *elem*)

Returns an instance of this when

is_case(*param_dict*)

type = 'item_type'

class galaxy.external_services.service.**PopulatedExternalService**(*service_group*, *service_instance*, *item*, *param_dict=None*)

Bases: object

get_action_by_name(*actions_list*)

perform_action_by_name(*actions_list*)

populate()

class galaxy.external_services.service.**ValueExternalServiceActionsGroupWhen**(*parent*, *name*, *value*, *label=None*)

Bases: galaxy.external_services.service.ExternalServiceActionsGroupWhen

classmethod from_elem(*parent*, *elem*)

Returns an instance of this when

is_case(*param_dict*)

type = 'value'

galaxy.external_services.service.**class_type**

alias of ItemIsInstanceExternalServiceActionsGroupWhen

Subpackages

result_handlers Package

basic Module

class galaxy.external_services.result_handlers.basic.**ExternalServiceActionJQueryGridResultHan**

Bases: galaxy.external_services.result_handlers.basic.ExternalServiceActionResultHandler

Class for External Service Actions JQuery Result Handler

handle_result(*result*, *param_dict*, *trans*)

type = 'jquery_grid'

class galaxy.external_services.result_handlers.basic.**ExternalServiceActionJSONResultHandler**(*e p e*

Bases: galaxy.external_services.result_handlers.basic.ExternalServiceActionResultHandler

Class for External Service Actions JQuery Result Handler

handle_result(*result*, *param_dict*, *trans*)

type = 'json_display'

class galaxy.external_services.result_handlers.basic.**ExternalServiceActionResultHandler**(*elem*, *parent*)

Bases: object

Basic Class for External Service Actions Result Handlers

classmethod from_elem(*elem*, *parent*)

handle_result(*result*, *param_dict*, *trans*)

type = 'display'

class galaxy.external_services.result_handlers.basic.**ExternalServiceActionURLRedirectResultHa**

Bases: galaxy.external_services.result_handlers.basic.ExternalServiceActionResultHandler

Basic Class for External Service Actions Result Handlers

classmethod from_elem(*elem*, *parent*)

handle_result(*result*, *param_dict*, *trans*)

type = 'web_redirect'

galaxy.external_services.result_handlers.basic.**handler_class**
alias of *ExternalServiceActionJSONResultHandler*

forms Package

forms Module FormDefinition and field factories

class galaxy.forms.forms.**FormDefinitionAddressFieldFactory**
Bases: *galaxy.forms.forms.FormDefinitionFieldFactory*

from_elem(*elem*, *layout=None*)
Return FormDefinition field created from an xml element.

new(*name=None*, *label=None*, *required=False*, *helptext=None*, *default=None*, *visible=True*, *layout=None*)
Return new FormDefinition field.

type = 'address'

class galaxy.forms.forms.**FormDefinitionFactory**(*form_types*, *field_type_factories*)
Bases: object

from_elem(*elem*, *form_definition_current=None*)
Return FormDefinition created from an xml element.

new(*form_type*, *name*, *description=None*, *fields=None*, *layout=None*, *form_definition_current=None*)
Return new FormDefinition.

class galaxy.forms.forms.**FormDefinitionFieldFactory**
Bases: object

from_elem(*elem*, *layout=None*)
Return FormDefinition created from an xml element.

new(*name=None*, *label=None*, *required=False*, *helptext=None*, *default=None*, *visible=True*, *layout=None*)
Return new FormDefinition field.

type = None

class galaxy.forms.forms.**FormDefinitionHistoryFieldFactory**
Bases: *galaxy.forms.forms.FormDefinitionFieldFactory*

from_elem(*elem*, *layout=None*)
Return FormDefinition field created from an xml element.

new(*name=None*, *label=None*, *required=False*, *helptext=None*, *default=None*, *visible=True*, *layout=None*)
Return new FormDefinition field.

type = 'history'

class galaxy.forms.forms.**FormDefinitionPasswordFieldFactory**
Bases: *galaxy.forms.forms.FormDefinitionFieldFactory*

from_elem(*elem*, *layout=None*)
Return FormDefinition field created from an xml element.

new(*name=None*, *label=None*, *required=False*, *helptext=None*, *default=None*, *visible=True*, *layout=None*, *area=False*)
Return new FormDefinition field.

type = 'password'

class galaxy.forms.forms.**FormDefinitionSelectFieldFactory**
Bases: *galaxy.forms.forms.FormDefinitionFieldFactory*

from_elem(*elem*, *layout=None*)
Return FormDefinition field created from an xml element.

new(*name=None*, *label=None*, *required=False*, *helptext=None*, *default=None*, *visible=True*, *layout=None*, *options=[]*, *checkboxes=False*)
Return new FormDefinition field.

type = 'select'

class galaxy.forms.forms.**FormDefinitionTextFieldFactory**
Bases: *galaxy.forms.forms.FormDefinitionFieldFactory*

from_elem(*elem*, *layout=None*)
Return FormDefinition field created from an xml element.

new(*name=None*, *label=None*, *required=False*, *helptext=None*, *default=None*, *visible=True*, *layout=None*, *area=False*)
Return new FormDefinition field.

type = 'text'

class galaxy.forms.forms.**FormDefinitionWorkflowFieldFactory**
Bases: *galaxy.forms.forms.FormDefinitionFieldFactory*

from_elem(*elem*, *layout=None*)
Return FormDefinition field created from an xml element.

new(*name=None*, *label=None*, *required=False*, *helptext=None*, *default=None*, *visible=True*, *layout=None*)
Return new FormDefinition field.

type = 'workflow'

class galaxy.forms.forms.**FormDefinitionWorkflowMappingFieldFactory**
Bases: *galaxy.forms.forms.FormDefinitionFieldFactory*

from_elem(*elem*, *layout=None*)
Return FormDefinition field created from an xml element.

new(*name=None*, *label=None*, *required=False*, *helptext=None*, *default=None*, *visible=True*, *layout=None*)
Return new FormDefinition field.

type = 'workflowmapping'

galaxy.forms.forms.**field**
alias of *FormDefinitionHistoryFieldFactory*

jobs Package

jobs Package Support for running a tool in Galaxy via an internal job management system

class galaxy.jobs.**ComputeEnvironment**
Bases: object

Definition of the job as it will be run on the (potentially) remote compute server.

config_directory()
Directory containing config files (potentially remote)

input_paths()
Input DatasetPaths defined by job.

new_file_path()
Absolute path to dump new files for this job on compute server.

output_paths()
Output DatasetPaths defined by job.

sep()
os.path.sep for the platform this job will execute in.

tool_directory()
Absolute path to tool files for this job on compute server.

unstructured_path_rewriter()
Return a function that takes in a value, determines if it is path to be rewritten (will be passed non-path values as well - onus is on this function to determine both if its input is a path and if it should be rewritten.)

version_path()
Location of the version file for the underlying tool.

working_directory()
Job working directory (potentially remote)

class galaxy.jobs.**JobConfiguration**(*app*)
Bases: object

A parser and interface to advanced job management features.

These features are configured in the job configuration, by default, job_conf.xml

DEFAULT_NWORKERS = 4

convert_legacy_destinations(*job_runners*)
Converts legacy (from a URL) destinations to contain the appropriate runner params defined in the URL.

Parameters **job_runners** (*list of job runner plugins*) – All loaded job runner plugins.

default_job_tool_configuration
The default JobToolConfiguration, used if a tool does not have an explicit defintion in the configuration. It consists of a reference to the default handler and default destination.

Returns JobToolConfiguration – a representation of a <tool> element that uses the default handler and destination

get_destination(*id_or_tag*)
Given a destination ID or tag, return the JobDestination matching the provided ID or tag

Parameters **id_or_tag** (*str*) – A destination ID or tag.

Returns JobDestination – A valid destination

Destinations are deepcopied as they are expected to be passed in to job runners, which will modify them for persisting params set at runtime.

get_destinations(*id_or_tag*)
Given a destination ID or tag, return all JobDestinations matching the provided ID or tag

Parameters **id_or_tag** (*str*) – A destination ID or tag.

Returns list or tuple of JobDestinations

Destinations are not deepcopied, so they should not be passed to anything which might modify them.

get_handler(*id_or_tag*)
Given a handler ID or tag, return the provided ID or an ID matching the provided tag

Parameters **id_or_tag** (*str*) – A handler ID or tag.

Returns str – A valid job handler ID.

get_job_runner_plugins(*handler_id*)
Load all configured job runner plugins

Returns list of job runner plugins

get_job_tool_configurations(*ids*)
Get all configured JobToolConfigurations for a tool ID, or, if given a list of IDs, the JobToolConfigurations for the first id in `ids` matching a tool definition.

Note: You should not mix tool shed tool IDs, versionless tool shed IDs, and tool config tool IDs that refer to the same tool.

Parameters **ids** (*list or str.*) – Tool ID or IDs to fetch the JobToolConfiguration of.

Returns list – JobToolConfiguration Bunches representing <tool> elements matching the specified ID(s).

Example tool ID strings include:

- Full tool shed id: `toolshed.example.org/repos/nate/filter_tool_repo/filter_tool/1.0.0`
- Tool shed id less version: `toolshed.example.org/repos/nate/filter_tool_repo/filter_tool`
- Tool config tool id: `filter_tool`

get_tool_resource_parameters(*tool_id*)
Given a tool id, return XML elements describing parameters to insert into job resources.

Tool id A tool ID (a string)

Returns List of parameter elements.

is_handler(*server_name*)
Given a server name, indicate whether the server is a job handler

Parameters **server_name** (*str*) – The name to check

Returns bool

is_id(*collection*)
Given a collection of handlers or destinations, indicate whether the collection represents a tag or a real ID

Parameters **collection** (*tuple or list*) – A representation of a destination or handler

Returns bool

is_tag(*collection*)
Given a collection of handlers or destinations, indicate whether the collection represents a tag or a real ID

Parameters **collection** (*tuple or list*) – A representation of a destination or handler

Returns bool

class galaxy.jobs.**JobDestination**(***kwds*)
Bases: galaxy.util.bunch.Bunch

Provides details about where a job runs

class galaxy.jobs.**JobToolConfiguration**(***kwds*)
Bases: galaxy.util.bunch.Bunch

Provides details on what handler and destination a tool should use

A JobToolConfiguration will have the required attribute 'id' and optional attributes 'handler', 'destination', and 'params'

get_resource_group()

class galaxy.jobs.**JobWrapper**(*job*, *queue*, *use_persisted_destination=False*)
Bases: object

Wraps a 'model.Job' with convenience methods for running processes and state management.

can_split()

change_ownership_for_run()

change_state(*state*, *info=False*)

check_limits(*runtime=None*)

check_tool_output(*stdout*, *stderr*, *tool_exit_code*, *job*)

cleanup(*delete_files=True*)

clear_working_directory()

commands_in_new_shell

compute_outputs()

create_working_directory()

default_compute_environment(*job=None*)

fail(*message*, *exception=False*, *stdout=''*, *stderr=''*, *exit_code=None*)
Indicate job failure by setting state and message on all output datasets.

finish(*stdout*, *stderr*, *tool_exit_code=None*, *remote_working_directory=None*)
Called to indicate that the associated command has been run. Updates the output datasets based on stderr and stdout from the command, and the contents of the output files.

galaxy_lib_dir

galaxy_system_pwent

get_command_line()

get_dataset_finish_context(*job_context*, *dataset*)

get_env_setup_clause()

get_id_tag()

get_input_dataset_fnames(*ds*)

get_input_fnames()

get_input_paths(*job=None*)

get_job()

get_job_runner()

get_job_runner_url()

get_mutable_output_fnames()

get_output_destination(*output_path*)
Destination for outputs marked as from_work_dir. This is the normal case, just copy these files directly to the ulimate destination.

get_output_file_id(*file*)

get_output_fnames()

get_output_hdas_and_fnames()

get_output_sizes()

get_parallelism()

get_param_dict()
Restore the dictionary of parameters from the database.

get_session_id()

get_state()

get_tool_provided_job_metadata()

get_version_string_path()

has_limits()

invalidate_external_metadata()

job_destination
Return the JobDestination that this job will use to run. This will either be a configured destination, a randomly selected destination if the configured destination was a tag, or a dynamically generated destination from the dynamic runner.

Calling this method for the first time causes the dynamic runner to do its calculation, if any.

Returns `JobDestination`

mark_as_resubmitted(*info=None*)

pause(*job=None*, *message=None*)

prepare(*compute_environment=None*)
Prepare the job to run by creating the working directory and the config files.

reclaim_ownership()

requires_setting_metadata

set_job_destination(*job_destination*, *external_id=None*)
Persist job destination params in the database for recovery.

self.job_destination is not used because a runner may choose to rewrite parts of the destination (e.g. the params).

set_runner(*runner_url*, *external_id*)

setup_external_metadata(*exec_dir=None*, *tmp_dir=None*, *dataset_files_path=None*, *config_root=None*, *config_file=None*, *datatypes_config=None*, *set_extension=True*, ***kwds*)

user

user_system_pwent

class galaxy.jobs.**NoopQueue**
Bases: object

Implements the JobQueue / JobStopQueue interface but does nothing

put(**args*, ***kwargs*)

put_stop(**args*)

shutdown()

class galaxy.jobs.**ParallelismInfo**(*tag*)
Bases: object

Stores the information (if any) for running multiple instances of the tool in parallel on the same set of inputs.

class galaxy.jobs.**SharedComputeEnvironment**(*job_wrapper*, *job*)
Bases: *galaxy.jobs.SimpleComputeEnvironment*

Default ComputeEnviornment for job and task wrapper to pass to ToolEvaluator - valid when Galaxy and compute share all the relevant file systems.

input_paths()

new_file_path()

output_paths()

tool_directory()

version_path()

working_directory()

class galaxy.jobs.**SimpleComputeEnvironment**
Bases: object

config_directory()

sep()

unstructured_path_rewriter()

class galaxy.jobs.**TaskWrapper**(*task*, *queue*)
Bases: *galaxy.jobs.JobWrapper*

Extension of JobWrapper intended for running tasks. Should be refactored into a generalized executable unit wrapper parent, then jobs and tasks.

can_split()

change_state(*state*, *info=False*)

cleanup(*delete_files=True*)

fail(*message*, *exception=False*)

finish(*stdout*, *stderr*, *tool_exit_code=None*)
Called to indicate that the associated command has been run. Updates the output datasets based on stderr and stdout from the command, and the contents of the output files.

get_command_line()

get_dataset_finish_context(*job_context*, *dataset*)

get_exit_code()

get_id_tag()

get_job()

get_output_destination(*output_path*)
Destination for outputs marked as from_work_dir. These must be copied with the same basenme as the path for the ultimate output destination. This is required in the task case so they can be merged.

get_output_file_id(*file*)

get_param_dict()
Restore the dictionary of parameters from the database.

get_session_id()

get_state()

get_task()

get_tool_provided_job_metadata()

prepare(*compute_environment=None*)
Prepare the job to run by creating the working directory and the config files.

set_runner(*runner_url*, *external_id*)

setup_external_metadata(*exec_dir=None*, *tmp_dir=None*, *dataset_files_path=None*, *config_root=None*, *config_file=None*, *datatypes_config=None*, *set_extension=True*, ***kwds*)

galaxy.jobs.**config_exception**(*e*, *file*)

handler Module Galaxy job handler, prepares, runs, tracks, and finishes Galaxy jobs

class galaxy.jobs.handler.**DefaultJobDispatcher**(*app*)
Bases: object

put(*job_wrapper*)

recover(*job*, *job_wrapper*)

shutdown()

stop(*job*)
Stop the given job. The input variable job may be either a Job or a Task.

url_to_destination(*url*)
This is used by the runner mapper (a.k.a. dynamic runner) and recovery methods to have runners convert URLs to destinations.

New-style runner plugin IDs must match the URL's scheme for this to work.

class galaxy.jobs.handler.**JobHandler**(*app*)
Bases: object

Handle the preparation, running, tracking, and finishing of jobs

shutdown()

start()

class galaxy.jobs.handler.**JobHandlerQueue**(*app*, *dispatcher*)
Bases: object

Job Handler's Internal Queue, this is what actually implements waiting for jobs to be runnable and dispatching to a JobRunner.

STOP_SIGNAL = <object object>

get_total_job_count_per_destination()

get_user_job_count(*user_id*)

get_user_job_count_per_destination(*user_id*)

increase_running_job_count(*user_id*, *destination_id*)

job_pair_for_id(*id*)

job_wrapper(*job*, *use_persisted_destination=False*)

put(*job_id*, *tool_id*)
Add a job to the queue (by job identifier)

shutdown()
Attempts to gracefully shut down the worker thread

start()
Starts the JobHandler's thread after checking for any unhandled jobs.

class galaxy.jobs.handler.**JobHandlerStopQueue**(*app*, *dispatcher*)
Bases: object

A queue for jobs which need to be terminated prematurely.

STOP_SIGNAL = <object object>

monitor()
Continually iterate the waiting jobs, stop any that are found.

monitor_step()
Called repeatedly by *monitor* to stop jobs.

put(*job_id*, *error_msg=None*)

shutdown()
Attempts to gracefully shut down the worker thread

manager Module Top-level Galaxy job manager, moves jobs to handler(s)

class galaxy.jobs.manager.**JobManager**(*app*)
Bases: object

Highest level interface to job management.
TODO: Currently the app accesses "job_queue" and "job_stop_queue" directly. This should be decoupled.

shutdown()

start()

class galaxy.jobs.manager.**NoopHandler**(**args*, ***kwargs*)
Bases: object

shutdown(**args*)

start()

mapper Module

exception galaxy.jobs.mapper.**JobMappingException**(*failure_message*)
Bases: exceptions.Exception

exception galaxy.jobs.mapper.**JobNotReadyException**(*job_state=None*, *message=None*)
Bases: exceptions.Exception

class galaxy.jobs.mapper.**JobRunnerMapper**(*job_wrapper*, *url_to_destination*, *job_config*)
Bases: object

This class is responsible to managing the mapping of jobs (in the form of job_wrappers) to job runner url strings.

cache_job_destination(*raw_job_destination*)

get_job_destination(*params*)
Cache the job_destination to avoid recalculation.

transfer_manager Module Manage transfers from arbitrary URLs to temporary files. Socket interface for IPC with multiple process configurations.

class galaxy.jobs.transfer_manager.**Sleeper**
Bases: object

Provides a 'sleep' method that sleeps for a number of seconds *unless* the notify method is called (from a different thread).

sleep(*seconds*)

wake()

class galaxy.jobs.transfer_manager.**TransferManager**(*app*)
Bases: object

Manage simple data transfers from URLs to temporary locations.

get_state(*transfer_jobs*, *via_socket=False*)

new(*path=None*, ***kwd*)

run(*transfer_jobs*)
This method blocks, so if invoking the transfer manager ever starts taking too long, we should move it to a thread. However, the transfer_manager will either daemonize or return after submitting to a running daemon, so it should be fairly quick to return.

shutdown()

Subpackages

actions Package

actions Package This package contains job action classes.

post Module Actions to be run at job completion (or output hda creation, as in the case of immediate_actions listed below. Currently only used in workflows.

class galaxy.jobs.actions.post.**ActionBox**
Bases: object

actions = {'ChangeDatatypeAction': <class 'galaxy.jobs.actions.post.ChangeDatatypeAction'>, 'RenameDatasetAction

classmethod execute(*app*, *sa_session*, *pja*, *job*, *replacement_dict=None*)

classmethod get_add_list()

classmethod get_forms(*trans*)

classmethod get_short_str(*action*)

classmethod handle_incoming(*incoming*)

immediate_actions = ['ChangeDatatypeAction', 'RenameDatasetAction', 'TagDatasetAction']

public_actions = ['RenameDatasetAction', 'ChangeDatatypeAction', 'ColumnSetAction', 'EmailAction', 'DeleteInte

class galaxy.jobs.actions.post.**ChangeDatatypeAction**
Bases: *galaxy.jobs.actions.post.DefaultJobAction*

classmethod execute(*app*, *sa_session*, *action*, *job*, *replacement_dict*)

classmethod get_config_form(*trans*)

classmethod get_short_str(*pja*)

name = 'ChangeDatatypeAction'

verbose_name = 'Change Datatype'

class galaxy.jobs.actions.post.**ColumnSetAction**
Bases: *galaxy.jobs.actions.post.DefaultJobAction*

classmethod execute(*app*, *sa_session*, *action*, *job*, *replacement_dict*)

classmethod get_config_form(*trans*)

classmethod get_short_str(*pja*)

name = 'ColumnSetAction'

verbose_name = 'Assign Columns'

class galaxy.jobs.actions.post.**DefaultJobAction**
Bases: object

Base job action.

classmethod execute(*app*, *sa_session*, *action*, *job*, *replacement_dict=None*)

classmethod get_config_form(*trans*)

classmethod get_short_str(*pja*)

name = 'DefaultJobAction'

verbose_name = 'Default Job'

class galaxy.jobs.actions.post.**DeleteDatasetAction**
Bases: *galaxy.jobs.actions.post.DefaultJobAction*

classmethod execute(*app*, *sa_session*, *action*, *job*, *replacement_dict*)

classmethod get_config_form(*trans*)

classmethod get_short_str(*pja*)

name = 'DeleteDatasetAction'

verbose_name = 'Delete Dataset'

class galaxy.jobs.actions.post.**DeleteIntermediatesAction**
Bases: *galaxy.jobs.actions.post.DefaultJobAction*

classmethod execute(*app*, *sa_session*, *action*, *job*, *replacement_dict*)

classmethod get_config_form(*trans*)

classmethod get_short_str(*pja*)

name = 'DeleteIntermediatesAction'

verbose_name = 'Delete Non-Output Completed Intermediate Steps'

class galaxy.jobs.actions.post.**EmailAction**
Bases: *galaxy.jobs.actions.post.DefaultJobAction*

This action sends an email to the galaxy user responsible for a job.

classmethod execute(*app*, *sa_session*, *action*, *job*, *replacement_dict*)

classmethod get_config_form(*trans*)

classmethod get_short_str(*pja*)

name = 'EmailAction'

verbose_name = 'Email Notification'

class galaxy.jobs.actions.post.**HideDatasetAction**
Bases: *galaxy.jobs.actions.post.DefaultJobAction*

classmethod execute(*app*, *sa_session*, *action*, *job*, *replacement_dict*)

classmethod get_config_form(*trans*)

classmethod get_short_str(*pja*)

name = 'HideDatasetAction'

verbose_name = 'Hide Dataset'

class galaxy.jobs.actions.post.**RenameDatasetAction**
Bases: *galaxy.jobs.actions.post.DefaultJobAction*

classmethod execute(*app*, *sa_session*, *action*, *job*, *replacement_dict*)

classmethod get_config_form(*trans*)

classmethod get_short_str(*pja*)

name = 'RenameDatasetAction'

verbose_name = 'Rename Dataset'

class galaxy.jobs.actions.post.**SetMetadataAction**
Bases: *galaxy.jobs.actions.post.DefaultJobAction*

classmethod execute(*app*, *sa_session*, *action*, *job*, *replacement_dict*)

classmethod get_config_form(*trans*)

name = 'SetMetadataAction'

class galaxy.jobs.actions.post.**TagDatasetAction**
Bases: *galaxy.jobs.actions.post.DefaultJobAction*

classmethod execute(*app*, *sa_session*, *action*, *job*, *replacement_dict*)

classmethod get_config_form(*trans*)

classmethod get_short_str(*pja*)

name = 'TagDatasetAction'

verbose_name = 'Add tag to dataset'

galaxy.jobs.actions.post.**get_form_template**(*action_type*, *title*, *content*, *help*, *on_output=True*)

deferred Package

deferred Package Queue for running deferred code via plugins.

class galaxy.jobs.deferred.**DeferredJobQueue**(*app*)
Bases: object

job_states = <galaxy.util.bunch.Bunch object>

shutdown()

class galaxy.jobs.deferred.**FakeTrans**(*app*, *history=None*, *user=None*)
Bases: object

A fake trans for calling the external set metadata tool

db_dataset_for(*dbkey*)

get_current_user_roles()

get_galaxy_session()

log_event(*message*, *tool_id=None*)

class galaxy.jobs.deferred.**Sleeper**
Bases: object

Provides a 'sleep' method that sleeps for a number of seconds *unless* the notify method is called (from a different thread).

sleep(*seconds*)

wake()

data_transfer Module Module for managing data transfer jobs.

class galaxy.jobs.deferred.data_transfer.**DataTransfer**(*app*)
Bases: object

check_interval = 15

check_job(*job*)

create_job(*trans*, ***kwd*)

dataset_datatype_re = <_sre.SRE_Pattern object>

dataset_name_re = <_sre.SRE_Pattern object>

run_job(*job*)

genome_index Module

genome_transfer Module

liftover_transfer Module

manual_data_transfer Module Generic module for managing manual data transfer jobs using Galaxy's built-in file browser. This module can be used by various external services that are configured to transfer data manually.

class galaxy.jobs.deferred.manual_data_transfer.**ManualDataTransferPlugin**(*app*)
Bases: galaxy.jobs.deferred.data_transfer.DataTransfer

check_job(*job*)

create_job(*trans*, ***kwd*)

pacific_biosciences_smrt_portal Module Module for managing jobs in Pacific Bioscience's SMRT Portal and automatically transferring files produced by SMRT Portal.

class galaxy.jobs.deferred.pacific_biosciences_smrt_portal.**SMRTPortalPlugin**(*app*)
Bases: galaxy.jobs.deferred.data_transfer.DataTransfer

api_path = '/smrtportal/api'

check_job(*job*)

create_job(*trans*, ***kwd*)

runners Package

runners Package Base classes for job runner plugins.

class galaxy.jobs.runners.**AsynchronousJobRunner**(*app*, *nworkers*, ***kwargs*)
Bases: galaxy.jobs.runners.BaseJobRunner

Parent class for any job runner that runs jobs asynchronously (e.g. via a distributed resource manager). Provides general methods for having a thread to monitor the state of asynchronous jobs and submitting those jobs to the correct methods (queue, finish, cleanup) at appropriate times..

check_watched_item(*job_state*)

check_watched_items()
This method is responsible for iterating over self.watched and handling state changes and updating self.watched with a new list of watched job states. Subclasses can opt to override this directly (as older job runners will initially) or just override check_watched_item and allow the list processing to reuse the logic here.

fail_job(*job_state*)

finish_job(*job_state*)
Get the output/error for a finished job, pass to *job_wrapper.finish* and cleanup all the job's temporary files.

handle_stop()

mark_as_failed(*job_state*)

mark_as_finished(*job_state*)

monitor()
Watches jobs currently in the monitor queue and deals with state changes (queued to running) and job completion.

monitor_job(*job_state*)

shutdown()
Attempts to gracefully shut down the monitor thread

class galaxy.jobs.runners.**AsynchronousJobState**(*files_dir=None*, *job_wrapper=None*, *job_id=None*, *job_file=None*, *output_file=None*, *error_file=None*, *exit_code_file=None*, *job_name=None*, *job_destination=None*)

Bases: galaxy.jobs.runners.JobState

Encapsulate the state of an asynchronous job, this should be subclassed as needed for various job runners to capture additional information needed to communicate with distributed resource manager.

check_limits(*runtime=None*)

cleanup()

register_cleanup_file_attribute(*attribute*)

running

class galaxy.jobs.runners.**BaseJobRunner**(*app*, *nworkers*, ***kwargs*)

Bases: object

DEFAULT_SPECS = {'recheck_missing_job_retries': {'default': 0, 'map': <type 'int'>, 'valid': <function <lambda> at 0x

build_command_line(*job_wrapper*, *include_metadata=False*, *include_work_dir_outputs=True*)

get_job_file(*job_wrapper*, ***kwds*)

get_work_dir_outputs(*job_wrapper*, *job_working_directory=None*)

Returns list of pairs (source_file, destination) describing path to work_dir output file and ultimate destination.

mark_as_queued(*job_wrapper*)

mark_as_resubmitted(*job_state*, *info=None*)

parse_destination_params(*params*)

Parse the JobDestination `params` dict and return the runner's native representation of those params.

prepare_job(*job_wrapper*, *include_metadata=False*, *include_work_dir_outputs=True*)

Some sanity checks that all runners' queue_job() methods are likely to want to do

put(*job_wrapper*)

Add a job to the queue (by job identifier), indicate that the job is ready to run.

queue_job(*job_wrapper*)

recover(*job*, *job_wrapper*)

run_next()

Run the next item in the work queue (a job waiting to run)

shutdown()

Attempts to gracefully shut down the worker threads

stop_job(*job*)

url_to_destination(*url*)

Convert a legacy URL to a JobDestination.

Job runner URLs are deprecated, JobDestinations should be used instead. This base class method converts from a URL to a very basic JobDestination without destination params.

class galaxy.jobs.runners.**JobState**

Bases: object

Encapsulate state of jobs.

static default_exit_code_file (*files_dir*, *id_tag*)

static default_job_file (*files_dir*, *id_tag*)

runner_states = <galaxy.util.bunch.Bunch object>

set_defaults (*files_dir*)

class galaxy.jobs.runners.**RunnerParams** (*specs=None*, *params=None*)
Bases: *galaxy.util.ParamsWithSpecs*

cli Module Job control via a command line interface (e.g. qsub/qstat), possibly over a remote connection (e.g. ssh).

class galaxy.jobs.runners.cli.**ShellJobRunner** (*app*, *nworkers*)
Bases: *galaxy.jobs.runners.AsynchronousJobRunner*

Job runner backed by a finite pool of worker threads. FIFO scheduling

check_watched_items ()
Called by the monitor thread to look at each watched job and deal with state changes.

finish_job (*job_state*)
For recovery of jobs started prior to standardizing the naming of files in the AsychronousJobState object

get_cli_plugins (*shell_params*, *job_params*)

parse_destination_params (*params*)

queue_job (*job_wrapper*)
Create job script and submit it to the DRM

recover (*job*, *job_wrapper*)
Recovers jobs stuck in the queued/running state when Galaxy started

runner_name = 'ShellRunner'

stop_job (*job*)
Attempts to delete a dispatched job

url_to_destination (*url*)

condor Module Job control via the Condor DRM.

class galaxy.jobs.runners.condor.**CondorJobRunner** (*app*, *nworkers*)
Bases: *galaxy.jobs.runners.AsynchronousJobRunner*

Job runner backed by a finite pool of worker threads. FIFO scheduling

check_watched_items ()
Called by the monitor thread to look at each watched job and deal with state changes.

queue_job (*job_wrapper*)
Create job script and submit it to the DRM

recover (*job*, *job_wrapper*)
Recovers jobs stuck in the queued/running state when Galaxy started

runner_name = 'CondorRunner'

stop_job (*job*)
Attempts to delete a job from the DRM queue

drmaa Module Job control via the DRMAA API.

class galaxy.jobs.runners.drmaa.**DRMAAJobRunner**(*app*, *nworkers*, ***kwargs*)
Bases: galaxy.jobs.runners.AsynchronousJobRunner

Job runner backed by a finite pool of worker threads. FIFO scheduling

check_watched_items()
Called by the monitor thread to look at each watched job and deal with state changes.

external_runjob(*jobtemplate_filename*, *username*)
runs an external script the will QSUB a new job. The external script will be run with sudo, and will setuid() to the specified user. Effectively, will QSUB as a different user (then the one used by Galaxy).

get_native_spec(*url*)
Get any native DRM arguments specified by the site configuration

queue_job(*job_wrapper*)
Create job script and submit it to the DRM

recover(*job*, *job_wrapper*)
Recovers jobs stuck in the queued/running state when Galaxy started

runner_name = 'DRMAARunner'

stop_job(*job*)
Attempts to delete a job from the DRM queue

store_jobtemplate(*job_wrapper*, *jt*)
Stores the content of a DRMAA JobTemplate object in a file as a JSON string. Path is hard-coded, but it's no worse than other path in this module. Uses Galaxy's JobID, so file is expected to be unique.

url_to_destination(*url*)
Convert a legacy URL to a job destination

local Module Job runner plugin for executing jobs on the local system via the command line.

class galaxy.jobs.runners.local.**LocalJobRunner**(*app*, *nworkers*)
Bases: galaxy.jobs.runners.BaseJobRunner

Job runner backed by a finite pool of worker threads. FIFO scheduling

queue_job(*job_wrapper*)

recover(*job*, *job_wrapper*)

runner_name = 'LocalRunner'

stop_job(*job*)

lwr Module

class galaxy.jobs.runners.lwr.**LwrJobRunner**(*app*, *nworkers*, ***kwds*)
Bases: galaxy.jobs.runners.AsynchronousJobRunner

LWR Job Runner

check_pid(*pid*)

check_watched_item(*job_state*)

fail_job(*job_state*)
Seperated out so we can use the worker threads for it.

finish_job(*job_state*)

get_client (*job_destination_params*, *job_id*, *env=[]*)

get_client_from_state (*job_state*)

get_client_from_wrapper (*job_wrapper*)

get_input_files (*job_wrapper*)

get_output_files (*job_wrapper*)

queue_job (*job_wrapper*)

recover (*job*, *job_wrapper*)
Recovers jobs stuck in the queued/running state when Galaxy started

runner_name = 'LWRRunner'

shutdown ()

stop_job (*job*)

url_to_destination (*url*)
Convert a legacy URL to a job destination

pbs Module

tasks Module

class galaxy.jobs.runners.tasks.**TaskedJobRunner** (*app*, *nworkers*)
Bases: *galaxy.jobs.runners.BaseJobRunner*

Job runner backed by a finite pool of worker threads. FIFO scheduling

queue_job (*job_wrapper*)

recover (*job*, *job_wrapper*)

runner_name = 'TaskRunner'

stop_job (*job*)

Subpackages

cli_job Package

cli_job Package

torque Module

cli_shell Package

cli_shell Package

rsh Module

splitters Package

basic Module

galaxy.jobs.splitters.basic.**do_merge**(*job_wrapper*, *task_wrappers*)
galaxy.jobs.splitters.basic.**do_split**(*job_wrapper*)

galaxy.jobs.splitters.basic.**set_basic_defaults**(*job_wrapper*)

multi Module

galaxy.jobs.splitters.multi.**do_merge**(*job_wrapper*, *task_wrappers*)
galaxy.jobs.splitters.multi.**do_split**(*job_wrapper*)

model Package

model Package Galaxy data model classes

Naming: try to use class names that have a distinct plural form so that the relationship cardinalities are obvious (e.g. prefer Dataset to Data)

class galaxy.model.**APIKeys**(*id=None*, *user_id=None*, *key=None*)
Bases: object

class galaxy.model.**BaseJobMetric**(*plugin*, *metric_name*, *metric_value*)
Bases: object

exception galaxy.model.**ConverterDependencyException**(*value*)
Bases: exceptions.Exception

class galaxy.model.**DataManagerHistoryAssociation**(*id=None*, *history=None*, *user=None*)
Bases: object

class galaxy.model.**DataManagerJobAssociation**(*id=None*, *job=None*, *data_manager_id=None*)
Bases: object

class galaxy.model.**Dataset**(*id=None*, *state=None*, *external_filename=None*, *extra_files_path=None*, *file_size=None*, *purgable=True*, *uuid=None*)
Bases: object

conversion_messages = <galaxy.util.bunch.Bunch object>

engine = None

extra_files_path

file_name

file_path = '/tmp/'

full_delete()
Remove the file and extra files, marks deleted and purged

get_access_roles(*trans*)

get_extra_files_path()

get_file_name()

get_manage_permissions_roles(*trans*)

get_size(*nice_size=False*)
Returns the size of the data on disk

get_total_size()

has_data()
Detects whether there is any data

has_manage_permissions_roles(*trans*)

in_ready_state()

is_multi_byte()

mark_deleted(*include_children=True*)

non_ready_states = ('upload', 'queued', 'running', 'setting_metadata')

object_store = None

permitted_actions = <galaxy.util.bunch.Bunch object>

ready_states = ('discarded', 'ok', 'failed_metadata', 'paused', 'error', 'new', 'empty')

set_extra_files_path(*extra_files_path*)

set_file_name(*filename*)

set_size()
Returns the size of the data on disk

set_total_size()

states = <galaxy.util.bunch.Bunch object>

user_can_purge

class galaxy.model.**DatasetCollection**(*id=None*, *collection_type=None*, *populated=True*)
Bases: object, *galaxy.model.item_attrs.Dictifiable*, *galaxy.model.item_attrs.UsesAnnotations*

copy(*destination=None*, *element_destination=None*)

dataset_elements

dataset_instances

dict_collection_visible_keys = ('id', 'collection_type')

dict_element_visible_keys = ('id', 'collection_type')

handle_population_failed(*message*)

mark_as_populated()

populated

populated_states = <galaxy.util.bunch.Bunch object>

set_from_dict(*new_data*)

state

validate()

waiting_for_elements

class galaxy.model.**DatasetCollectionElement**(*id=None*, *collection=None*, *element=None*, *element_index=None*, *element_identifier=None*)
Bases: object, *galaxy.model.item_attrs.Dictifiable*

Associates a DatasetInstance (hda or ldda) with a DatasetCollection.

copy_to_collection(*collection*, *destination=None*, *element_destination=None*)

dataset

dataset_instance

dict_collection_visible_keys = ('id', 'element_type', 'element_index', 'element_identifier')

dict_element_visible_keys = ('id', 'element_type', 'element_index', 'element_identifier')

element_object

element_type

first_dataset_instance()

is_collection

class galaxy.model.**DatasetCollectionInstance**(*collection=None*, *deleted=False*)

Bases: object, *galaxy.model.HasName*

display_name()

set_from_dict(*new_data*)

Set object attributes to the values in dictionary new_data limiting to only those keys in dict_element_visible_keys.

Returns a dictionary of the keys, values that have been changed.

state

class galaxy.model.**DatasetInstance**(*id=None*, *hid=None*, *name=None*, *info=None*, *blurb=None*, *peek=None*, *tool_version=None*, *extension=None*, *dbkey=None*, *metadata=None*, *history=None*, *dataset=None*, *deleted=False*, *designation=None*, *parent_id=None*, *validation_errors=None*, *visible=True*, *create_dataset=False*, *sa_session=None*, *extended_metadata=None*)

Bases: object

A base class for all 'dataset instances', HDAs, LDAs, etc

add_validation_error(*validation_error*)

as_display_type(*type*, ***kwd*)

can_convert_to(*format*)

change_datatype(*new_ext*)

clear_associated_files(*metadata_safe=False*, *purge=False*)

conversion_messages = <galaxy.util.bunch.Bunch object>

convert_dataset(*trans*, *target_type*)

Converts a dataset to the target_type and returns a message indicating status of the conversion. None is returned to indicate that dataset was converted successfully.

creating_job

datatype

dbkey

display_info()

display_name()

display_peek()

ext

extend_validation_errors(*validation_errors*)

extra_files_path

file_name

find_conversion_destination(*accepted_formats*, ***kwd*)
Returns (target_ext, existing converted dataset)

get_child_by_designation(*designation*)

get_converted_dataset(*trans*, *target_ext*)
Return converted dataset(s) if they exist, along with a dict of dependencies. If not converted yet, do so and return None (the first time). If unconvertible, raise exception.

get_converted_dataset_deps(*trans*, *target_ext*)
Returns dict of { "dependency" => HDA }

get_converted_files_by_type(*file_type*)

get_converter_types()

get_dataset_state()

get_datasources(*trans*)
Returns datasources for dataset; if datasources are not available due to indexing, indexing is started. Return value is a dictionary with entries of type (<datasource_type> : {<datasource_name>, <indexing_message>}).

get_dbkey()

get_display_applications(*trans*)

get_file_name()

get_metadata()

get_metadata_dataset(*dataset_ext*)
Returns an HDA that points to a metadata file which contains a converted data with the requested extension.

get_mime()
Returns the mime type of the data

get_raw_data()
Returns the full data. To stream it open the file_name and read/write as needed

get_size(*nice_size=False*)
Returns the size of the data on disk

get_total_size()

get_visualizations()

has_data()
Detects whether there is any data

init_meta(*copy_from=None*)

is_multi_byte()
Data consists of multi-byte characters

is_pending
Return true if the dataset is neither ready nor in error

mark_deleted(*include_children=True*)

mark_undeleted(*include_children=True*)

mark_unhidden(*include_children=True*)

metadata

missing_meta(***kwd*)

permitted_actions = <galaxy.util.bunch.Bunch object>

set_dataset_state(*state*)

set_dbkey(*value*)

set_file_name(*filename*)

set_meta(***kwd*)

set_metadata(*bunch*)

set_peek(*is_multi_byte=False*)

set_raw_data(*data*)
Saves the data on the disc

set_size()
Returns the size of the data on disk

set_total_size()

source_dataset_chain

source_library_dataset

state

states = <galaxy.util.bunch.Bunch object>

undeletable()

write_from_stream(*stream*)
Writes data from a stream

class galaxy.model.**DatasetPermissions**(*action*, *dataset*, *role*)
Bases: object

class galaxy.model.**DatasetTagAssociation**(*id=None*, *user=None*, *item_id=None*, *tag_id=None*, *user_tname=None*, *value=None*)
Bases: *galaxy.model.ItemTagAssociation*

class galaxy.model.**DatasetToValidationErrorAssociation**(*dataset*, *validation_error*)
Bases: object

class galaxy.model.**DefaultHistoryPermissions**(*history*, *action*, *role*)
Bases: object

class galaxy.model.**DefaultQuotaAssociation**(*type*, *quota*)
Bases: *galaxy.model.Quota*, *galaxy.model.item_attrs.Dictifiable*

dict_element_visible_keys = ('type',)

types = <galaxy.util.bunch.Bunch object>

class galaxy.model.**DefaultUserPermissions**(*user*, *action*, *role*)
Bases: object

class galaxy.model.**DeferredJob**(*state=None*, *plugin=None*, *params=None*)
Bases: object

check_interval

get_check_interval()

get_last_check()

is_check_time

last_check

set_check_interval(*seconds*)

set_last_check(*seconds*)

states = <galaxy.util.bunch.Bunch object>

class galaxy.model.**Event**(*message=None*, *history=None*, *user=None*, *galaxy_session=None*)
Bases: object

class galaxy.model.**ExtendedMetadata**(*data*)
Bases: object

class galaxy.model.**ExtendedMetadataIndex**(*extended_metadata*, *path*, *value*)
Bases: object

class galaxy.model.**ExternalService**(*name=None*, *description=None*, *external_service_type_id=None*, *version=None*, *form_definition_id=None*, *form_values_id=None*, *deleted=None*)
Bases: object

data_transfer_protocol = <galaxy.util.bunch.Bunch object>

get_external_service_type(*trans*)

load_data_transfer_settings(*trans*)

populate_actions(*trans*, *item*, *param_dict=None*)

class galaxy.model.**FormDefinition**(*name=None*, *desc=None*, *fields=[]*, *form_definition_current=None*, *form_type=None*, *layout=None*)
Bases: object, galaxy.model.item_attrs.Dictifiable

dict_collection_visible_keys = ('id', 'name')

dict_element_visible_keys = ('id', 'name', 'desc', 'form_definition_current_id', 'fields', 'layout')

field_as_html(*field*)
Generates disabled html for a field

get_widgets(*user*, *contents={}*, ***kwd*)
Return the list of widgets that comprise a form definition, including field contents if any.

grid_fields(*grid_index*)

supported_field_types = [<class 'galaxy.web.form_builder.AddressField'>, <class 'galaxy.web.form_builder.Checkl

types = <galaxy.util.bunch.Bunch object>

class galaxy.model.**FormDefinitionCurrent**(*form_definition=None*)
Bases: object

class galaxy.model.**FormValues**(*form_def=None*, *content=None*)
Bases: object

class galaxy.model.**GalaxySession**(*id=None*, *user=None*, *remote_host=None*, *remote_addr=None*, *referer=None*, *current_history=None*, *session_key=None*, *is_valid=False*, *prev_session_id=None*, *last_action=None*)

Bases: object

add_history(*history*, *association=None*)

get_disk_usage()

set_disk_usage(*bytes*)

total_disk_usage

class galaxy.model.**GalaxySessionToHistoryAssociation**(*galaxy_session*, *history*)

Bases: object

class galaxy.model.**GenomeIndexToolData**(*job=None*, *params=None*, *dataset=None*, *deferred_job=None*, *transfer_job=None*, *fasta_path=None*, *created_time=None*, *modified_time=None*, *dbkey=None*, *user=None*, *indexer=None*)

Bases: object

class galaxy.model.**Group**(*name=None*)

Bases: object, *galaxy.model.item_attrs.Dictifiable*

dict_collection_visible_keys = ('id', 'name')

dict_element_visible_keys = ('id', 'name')

class galaxy.model.**GroupQuotaAssociation**(*group*, *quota*)

Bases: object, *galaxy.model.item_attrs.Dictifiable*

dict_element_visible_keys = ('group',)

class galaxy.model.**GroupRoleAssociation**(*group*, *role*)

Bases: object

class galaxy.model.**HasJobMetrics**

add_metric(*plugin*, *metric_name*, *metric_value*)

metrics

class galaxy.model.**HasName**

get_display_name()

These objects have a name attribute can be either a string or a unicode object. If string, convert to unicode object assuming 'utf-8' format.

class galaxy.model.**History**(*id=None*, *name=None*, *user=None*)

Bases: object, *galaxy.model.item_attrs.Dictifiable*, *galaxy.model.item_attrs.UsesAnnotations*, *galaxy.model.HasName*

activatable_datasets

active_contents

Return all active contents ordered by hid.

active_datasets_children_and_roles

add_dataset(*dataset*, *parent_id=None*, *genome_build=None*, *set_hid=True*, *quota=True*)

add_dataset_collection(*history_dataset_collection*, *set_hid=True*)

add_galaxy_session(*galaxy_session*, *association=None*)

contents_iter(***kwds*)
Fetch filtered list of contents of history.

copy(*name=None*, *target_user=None*, *activatable=False*, *all_datasets=False*)
Return a copy of this history using the given *name* and *target_user*. If *activatable*, copy only non-deleted datasets. If *all_datasets*, copy non-deleted, deleted, and purged datasets.

copy_tags_from(*target_user*, *source_history*)

default_name = 'Unnamed history'

dict_collection_visible_keys = ('id', 'name', 'published', 'deleted')

dict_element_visible_keys = ('id', 'name', 'genome_build', 'deleted', 'purged', 'update_time', 'published', 'impo

empty

get_disk_size(*nice_size=False*)

get_disk_size_bytes

latest_export

resume_paused_jobs()

to_dict(*view='collection'*, *value_mapper=None*)

unhide_datasets()

class galaxy.model.**HistoryAnnotationAssociation**
Bases: object

class galaxy.model.**HistoryDatasetAssociation**(*hid=None*, *history=None*, *copied_from_history_dataset_association=None*, *copied_from_library_dataset_dataset_association=None*, *sa_session=None*, ***kwd*)
Bases: galaxy.model.DatasetInstance, galaxy.model.item_attrs.Dictifiable, galaxy.model.item_attrs.UsesAnnotations, galaxy.model.HasName

Resource class that creates a relation between a dataset and a user history.

clear_associated_files(*metadata_safe=False*, *purge=False*)

copy(*copy_children=False*, *parent_id=None*)
Create a copy of this HDA.

get_access_roles(*trans*)
Return The access roles associated with this HDA's dataset.

history_content_type

quota_amount(*user*)
Return the disk space used for this HDA relevant to user quotas.

If the user has multiple instances of this dataset, it will not affect their disk usage statistic.

to_dict(*view='collection'*, *expose_dataset_path=False*)
Return attributes of this HDA that are exposed using the API.

to_library_dataset_dataset_association(*trans*, *target_folder*, *replace_dataset=None*, *parent_id=None*, *user=None*, *roles=None*, *ldda_message=''*)
Copy this HDA to a library optionally replacing an existing LDDA.

class galaxy.model.**HistoryDatasetAssociationAnnotationAssociation**
Bases: object

class galaxy.model.**HistoryDatasetAssociationDisplayAtAuthorization**(*hda=None*, *user=None*, *site=None*)
Bases: object

class galaxy.model.**HistoryDatasetAssociationRatingAssociation**(*id=None*, *user=None*, *item=None*, *rating=0*)
Bases: *galaxy.model.ItemRatingAssociation*

set_item(*history_dataset_association*)

class galaxy.model.**HistoryDatasetAssociationSubset**(*hda*, *subset*, *location*)
Bases: object

class galaxy.model.**HistoryDatasetAssociationTagAssociation**(*id=None*, *user=None*, *item_id=None*, *tag_id=None*, *user_tname=None*, *value=None*)
Bases: *galaxy.model.ItemTagAssociation*

class galaxy.model.**HistoryDatasetCollectionAnnotationAssociation**
Bases: object

class galaxy.model.**HistoryDatasetCollectionAssociation**(*id=None*, *hid=None*, *collection=None*, *history=None*, *name=None*, *deleted=False*, *visible=True*, *copied_from_history_dataset_collection_association=N*
implicit_output_name=None, *implicit_input_collections=[]*)
Bases: *galaxy.model.DatasetCollectionInstance*, *galaxy.model.item_attrs.Dictifiable*

Associates a DatasetCollection with a History.

add_implicit_input_collection(*name*, *history_dataset_collection*)

copy(*element_destination=None*)
Create a copy of this history dataset collection association. Copy underlying collection.

editable_keys = ('name', 'deleted', 'visible')

find_implicit_input_collection(*name*)

history_content_type

to_dict(*view='collection'*)

class galaxy.model.**HistoryDatasetCollectionRatingAssociation**(*id=None*, *user=None*, *item=None*, *rating=0*)
Bases: *galaxy.model.ItemRatingAssociation*

set_item(*dataset_collection*)

class galaxy.model.**HistoryDatasetCollectionTagAssociation**(*id=None*, *user=None*, *item_id=None*, *tag_id=None*, *user_tname=None*, *value=None*)

Bases: *galaxy.model.ItemTagAssociation*

class galaxy.model.**HistoryRatingAssociation**(*id=None*, *user=None*, *item=None*, *rating=0*)
Bases: *galaxy.model.ItemRatingAssociation*

set_item(*history*)

class galaxy.model.**HistoryTagAssociation**(*id=None*, *user=None*, *item_id=None*, *tag_id=None*, *user_tname=None*, *value=None*)
Bases: *galaxy.model.ItemTagAssociation*

class galaxy.model.**HistoryUserShareAssociation**
Bases: object

class galaxy.model.**ImplicitlyConvertedDatasetAssociation**(*id=None*, *parent=None*, *dataset=None*, *file_type=None*, *deleted=False*, *purged=False*, *metadata_safe=True*)
Bases: object

clear(*purge=False*, *delete_dataset=True*)

class galaxy.model.**ImplicitlyCreatedDatasetCollectionInput**(*name*, *input_dataset_collection*)
Bases: object

class galaxy.model.**ItemRatingAssociation**(*id=None*, *user=None*, *item=None*, *rating=0*)
Bases: object

set_item(*item*)
Set association's item.

class galaxy.model.**ItemTagAssociation**(*id=None*, *user=None*, *item_id=None*, *tag_id=None*, *user_tname=None*, *value=None*)
Bases: object, *galaxy.model.item_attrs.Dictifiable*

copy()

dict_collection_visible_keys = ('id', 'user_tname', 'user_value')

dict_element_visible_keys = ('id', 'user_tname', 'user_value')

class galaxy.model.**Job**
Bases: object, *galaxy.model.HasJobMetrics*, *galaxy.model.item_attrs.Dictifiable*

add_implicit_output_dataset_collection(*name*, *dataset_collection*)

add_input_dataset(*name*, *dataset*)

add_input_dataset_collection(*name*, *dataset*)

add_input_library_dataset(*name*, *dataset*)

add_output_dataset(*name*, *dataset*)

add_output_dataset_collection(*name*, *dataset_collection_instance*)

add_output_library_dataset(*name*, *dataset*)

add_parameter(*name*, *value*)

add_post_job_action(*pja*)

check_if_output_datasets_deleted()
Return true if all of the output datasets associated with this job are in the deleted state

dict_collection_visible_keys = ['id', 'state', 'exit_code', 'update_time', 'create_time']

dict_element_visible_keys = ['id', 'state', 'exit_code', 'update_time', 'create_time']

A job represents a request to run a tool given input datasets, tool parameters, and output datasets.

finished

get_command_line()

get_external_output_metadata()

The external_output_metadata is currently a reference from Job to JobExternalOutputMetadata. It exists for a job but not a task.

get_handler()

get_id()

get_id_tag()

Return a tag that can be useful in identifying a Job. This returns the Job's get_id

get_imported()

get_info()

get_input_datasets()

get_input_library_datasets()

get_job()

get_job_runner_external_id()

get_job_runner_name()

get_output_datasets()

get_output_library_datasets()

get_param_filename()

get_param_values(*app*, *ignore_errors=False*)

Read encoded parameter values from the database and turn back into a dict of tool parameter values.

get_parameters()

get_params()

get_post_job_actions()

get_session_id()

get_state()

get_tasks()

get_tool_id()

get_tool_version()

get_user()

get_user_id()

mark_deleted(*track_jobs_in_database=False*)

Mark this job as deleted, and mark any output datasets as discarded.

raw_param_dict()

set_command_line(*command_line*)

set_final_state(*final_state*)

set_handler(*handler*)

set_imported(*imported*)

set_info(*info*)

set_input_datasets(*input_datasets*)

set_input_library_datasets(*input_library_datasets*)

set_output_datasets(*output_datasets*)

set_output_library_datasets(*output_library_datasets*)

set_param_filename(*param_filename*)

set_parameters(*parameters*)

set_params(*params*)

set_post_job_actions(*post_job_actions*)

set_runner_external_id(*job_runner_external_id*)

set_runner_name(*job_runner_name*)

set_session_id(*session_id*)

set_state(*state*)
Save state history

set_tool_id(*tool_id*)

set_tool_version(*tool_version*)

set_user_id(*user_id*)

states = <galaxy.util.bunch.Bunch object>

to_dict(*view='collection'*, *system_details=False*)

class galaxy.model.**JobExportHistoryArchive**(*job=None*, *history=None*, *dataset=None*, *compressed=False*, *history_attrs_filename=None*, *datasets_attrs_filename=None*, *jobs_attrs_filename=None*)
Bases: object

export_name

preparing

ready

up_to_date
Return False, if a new export should be generated for corresponding history.

class galaxy.model.**JobExternalOutputMetadata**(*job=None*, *dataset=None*)
Bases: object

dataset

class galaxy.model.**JobImportHistoryArchive**(*job=None*, *history=None*, *archive_dir=None*)
Bases: object

class galaxy.model.**JobMetricNumeric**(*plugin*, *metric_name*, *metric_value*)
Bases: *galaxy.model.BaseJobMetric*

class galaxy.model.**JobMetricText**(*plugin*, *metric_name*, *metric_value*)
Bases: galaxy.model.BaseJobMetric

class galaxy.model.**JobParameter**(*name*, *value*)
Bases: object

class galaxy.model.**JobStateHistory**(*job*)
Bases: object

class galaxy.model.**JobToImplicitOutputDatasetCollectionAssociation**(*name*, *dataset_collection*)
Bases: object

class galaxy.model.**JobToInputDatasetAssociation**(*name*, *dataset*)
Bases: object

class galaxy.model.**JobToInputDatasetCollectionAssociation**(*name*, *dataset*)
Bases: object

class galaxy.model.**JobToInputLibraryDatasetAssociation**(*name*, *dataset*)
Bases: object

class galaxy.model.**JobToOutputDatasetAssociation**(*name*, *dataset*)
Bases: object

class galaxy.model.**JobToOutputDatasetCollectionAssociation**(*name*, *dataset_collection_instance*)
Bases: object

class galaxy.model.**JobToOutputLibraryDatasetAssociation**(*name*, *dataset*)
Bases: object

class galaxy.model.**Library**(*name=None*, *description=None*, *synopsis=None*, *root_folder=None*)
Bases: object, galaxy.model.item_attrs.Dictifiable, galaxy.model.HasName

dict_collection_visible_keys = ('id', 'name')

dict_element_visible_keys = ('id', 'deleted', 'name', 'description', 'synopsis', 'root_folder_id')

get_access_roles(*trans*)

get_active_folders(*folder*, *folders=None*)

get_info_association(*restrict=False*, *inherited=False*)

get_template_widgets(*trans*, *get_contents=True*)

permitted_actions = <galaxy.util.bunch.Bunch object>

to_dict(*view='collection'*, *value_mapper=None*)
We prepend an F to folders.

class galaxy.model.**LibraryDataset**(*folder=None*, *order_id=None*, *name=None*, *info=None*, *library_dataset_dataset_association=None*, ***kwd*)
Bases: object

display_name()

get_info()

get_name()

info

name

set_info(*info*)

set_library_dataset_dataset_association(*ldda*)

set_name(*name*)

to_dict(*view='collection'*)

upload_options = [('upload_file', 'Upload files'), ('upload_directory', 'Upload directory of files'), ('upload_paths', 'Up

class galaxy.model.**LibraryDatasetCollectionAnnotationAssociation**

Bases: object

class galaxy.model.**LibraryDatasetCollectionAssociation**(*id=None, collection=None, name=None, deleted=False, folder=None*)

Bases: galaxy.model.DatasetCollectionInstance, galaxy.model.item_attrs.Dictifiable

Associates a DatasetCollection with a library folder.

editable_keys = ('name', 'deleted')

to_dict(*view='collection'*)

class galaxy.model.**LibraryDatasetCollectionRatingAssociation**(*id=None, user=None, item=None, rating=0*)

Bases: galaxy.model.ItemRatingAssociation

set_item(*dataset_collection*)

class galaxy.model.**LibraryDatasetCollectionTagAssociation**(*id=None, user=None, item_id=None, tag_id=None, user_tname=None, value=None*)

Bases: galaxy.model.ItemTagAssociation

class galaxy.model.**LibraryDatasetDatasetAssociation**(*copied_from_history_dataset_association=None, copied_from_library_dataset_dataset_association=None, library_dataset=None, user=None, sa_session=None, **kwd*)

Bases: galaxy.model.DatasetInstance, galaxy.model.HasName

clear_associated_files(*metadata_safe=False, purge=False*)

copy(*copy_children=False, parent_id=None, target_folder=None*)

get_access_roles(*trans*)

get_info_association(*restrict=False, inherited=False*)

get_manage_permissions_roles(*trans*)

get_template_widgets(*trans, get_contents=True*)

has_manage_permissions_roles(*trans*)

templates_dict(*use_name=False*)

Returns a dict of template info

templates_json(*use_name=False*)

to_dict(*view='collection'*)

to_history_dataset_association(*target_history, parent_id=None, add_to_history=False*)

class galaxy.model.**LibraryDatasetDatasetAssociationPermissions**(*action, library_item, role*)

Bases: object

class galaxy.model.**LibraryDatasetDatasetInfoAssociation**(*library_dataset_dataset_association*, *form_definition*, *info*)

Bases: object

inheritable

class galaxy.model.**LibraryDatasetPermissions**(*action*, *library_item*, *role*)

Bases: object

class galaxy.model.**LibraryFolder**(*name=None*, *description=None*, *item_count=0*, *order_id=None*)

Bases: object, galaxy.model.item_attrs.Dictifiable, galaxy.model.HasName

activatable_library_datasets

add_folder(*folder*)

add_library_dataset(*library_dataset*, *genome_build=None*)

dict_element_visible_keys = ('id', 'parent_id', 'name', 'description', 'item_count', 'genome_build', 'update_time

get_info_association(*restrict=False*, *inherited=False*)

get_template_widgets(*trans*, *get_contents=True*)

library_path

parent_library

to_dict(*view='collection'*, *value_mapper=None*)

class galaxy.model.**LibraryFolderInfoAssociation**(*folder*, *form_definition*, *info*, *inheritable=False*)

Bases: object

class galaxy.model.**LibraryFolderPermissions**(*action*, *library_item*, *role*)

Bases: object

class galaxy.model.**LibraryInfoAssociation**(*library*, *form_definition*, *info*, *inheritable=False*)

Bases: object

class galaxy.model.**LibraryPermissions**(*action*, *library_item*, *role*)

Bases: object

class galaxy.model.**MetadataFile**(*dataset=None*, *name=None*)

Bases: object

file_name

exception galaxy.model.**NoConverterException**(*value*)

Bases: exceptions.Exception

class galaxy.model.**Page**

Bases: object, galaxy.model.item_attrs.Dictifiable

dict_element_visible_keys = ['id', 'title', 'latest_revision_id', 'slug', 'published', 'importable', 'deleted']

to_dict(*view='element'*)

class galaxy.model.**PageAnnotationAssociation**

Bases: object

class galaxy.model.**PageRatingAssociation**(*id=None*, *user=None*, *item=None*, *rating=0*)

Bases: galaxy.model.ItemRatingAssociation

set_item(*page*)

class galaxy.model.**PageRevision**
Bases: object, galaxy.model.item_attrs.Dictifiable

dict_element_visible_keys = ['id', 'page_id', 'title', 'content']

to_dict(*view='element'*)

class galaxy.model.**PageTagAssociation**(*id=None, user=None, item_id=None, tag_id=None, user_tname=None, value=None*)
Bases: galaxy.model.ItemTagAssociation

class galaxy.model.**PageUserShareAssociation**
Bases: object

class galaxy.model.**PasswordResetToken**(*user, token=None*)
Bases: object

class galaxy.model.**PostJobAction**(*action_type, workflow_step, output_name=None, action_arguments=None*)
Bases: object

class galaxy.model.**PostJobActionAssociation**(*pja, job*)
Bases: object

class galaxy.model.**Quota**(*name='', description='', amount=0, operation='='*)
Bases: object, galaxy.model.item_attrs.Dictifiable

amount

dict_collection_visible_keys = ('id', 'name')

dict_element_visible_keys = ('id', 'name', 'description', 'bytes', 'operation', 'display_amount', 'default', 'users',

display_amount

get_amount()

set_amount(*amount*)

valid_operations = ('+', '-', '=')

class galaxy.model.**Request**(*name=None, desc=None, request_type=None, user=None, form_values=None, notification=None*)
Bases: object, galaxy.model.item_attrs.Dictifiable

dict_collection_visible_keys = ('id', 'name', 'state')

get_sample(*sample_name*)

is_complete

is_new

is_rejected

is_submitted

is_unsubmitted

last_comment

latest_event

samples_have_common_state
Returns the state of this request's samples when they are all in one common state. Otherwise returns False.

samples_with_bar_code

samples_without_library_destinations

send_email_notification(*trans*, *common_state*, *final_state=False*)

state

states = <galaxy.util.bunch.Bunch object>

class galaxy.model.**RequestEvent**(*request=None*, *request_state=None*, *comment=''*)
Bases: object

class galaxy.model.**RequestType**(*name=None*, *desc=None*, *request_form=None*, *sample_form=None*)
Bases: object, galaxy.model.item_attrs.Dictifiable

add_external_service_association(*trans*, *external_service*)

delete_external_service_associations(*trans*)
Deletes all external service associations.

dict_collection_visible_keys = ('id', 'name', 'desc')

dict_element_visible_keys = ('id', 'name', 'desc', 'request_form_id', 'sample_form_id')

external_services

final_sample_state

get_external_service(*external_service_type_id*)

get_external_services_for_manual_data_transfer(*trans*)
Returns all external services that use manual data transfer

get_template_widgets(*trans*, *get_contents=True*)

permitted_actions = <galaxy.util.bunch.Bunch object>

rename_dataset_options = <galaxy.util.bunch.Bunch object>

run_details

class galaxy.model.**RequestTypeExternalServiceAssociation**(*request_type*, *external_service*)
Bases: object

class galaxy.model.**RequestTypePermissions**(*action*, *request_type*, *role*)
Bases: object

class galaxy.model.**RequestTypeRunAssociation**(*request_type*, *run*)
Bases: object

class galaxy.model.**Role**(*name=''*, *description=''*, *type='system'*, *deleted=False*)
Bases: object, galaxy.model.item_attrs.Dictifiable

dict_collection_visible_keys = ('id', 'name')

dict_element_visible_keys = ('id', 'name', 'description', 'type')

private_id = None

types = <galaxy.util.bunch.Bunch object>

class galaxy.model.**Run**(*form_definition*, *form_values*, *subindex=None*)
Bases: object

class galaxy.model.**Sample**(*name=None*, *desc=None*, *request=None*, *form_values=None*, *bar_code=None*, *library=None*, *folder=None*, *workflow=None*, *history=None*)
Bases: object, galaxy.model.item_attrs.Dictifiable

adding_to_library_dataset_files

bulk_operations = <galaxy.util.bunch.Bunch object>

dict_collection_visible_keys = ('id', 'name')

get_template_widgets(*trans*, *get_contents=True*)

get_untransferred_dataset_size(*filepath*, *scp_configs*)

inprogress_dataset_files

latest_event

populate_external_services(*param_dict=None*, *trans=None*)

queued_dataset_files

run_details

state

supported_field_types = [<class 'galaxy.web.form_builder.CheckboxField'>, <class 'galaxy.web.form_builder.Selec

transfer_error_dataset_files

transferred_dataset_files

transferring_dataset_files

untransferred_dataset_files

class galaxy.model.**SampleDataset**(*sample=None*, *name=None*, *file_path=None*, *status=None*, *error_msg=None*, *size=None*, *external_service=None*)

Bases: object

transfer_status = <galaxy.util.bunch.Bunch object>

class galaxy.model.**SampleEvent**(*sample=None*, *sample_state=None*, *comment=''*)

Bases: object

class galaxy.model.**SampleRunAssociation**(*sample*, *run*)

Bases: object

class galaxy.model.**SampleState**(*name=None*, *desc=None*, *request_type=None*)

Bases: object

class galaxy.model.**StoredWorkflow**

Bases: object, *galaxy.model.item_attrs.Dictifiable*

copy_tags_from(*target_user*, *source_workflow*)

dict_collection_visible_keys = ('id', 'name', 'published', 'deleted')

dict_element_visible_keys = ('id', 'name', 'published', 'deleted')

to_dict(*view='collection'*, *value_mapper=None*)

class galaxy.model.**StoredWorkflowAnnotationAssociation**

Bases: object

class galaxy.model.**StoredWorkflowMenuEntry**

Bases: object

class galaxy.model.**StoredWorkflowRatingAssociation**(*id=None*, *user=None*, *item=None*, *rating=0*)

Bases: *galaxy.model.ItemRatingAssociation*

set_item(*stored_workflow*)

class galaxy.model.**StoredWorkflowTagAssociation**(*id=None*, *user=None*, *item_id=None*, *tag_id=None*, *user_tname=None*, *value=None*)

Bases: galaxy.model.ItemTagAssociation

class galaxy.model.**StoredWorkflowUserShareAssociation**

Bases: object

class galaxy.model.**Tag**(*id=None*, *type=None*, *parent_id=None*, *name=None*)

Bases: object

class galaxy.model.**Task**(*job*, *working_directory*, *prepare_files_cmd*)

Bases: object, galaxy.model.HasJobMetrics

A task represents a single component of a job.

get_command_line()

get_external_output_metadata()

The external_output_metadata is currently a backref to JobExternalOutputMetadata. It exists for a job but not a task, and when a task is cancelled its corresponding parent Job will be cancelled. So None is returned now, but that could be changed to self.get_job().get_external_output_metadata().

get_id()

get_id_tag()

Return an id tag suitable for identifying the task. This combines the task's job id and the task's own id.

get_info()

get_job()

get_job_runner_external_id()

Runners will use the same methods to get information about the Task class as they will about the Job class, so this method just returns the task's external id.

get_job_runner_name()

Since runners currently access Tasks the same way they access Jobs, this method just refers to *this* instance's runner.

get_param_values(*app*)

Read encoded parameter values from the database and turn back into a dict of tool parameter values.

get_parameters()

get_prepare_input_files_cmd()

get_session_id()

get_state()

get_stderr()

get_stdout()

get_task_runner_external_id()

get_task_runner_name()

get_working_directory()

set_command_line(*command_line*)

set_id(*id*)

set_info(*info*)

set_job(*job*)

set_job_runner_external_id(*task_runner_external_id*)

set_parameters(*parameters*)

set_prepare_input_files_cmd(*prepare_input_files_cmd*)

set_state(*state*)

set_stderr(*stderr*)

set_stdout(*stdout*)

set_task_runner_external_id(*task_runner_external_id*)

set_task_runner_name(*task_runner_name*)

set_working_directory(*working_directory*)

states = <galaxy.util.bunch.Bunch object>

class galaxy.model.**TaskMetricNumeric**(*plugin*, *metric_name*, *metric_value*)
Bases: *galaxy.model.BaseJobMetric*

class galaxy.model.**TaskMetricText**(*plugin*, *metric_name*, *metric_value*)
Bases: *galaxy.model.BaseJobMetric*

class galaxy.model.**ToolTagAssociation**(*id=None*, *user=None*, *tool_id=None*, *tag_id=None*, *user_tname=None*, *value=None*)
Bases: *galaxy.model.ItemTagAssociation*

class galaxy.model.**TransferJob**(*state=None*, *path=None*, *info=None*, *pid=None*, *socket=None*, *params=None*)
Bases: object

states = <galaxy.util.bunch.Bunch object>

terminal_states = ['error', 'done']

class galaxy.model.**UCI**
Bases: object

class galaxy.model.**User**(*email=None*, *password=None*)
Bases: object, *galaxy.model.item_attrs.Dictifiable*

all_roles()
Return a unique list of Roles associated with this user or any of their groups.

calculate_disk_usage()
Return byte count total of disk space used by all non-purged, non-library HDAs in non-purged histories.

check_password(*cleartext*)
Check if *cleartext* matches user password when hashed.

dict_collection_visible_keys = ('id', 'email', 'username')

dict_element_visible_keys = ('id', 'email', 'username', 'total_disk_usage', 'nice_total_disk_usage')

static expand_user_properties(*user*, *in_string*)

get_disk_usage(*nice_size=False*)
Return byte count of disk space used by user or a human-readable string if *nice_size* is *True*.

nice_total_disk_usage
Return byte count of disk space used in a human-readable string.

set_disk_usage(*bytes*)
Manually set the disk space used by a user to *bytes*.

set_password_cleartext(*cleartext*)
Set user password to the digest of *cleartext*.

total_disk_usage
Return byte count of disk space used by user or a human-readable string if *nice_size* is *True*.

use_pbkdf2 = True
Data for a Galaxy user or admin and relations to their histories, credentials, and roles.

static **user_template_environment**(*user*)

```
>>> env = User.user_template_environment(None)
>>> env['__user_email__']
'Anonymous'
>>> env['__user_id__']
'Anonymous'
>>> user = User('foo@example.com')
>>> user.id = 6
>>> user.username = 'foo2'
>>> env = User.user_template_environment(user)
>>> env['__user_id__']
'6'
>>> env['__user_name__']
'foo2'
```

class galaxy.model.**UserAction**(*id=None*, *create_time=None*, *user_id=None*, *session_id=None*, *action=None*, *params=None*, *context=None*)
Bases: object

class galaxy.model.**UserAddress**(*user=None*, *desc=None*, *name=None*, *institution=None*, *address=None*, *city=None*, *state=None*, *postal_code=None*, *country=None*, *phone=None*)
Bases: object

get_html()

class galaxy.model.**UserGroupAssociation**(*user*, *group*)
Bases: object

class galaxy.model.**UserOpenID**(*user=None*, *session=None*, *openid=None*)
Bases: object

class galaxy.model.**UserPreference**(*name=None*, *value=None*)
Bases: object

class galaxy.model.**UserQuotaAssociation**(*user*, *quota*)
Bases: object, *galaxy.model.item_attrs.Dictifiable*

dict_element_visible_keys = ('user',)

class galaxy.model.**UserRoleAssociation**(*user*, *role*)
Bases: object

class galaxy.model.**ValidationError**(*message=None*, *err_type=None*, *attributes=None*)
Bases: object

class galaxy.model.**Visualization**(*id=None*, *user=None*, *type=None*, *title=None*, *dbkey=None*, *slug=None*, *latest_revision=None*)
Bases: object

copy (*user=None*, *title=None*)
Provide copy of visualization with only its latest revision.

class galaxy.model.**VisualizationAnnotationAssociation**
Bases: object

class galaxy.model.**VisualizationRatingAssociation** (*id=None*, *user=None*, *item=None*, *rating=0*)
Bases: galaxy.model.ItemRatingAssociation

set_item (*visualization*)

class galaxy.model.**VisualizationRevision** (*visualization=None*, *title=None*, *dbkey=None*, *config=None*)
Bases: object

copy (*visualization=None*)
Returns a copy of this object.

class galaxy.model.**VisualizationTagAssociation** (*id=None*, *user=None*, *item_id=None*, *tag_id=None*, *user_tname=None*, *value=None*)
Bases: galaxy.model.ItemTagAssociation

class galaxy.model.**VisualizationUserShareAssociation**
Bases: object

class galaxy.model.**WorkRequestTagAssociation** (*id=None*, *user=None*, *workflow_request_id=None*, *tag_id=None*, *user_tname=None*, *value=None*)
Bases: galaxy.model.ItemTagAssociation

class galaxy.model.**Workflow** (*uuid=None*)
Bases: object, galaxy.model.item_attrs.Dictifiable

dict_collection_visible_keys = ('name', 'has_cycles', 'has_errors')

dict_element_visible_keys = ('name', 'has_cycles', 'has_errors')

has_outputs_defined ()
Returns true or false indicating whether or not a workflow has outputs defined.

to_dict (*view='collection'*, *value_mapper=None*)

class galaxy.model.**WorkflowInvocation**
Bases: object, galaxy.model.item_attrs.Dictifiable

active
Indicates the workflow invocation is somehow active - and in particular valid actions may be performed on its ''WorkflowInvocationStep''s.

add_input (*content*, *step_id*)

cancel ()

dict_collection_visible_keys = ('id', 'update_time', 'workflow_id', 'history_id', 'uuid', 'state')

dict_element_visible_keys = ('id', 'update_time', 'workflow_id', 'history_id', 'uuid', 'state')

fail ()

has_input_for_step (*step_id*)

static **poll_active_workflow_ids** (*sa_session*, *scheduler=None*, *handler=None*)

states = <galaxy.util.bunch.Bunch object>

step_invocations_by_step_id()

step_invocations_for_step_id(*step_id*)

step_states_by_step_id()

to_dict(*view='collection'*, *value_mapper=None*)

update()

class galaxy.model.**WorkflowInvocationStep**
Bases: object, *galaxy.model.item_attrs.Dictifiable*

dict_collection_visible_keys = ('id', 'update_time', 'job_id', 'workflow_step_id', 'action')

dict_element_visible_keys = ('id', 'update_time', 'job_id', 'workflow_step_id', 'action')

to_dict(*view='collection'*, *value_mapper=None*)

update()

class galaxy.model.**WorkflowOutput**(*workflow_step*, *output_name*)
Bases: object

class galaxy.model.**WorkflowRequest**
Bases: object, *galaxy.model.item_attrs.Dictifiable*

dict_collection_visible_keys = ['id', 'name', 'type', 'state', 'history_id', 'workflow_id']

dict_element_visible_keys = ['id', 'name', 'type', 'state', 'history_id', 'workflow_id']

to_dict(*view='collection'*, *value_mapper=None*)

class galaxy.model.**WorkflowRequestInputParameter**(*name=None*, *value=None*, *type=None*)
Bases: object, *galaxy.model.item_attrs.Dictifiable*

Workflow-related parameters not tied to steps or inputs.

dict_collection_visible_keys = ['id', 'name', 'value', 'type']

types = <galaxy.util.bunch.Bunch object>

class galaxy.model.**WorkflowRequestStepState**(*workflow_step=None*, *name=None*, *value=None*)
Bases: object, *galaxy.model.item_attrs.Dictifiable*

Workflow step value parameters.

dict_collection_visible_keys = ['id', 'name', 'value', 'workflow_step_id']

class galaxy.model.**WorkflowRequestToInputDatasetAssociation**
Bases: object, *galaxy.model.item_attrs.Dictifiable*

Workflow step input dataset parameters.

dict_collection_visible_keys = ['id', 'workflow_invocation_id', 'workflow_step_id', 'dataset_id', 'name']

class galaxy.model.**WorkflowRequestToInputDatasetCollectionAssociation**
Bases: object, *galaxy.model.item_attrs.Dictifiable*

Workflow step input dataset collection parameters.

dict_collection_visible_keys = ['id', 'workflow_invocation_id', 'workflow_step_id', 'dataset_collection_id', 'na

class galaxy.model.**WorkflowStep**
Bases: object

class galaxy.model.**WorkflowStepAnnotationAssociation**
Bases: object

class galaxy.model.**WorkflowStepConnection**

Bases: object

NON_DATA_CONNECTION = '__NO_INPUT_OUTPUT_NAME__'

non_data_connection

set_non_data_connection()

class galaxy.model.**WorkflowStepTagAssociation**(*id=None*, *user=None*, *item_id=None*, *tag_id=None*, *user_tname=None*, *value=None*)

Bases: *galaxy.model.ItemTagAssociation*

galaxy.model.**set_datatypes_registry**(*d_registry*)

Set up datatypes_registry

custom_types Module

class galaxy.model.custom_types.**JSONType**(**args*, ***kwargs*)

Bases: sqlalchemy.sql.type_api.TypeDecorator

Represents an immutable structure as a json-encoded string.

If default is, for example, a dict, then a NULL value in the database will be exposed as an empty dict.

compare_values(*x*, *y*)

copy_value(*value*)

impl

alias of LargeBinary

load_dialect_impl(*dialect*)

process_bind_param(*value*, *dialect*)

process_result_value(*value*, *dialect*)

class galaxy.model.custom_types.**MetadataType**(**args*, ***kwargs*)

Bases: *galaxy.model.custom_types.JSONType*

Backward compatible metadata type. Can read pickles or JSON, but always writes in JSON.

process_result_value(*value*, *dialect*)

class galaxy.model.custom_types.**MutationDict**

Bases: *galaxy.model.custom_types.MutationObj*, dict

classmethod coerce(*key*, *value*)

Convert plain dictionary to MutationDict

class galaxy.model.custom_types.**MutationList**

Bases: *galaxy.model.custom_types.MutationObj*, list

append(*value*)

classmethod coerce(*key*, *value*)

Convert plain list to MutationList

extend(*values*)

insert(*idx*, *value*)

pop(**args*, ***kw*)

remove(*value*)

class galaxy.model.custom_types.**MutationObj**
Bases: sqlalchemy.ext.mutable.Mutable

Mutable JSONType for SQLAlchemy from original gist: https://gist.github.com/dbarnett/1730610

Using minor changes from this fork of the gist: https://gist.github.com/miracle2k/52a031cced285ba9b8cd

And other minor changes to make it work for us.

classmethod coerce(*key*, *value*)

class galaxy.model.custom_types.**TrimmedString**(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

impl
alias of String

process_bind_param(*value*, *dialect*)
Automatically truncate string values

class galaxy.model.custom_types.**UUIDType**(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

Platform-independent UUID type.

Based on http://docs.sqlalchemy.org/en/rel_0_8/core/types.html#backend-agnostic-guid-type Changed to remove sqlalchemy 0.8 specific code

CHAR(32), storing as stringified hex values.

impl
alias of CHAR

load_dialect_impl(*dialect*)

process_bind_param(*value*, *dialect*)

process_result_value(*value*, *dialect*)

item_attrs Module

class galaxy.model.item_attrs.**Dictifiable**
Mixin that enables objects to be converted to dictionaries. This is useful when for sharing objects across boundaries, such as the API, tool scripts, and JavaScript code.

to_dict(*view='collection'*, *value_mapper=None*)
Return item dictionary.

exception galaxy.model.item_attrs.**RuntimeException**
Bases: exceptions.Exception

class galaxy.model.item_attrs.**UsesAnnotations**
Mixin for getting and setting item annotations.

add_item_annotation(*db_session*, *user*, *item*, *annotation*)
Add or update an item's annotation; a user can only have a single annotation for an item.

copy_item_annotation(*db_session*, *source_user*, *source_item*, *target_user*, *target_item*)
Copy an annotation from a user/item source to a user/item target.

delete_item_annotation(*db_session*, *user*, *item*)

get_item_annotation_obj(*db_session*, *user*, *item*)
Returns a user's annotation object for an item.

get_item_annotation_str(*db_session*, *user*, *item*)
Returns a user's annotation string for an item.

class galaxy.model.item_attrs.**UsesItemRatings**
Mixin for getting and setting item ratings.

Class makes two assumptions: (1) item-rating association table is named <item_class>RatingAssocation (2) item-rating association table has a column with a foreign key referencing item table that contains the item's id.

get_ave_item_rating_data(*db_session*, *item*, *webapp_model=None*)
Returns the average rating for an item.

get_user_item_rating(*db_session*, *user*, *item*, *webapp_model=None*)
Returns user's rating for an item. Return type is <item_class>RatingAssociation.

rate_item(*db_session*, *user*, *item*, *rating*, *webapp_model=None*)
Rate an item. Return type is <item_class>RatingAssociation.

mapping Module Details of how the data model objects are mapped onto the relational database are encapsulated here.

galaxy.model.mapping.**annotation_mapping**(*annotation_class*, ***kwds*)

galaxy.model.mapping.**db_next_hid**(*self*)
Override __next_hid to generate from the database in a concurrency safe way. Loads the next history ID from the DB and returns it. It also saves the future next_id into the DB.

Return type *int*

Returns the next history id

galaxy.model.mapping.**init**(*file_path*, *url*, *engine_options={}*, *create_tables=False*, *map_install_models=False*, *database_query_profiling_proxy=False*, *object_store=None*, *trace_logger=None*, *use_pbkdf2=True*)
Connect mappings to the database

galaxy.model.mapping.**now**()
Return a new datetime representing UTC day and time.

galaxy.model.mapping.**rating_mapping**(*rating_class*, ***kwds*)

galaxy.model.mapping.**simple_mapping**(*model*, ***kwds*)

galaxy.model.mapping.**tag_mapping**(*tag_association_class*, *backref_name*)

mapping_tests Module

Subpackages

migrate Package

check Module

galaxy.model.migrate.check.**create_or_verify_database**(*url*, *galaxy_config_file*, *engine_options={}*, *app=None*)
Check that the database is use-able, possibly creating it if empty (this is the only time we automatically create tables, otherwise we force the user to do it using the management script so they can create backups).

1.Empty database –> initialize with latest version and return
2.Database older than migration support –> fail and require manual update

3. Database at state where migrate support introduced –> add version control information but make no changes (might still require manual update)
4. Database versioned but out of date –> fail with informative message, user must run "sh manage_db.sh upgrade"

galaxy.model.migrate.check.**migrate_to_current_version**(*engine*, *schema*)

orm Package

orm Package

galaxy.model.orm.**load_egg_for_url**(*url*)

logging_connection_proxy Module

class galaxy.model.orm.logging_connection_proxy.**LoggingProxy**

Bases: sqlalchemy.interfaces.ConnectionProxy

Logs SQL statements using standard logging module

begin(*conn*, *begin*)

commit(*conn*, *commit*)

cursor_execute(*execute*, *cursor*, *statement*, *parameters*, *context*, *executemany*)

rollback(*conn*, *rollback*)

class galaxy.model.orm.logging_connection_proxy.**TraceLoggerProxy**(*trace_logger*)

Bases: sqlalchemy.interfaces.ConnectionProxy

Logs SQL statements using a metlog client

cursor_execute(*execute*, *cursor*, *statement*, *parameters*, *context*, *executemany*)

galaxy.model.orm.logging_connection_proxy.**pretty_stack**()

galaxy.model.orm.logging_connection_proxy.**stripwd**(*s*)

Subpackages

ext Package

ext Package

assignmapper Module

managers Package

managers Package Classes that manage resources (models, tools, etc.) by using the current Transaction.

Encapsulates the intersection of trans (or trans.sa_session), models, and Controllers.

Responsibilities: model operations that involve the trans/sa_session (CRUD) security:

ownership, accessibility

common aspect-oriented operations via new mixins: sharable, annotatable, tagable, ratable

Not responsible for: encoding/decoding ids any http gobblygook formatting of returned data (always python structures) formatting of raised errors

The goal is to have Controllers only handle: query-string/payload parsing and encoding/decoding ids http return formatting

and: control, improve namespacing in Controllers DRY for Controller ops (define here - use in both UI/API Controllers)

In other words, 'Business logic' independent of web transactions/user context (trans) should be pushed into models - but logic that requires the context trans should be placed under this module.

`api_keys` Module

class `galaxy.managers.api_keys.`**`ApiKeyManager`**(*app*)

Bases: `object`

`create_api_key`(*user*)

`get_or_create_api_key`(*user*)

`base` Module Keeps the older BaseController security and fetching methods and also defines a base ModelManager, ModelSerializer, and ModelDeserializer.

ModelManagers are used for operations on models that occur outside the scope of a single model object, such as:

- object creation
- object lookup
- interactions between 2+ objects of different model classes

(Since these were to replace model Mixins from web/framework/base/controller.py the rule of thumb used there also generally has been applied here: if it uses the trans or sa_session, put it in a manager and not the model.)

ModelSerializers allow flexible conversion of model objects to dictionaries. They control what keys are sent, how values are simplified, can remap keys, and allow both predefined and user controlled key sets.

ModelDeserializers control how a model validates and process an incoming attribute change to a model object.

class `galaxy.managers.base.`**`ModelDeserializer`**(*app*)

Bases: `object`

An object that converts an incoming serialized dict into values that can be directly assigned to an item's attributes and assigns them.

`add_deserializers`()

Register a map of attribute keys -> functions that will deserialize data into attributes to be assigned to the item.

`default_deserializer`(*item*, *key*, *val*, ***context*)

If the incoming *val* is different than the *item* value change it and, in either case, return the value.

`deserialize`(*item*, *data*, *flush=True*, ***context*)

Convert an incoming serialized dict into values that can be directly assigned to an item's attributes and assign them

`deserialize_basestring`(*item*, *key*, *val*, *convert_none_to_empty=False*, ***context*)

`deserialize_bool`(*item*, *key*, *val*, ***context*)

`deserialize_genome_build`(*item*, *key*, *val*, ***context*)

Make sure *val* is a valid dbkey and assign it.

deserialize_int(*item*, *key*, *val*, *min=None*, *max=None*, ***context*)

model_manager_class = None
the class used to create this deserializer's generically accessible model_manager

exception galaxy.managers.base.**ModelDeserializingError**(*err_msg=None*, *type='info'*, ***extra_error_info*)
Bases: *galaxy.exceptions.ObjectAttributeInvalidException*

Thrown when an incoming value isn't usable by the model (bad type, out of range, etc.)

class galaxy.managers.base.**ModelFilterParser**(*app*)
Bases: object

Converts string tuples (partially converted query string params) of attr, op, val into either:
•ORM based filters (filters that can be applied by the ORM at the SQL

level) or - functional filters (filters that use derived values or values not within the SQL tables)
These filters can then be applied to queries.

This abstraction allows 'smarter' application of limit and offset at either the SQL level or the generator/list level based on the presence of functional filters. In other words, if no functional filters are present, limit and offset may be applied at the SQL level. If functional filters are present, limit and offset need to applied at the list level.

These might be safely be replaced in the future by creating SQLAlchemy hybrid properties or more thoroughly mapping derived values.

UNDERSCORED_OPS = ('lt', 'le', 'eq', 'ne', 'ge', 'gt')
these are the easier/shorter string equivalents to the python operator fn names that need '__' around them

fn_filter_parsers = None
dictionary containing parsing data for functional filters - applied after a query is made

model_class = None
model class

orm_filter_parsers = None

dictionary containing parsing data for ORM/SQLAlchemy-based filters over potentially expensive queries

parse_bool(*bool_string*)
Parse a boolean from a string.

parse_filter(*attr*, *op*, *val*)
Attempt to parse filter as a custom/fn filter, then an orm filter, and if neither work - raise an error.

Raises exceptions.RequestParameterInvalidException if no functional or orm filter can be parsed.

parse_filters(*filter_tuple_list*)
Parse string 3-tuples (attr, op, val) into orm or functional filters.

parse_id_list(*id_list_string*, *sep=', '*)
Split *id_list_string* at *sep*.

string_standard_ops(*key*)

class galaxy.managers.base.**ModelManager**(*app*)
Bases: object

Base class for all model/resource managers.

Provides common queries and CRUD operations as a (hopefully) light layer over the ORM.

associate(*associate_with*, *item*, *foreign_key_name=None*)
Generically associate *item* with *associate_with* based on *foreign_key_name*.

by_id(*id*, ***kwargs*)
Gets a model by primary id.

by_ids(*ids*, *filters=None*, ***kwargs*)
Returns an in-order list of models with the matching ids in *ids*.

copy(*item*, ***kwargs*)
Clone or copy an item.

create(*flush=True*, **args*, ***kwargs*)
Generically create a new model.

foreign_key_name = None

list(*filters=None*, *order_by=None*, *limit=None*, *offset=None*, ***kwargs*)
Returns all objects matching the given filters

model_class
alias of `object`

one(***kwargs*)
Sends kwargs to build the query and returns one and only one model.

query(*eagerloads=True*, *filters=None*, *order_by=None*, *limit=None*, *offset=None*, ***kwargs*)
Return a basic query from model_class, filters, order_by, and limit and offset.

Set eagerloads to False to disable them for this query.

query_associated(*associated_model_class*, *item*, *foreign_key_name=None*)
Generically query other items that have been associated with this *item*.

session()

update(*item*, *new_values*, *flush=True*, ***kwargs*)
Given a dictionary of new values, update *item* and return it.

..note: NO validation or deserialization occurs here.

class `galaxy.managers.base.`**ModelSerializer**(*app*)
Bases: `object`

Turns models into JSONable dicts.

Maintains a map of requestable keys and the Callable() serializer functions that should be called for those keys. E.g. { 'x' : lambda item, key: item.x, ... }

Note: if a key to serialize is not listed in the Serializer.serializable_keyset or serializers, it will not be returned.
To serialize call: my_serializer = MySerializer(app) ... keys_to_serialize = ['id', 'name', 'attr1', 'attr2', ...]
item_dict = MySerializer.serialize(my_item, keys_to_serialize)

add_serializers()
Register a map of attribute keys -> serializing functions that will serialize the attribute.

add_view(*view_name*, *key_list*, *include_keys_from=None*)
Add the list of serializable attributes *key_list* to the serializer's view dictionary under the key *view_name*.

If *include_keys_from* is a proper view name, extend *key_list* by the list in that view.

default_serializer(*item*, *key*, ***context*)
Serialize the *item*'s attribute named *key*.

serialize(*item*, *keys*, ***context*)
Serialize the model *item* to a dictionary.

Given model *item* and the list *keys*, create and return a dictionary built from each key in *keys* that also exists in *serializers* and values of calling the keyed/named serializers on item.

serialize_date(*item*, *key*, ***context*)
Serialize a date attribute of *item*.

serialize_id(*item*, *key*, ***context*)
Serialize an id attribute of *item*.

serialize_to_view(*item*, *view=None*, *keys=None*, *default_view=None*, ***context*)
Use a predefined list of keys (the string *view*) and any additional keys listed in *keys*.

The combinations can be: *view* only: return those keys listed in the named view *keys* only: return those keys listed no *view* or *keys*: use the *default_view* if any *view* and *keys*: combine both into one list of keys

skip(*msg='skipped'*)
To be called from inside a serializer to skip it.

Handy for config checks, information hiding, etc.

static url_for(**args*, ***kargs*)
'service' to use for getting urls - use class var to allow overriding when testing

exception galaxy.managers.base.**ModelSerializingError**(*err_msg=None*, *type='info'*, ***extra_error_info*)
Bases: *galaxy.exceptions.InternalServerError*

Thrown when request model values can't be serialized

class galaxy.managers.base.**ModelValidator**(*app*, **args*, ***kwargs*)
Bases: object

An object that inspects a dictionary (generally meant to be a set of new/updated values for the model) and raises an error if a value is not acceptable.

basestring(*key*, *val*)

basestring_list(*key*, *val*)
Must be a list of basestrings.

bool(*key*, *val*)

genome_build(*key*, *val*)
Must be a valid base_string.

Note: no checking against installation's ref list is done as many data sources consider this an open field.

int(*key*, *val*)

int_range(*key*, *val*, *min=None*, *max=None*)
Must be a int between min and max.

nullable_basestring(*key*, *val*)
Must be a basestring or None.

type(*key*, *val*, *types*)
Check *val* against the type (or tuple of types) in *types*.

Raises exceptions.RequestParameterInvalidException if not an instance.

exception galaxy.managers.base.**SkipAttribute**
Bases: exceptions.Exception

Raise this inside a serializer to prevent the returned dictionary from having a the associated key or value for this attribute.

galaxy.managers.base.**get_class**(*class_name*)
Returns the class object that a string denotes. Without this method, we'd have to do eval(<class_name>).

galaxy.managers.base.**get_object**(*trans*, *id*, *class_name*, *check_ownership=False*, *check_accessible=False*, *deleted=None*)
Convenience method to get a model object with the specified checks. This is a generic method for dealing with objects uniformly from the older controller mixin code - however whenever possible the managers for a particular model should be used to load objects.

galaxy.managers.base.**security_check**(*trans*, *item*, *check_ownership=False*, *check_accessible=False*)
Security checks for an item: checks if (a) user owns item or (b) item is accessible to user. This is a generic method for dealing with objects uniformly from the older controller mixin code - however whenever possible the managers for a particular model should be used to perform security checks.

citations Module

class galaxy.managers.citations.**BaseCitation**
Bases: object

equals(*other_citation*)

has_doi()

to_dict(*citation_format*)

class galaxy.managers.citations.**BibtexCitation**(*elem*, *directory*, *citation_manager*)
Bases: *galaxy.managers.citations.BaseCitation*

to_bibtex()

class galaxy.managers.citations.**CitationCollection**
Bases: object

add(*new_citation*)

class galaxy.managers.citations.**CitationsManager**(*app*)
Bases: object

citations_for_tool(*tool*)

citations_for_tool_ids(*tool_ids*)

parse_citation(*citation_elem*, *tool_directory*)

class galaxy.managers.citations.**DoiCache**(*config*)
Bases: object

get_bibtex(*doi*)

class galaxy.managers.citations.**DoiCitation**(*elem*, *directory*, *citation_manager*)
Bases: *galaxy.managers.citations.BaseCitation*

BIBTEX_UNSET = <object object>

doi()

has_doi()

to_bibtex()

galaxy.managers.citations.**parse_citation**(*elem*, *directory*, *citation_manager*)
Parse an abstract citation entry from the specified XML element. The directory parameter should be used to find external files for this citation.

collections Module

class galaxy.managers.collections.**DatasetCollectionManager**(*app*)
Bases: object

Abstraction for interfacing with dataset collections instance - ideally abstarcts out model and plugin details.

ELEMENTS_UNINITIALIZED = <object object>

copy(*trans*, *parent*, *source*, *encoded_source_id*)

create(*trans*, *parent*, *name*, *collection_type*, *element_identifiers=None*, *elements=None*, *implicit_collection_info=None*)

create_dataset_collection(*trans*, *collection_type*, *elements=None*)

delete(*trans*, *instance_type*, *id*)

get_dataset_collection(*trans*, *encoded_id*)

get_dataset_collection_instance(*trans*, *instance_type*, *id*, ***kwds*)

history_dataset_collections(*history*, *query*)

match_collections(*collections_to_match*)
May seem odd to place it here, but planning to grow sophistication and get plugin types involved so it will likely make sense in the future.

set_collection_elements(*dataset_collection*, *dataset_instances*)

update(*trans*, *instance_type*, *id*, *payload*)

collections_util Module

galaxy.managers.collections_util.**api_payload_to_create_params**(*payload*)
Cleanup API payload to pass into dataset_collections.

galaxy.managers.collections_util.**dictify_dataset_collection_instance**(*dataset_collection_instance*, *parent*, *security*, *view='element'*)

galaxy.managers.collections_util.**dictify_element**(*element*)

galaxy.managers.collections_util.**validate_input_element_identifiers**(*element_identifiers*)
Scan through the list of element identifiers supplied by the API consumer and verify the structure is valid.

context Module

Mixins for transaction-like objects.

class galaxy.managers.context.**ProvidesAppContext**
Bases: object

For transaction-like objects to provide Galaxy convience layer for database and event handling.

Mixed in class must provide *app* property.

expunge_all()

get_toolbox()
Returns the application toolbox

install_model

log_action(*user=None*, *action=None*, *context=None*, *params=None*)
Application-level logging of user actions.

log_event(*message*, *tool_id=None*, ***kwargs*)
Application level logging. Still needs fleshing out (log levels and such) Logging events is a config setting - if False, do not log.

model

request_types()

sa_session
Returns a SQLAlchemy session – currently just gets the current session from the threadlocal session context, but this is provided to allow migration toward a more SQLAlchemy 0.4 style of use.

class galaxy.managers.context.**ProvidesHistoryContext**
Bases: object

For transaction-like objects to provide Galaxy convience layer for reasoning about histories.

Mixed in class must provide *user*, *history*, and *app* properties.

db_builds
Returns the builds defined by galaxy and the builds defined by the user (chromInfo in history).

db_dataset_for(*dbkey*)
Returns the db_file dataset associated/needed by *dataset*, or *None*.

class galaxy.managers.context.**ProvidesUserContext**
Bases: object

For transaction-like objects to provide Galaxy convience layer for reasoning about users.

Mixed in class must provide *user*, *api_inherit_admin*, and *app* properties.

anonymous

get_current_user_roles()

user_can_do_run_as()

user_ftp_dir

user_is_admin()

folders Module Manager and Serializer for Library Folders.

class galaxy.managers.folders.**FolderManager**
Bases: object

Interface/service object for interacting with folders.

can_add_item(*trans*, *folder*)
Return true if the user has permissions to add item to the given folder.

check_accessible(*trans*, *folder*)
Check whether the folder is accessible to current user. By default every folder is accessible (contents have their own permissions).

check_manageable(*trans*, *folder*)
Check whether the user can manage the folder.

Returns the original folder

Return type *LibraryFolder*

Raises AuthenticationRequired, InsufficientPermissionsException

create (*trans*, *parent_folder_id*, *new_folder_name*, *new_folder_description='*')

Create a new folder under the given folder.

Parameters

- **parent_folder_id** (int) – decoded id
- **new_folder_name** (*str*) – name of the new folder
- **new_folder_description** (*str*) – description of the folder (optional, defaults to empty string)

Returns the new folder

Return type *LibraryFolder*

Raises InsufficientPermissionsException

cut_and_decode (*trans*, *encoded_folder_id*)

Cuts the folder prefix (the prepended 'F') and returns the decoded id.

Parameters **encoded_folder_id** (*string*) – encoded id of the Folder object

Returns decoded Folder id

Return type *int*

cut_the_prefix (*encoded_folder_id*)

Remove the prefix from the encoded folder id.

Parameters **encoded_folder_id** (*string*) – encoded id of the Folder object with 'F' prepended

Returns encoded Folder id without the 'F' prefix

Return type string

Raises MalformedId

decode_folder_id (*trans*, *encoded_folder_id*)

Decode the folder id given that it has already lost the prefixed 'F'.

Parameters **encoded_folder_id** (*string*) – encoded id of the Folder object

Returns decoded Folder id

Return type *int*

Raises MalformedId

delete (*trans*, *folder*, *undelete=False*)

Mark given folder deleted/undeleted based on the flag.

Parameters

- **folder** (LibraryFolder) – the model object
- **undelete** (*Bool*) – flag whether to delete (when False) or undelete

Returns the folder

Return type *LibraryFolder*

Raises ItemAccessibilityException

get (*trans*, *decoded_folder_id*, *check_manageable=False*, *check_accessible=True*)
Get the folder from the DB.

Parameters

- **decoded_folder_id** (int) – decoded folder id
- **check_manageable** (bool) – flag whether the check that user can manage item
- **check_accessible** (bool) – flag whether to check that user can access item

Returns the requested folder

Return type *LibraryFolder*

Raises InconsistentDatabase, RequestParameterInvalidException, InternalServerError

get_current_roles (*trans*, *folder*)
Find all roles currently connected to relevant permissions on the folder.

Parameters **folder** (LibraryFolder) – the model object

Returns dict of current roles for all available permission types

Return type dictionary

get_folder_dict (*trans*, *folder*)
Return folder data in the form of a dictionary.

Parameters **folder** (LibraryFolder) – folder item

Returns dict with data about the folder

Return type dictionary

secure (*trans*, *folder*, *check_manageable=True*, *check_accessible=True*)
Check if (a) user can manage folder or (b) folder is accessible to user.

Parameters

- **folder** (LibraryFolder) – folder item
- **check_manageable** (bool) – flag whether to check that user can manage item
- **check_accessible** (bool) – flag whether to check that user can access item

Returns the original folder

Return type *LibraryFolder*

hdas Module Manager and Serializer for HDAs.

HistoryDatasetAssociations (HDAs) are datasets contained or created in a history.

class galaxy.managers.hdas.**HDADeserializer** (*app*)
Bases: galaxy.managers.datasets.DatasetAssociationDeserializer, galaxy.managers.taggable.TaggableDeserializerMixin, galaxy.managers.annotatable.Annotata

Interface/service object for validating and deserializing dictionaries into histories.

add_deserializers ()

model_manager_class
alias of *HDAManager*

class galaxy.managers.hdas.**HDAFilterParser**(*app*)

Bases: galaxy.managers.datasets.DatasetAssociationFilterParser, galaxy.managers.taggable.TaggableFilterMixin, galaxy.managers.annotatable.AnnotatableFil

model_class

alias of HistoryDatasetAssociation

class galaxy.managers.hdas.**HDAManager**(*app*)

Bases: galaxy.managers.datasets.DatasetAssociationManager, galaxy.managers.secured.OwnableManagerMixin, galaxy.managers.taggable.TaggableManagerMix galaxy.managers.annotatable.AnnotatableManagerMixin

Interface/service object for interacting with HDAs.

annotation_assoc

alias of HistoryDatasetAssociationAnnotationAssociation

copy(*hda*, *history=None*, ***kwargs*)

Copy and return the given HDA.

copy_ldda(*history*, *ldda*, ***kwargs*)

Copy this HDA as a LDDA and return.

create(*history=None*, *dataset=None*, *flush=True*, ***kwargs*)

Create a new hda optionally passing in it's history and dataset.

..note: to explicitly set hid to *None* you must pass in *hid=None*, otherwise it will be automatically set.

data_conversion_status(*hda*)

Returns a message if an hda is not ready to be used in visualization.

error_if_uploading(*hda*)

Raise error if HDA is still uploading.

foreign_key_name = 'history_dataset_association'

has_been_resubmitted(*hda*)

Return True if the hda's job was resubmitted at any point.

is_accessible(*hda*, *user*, ***kwargs*)

Override to allow owners (those that own the associated history).

is_owner(*hda*, *user*, *current_history=None*, ***kwargs*)

Use history to see if current user owns HDA.

model_class

alias of HistoryDatasetAssociation

purge(*hda*, *current_user=None*, *flush=True*)

Purge this HDA and the dataset underlying it.

tag_assoc

alias of HistoryDatasetAssociationTagAssociation

text_data(*hda*, *preview=True*)

Get data from text file, truncating if necessary.

class galaxy.managers.hdas.**HDASerializer**(*app*)

Bases: galaxy.managers.datasets.DatasetAssociationSerializer, galaxy.managers.taggable.TaggableSerializerMixin, galaxy.managers.annotatable.Annotatabl

add_serializers()

serialize_display_apps (*hda*, *key*, *trans=None*, ***context*)
Return dictionary containing new-style display app urls.

serialize_old_display_applications (*hda*, *key*, *trans=None*, ***context*)
Return dictionary containing old-style display app urls.

serialize_type_id (*hda*, *key*, ***context*)

serialize_urls (*hda*, *key*, ***context*)
Return web controller urls useful for this HDA.

serialize_visualization_links (*hda*, *key*, *trans=None*, ***context*)
Return a list of dictionaries with links to visualization pages for those visualizations that apply to this hda.

histories Module Manager and Serializer for histories.

Histories are containers for datasets or dataset collections created (or copied) by users over the course of an analysis.

class galaxy.managers.histories.**HistoryDeserializer** (*app*)
Bases: galaxy.managers.sharable.SharableModelDeserializer, galaxy.managers.deletable.PurgableDeserializerMixin

Interface/service object for validating and deserializing dictionaries into histories.

add_deserializers ()

model_manager_class
alias of *HistoryManager*

class galaxy.managers.histories.**HistoryFilters** (*app*)
Bases: galaxy.managers.sharable.SharableModelFilters, galaxy.managers.deletable.PurgableFiltersMixin

model_class
alias of History

class galaxy.managers.histories.**HistoryManager** (*app*, **args*, ***kwargs*)
Bases: galaxy.managers.sharable.SharableModelManager, galaxy.managers.deletable.PurgableManagerMixin

annotation_assoc
alias of HistoryAnnotationAssociation

by_user (*user*, *current_history=None*, ***kwargs*)
Get all the histories for a given user (allowing anon users' theirs) ordered by update time.

copy (*history*, *user*, ***kwargs*)
Copy and return the given *history*.

foreign_key_name = 'history'

get_current (*trans*)
Return the current history.

is_owner (*history*, *user*, *current_history=None*, ***kwargs*)
True if the current user is the owner of the given history.

model_class
alias of History

most_recent (*user*, *filters=None*, *current_history=None*, ***kwargs*)
Return the most recently update history for the user.

If user is anonymous, return the current history. If the user is anonymous and the current history is deleted, return None.

purge (*history*, *flush=True*, ***kwargs*)
Purge this history and all HDAs, Collections, and Datasets inside this history.

rating_assoc
alias of `HistoryRatingAssociation`

set_current (*trans*, *history*)
Set the current history.

set_current_by_id (*trans*, *history_id*)
Set the current history by an id.

tag_assoc
alias of `HistoryTagAssociation`

user_share_model
alias of `HistoryUserShareAssociation`

class `galaxy.managers.histories.`**HistorySerializer** (*app*)
Bases: `galaxy.managers.sharable.SharableModelSerializer`, `galaxy.managers.deletable.PurgableSerializerMixin`

Interface/service object for serializing histories into dictionaries.

SINGLE_CHAR_ABBR = 'h'

add_serializers ()

serialize_contents (*history*, **args*, ***context*)

serialize_history_state (*history*, *key*, ***context*)
Returns the history state based on the states of the HDAs it contains.

serialize_state_counts (*history*, *key*, *exclude_deleted=True*, *exclude_hidden=False*, ***context*)
Return a dictionary keyed to possible dataset states and valued with the number of datasets in this history that have those states.

serialize_state_ids (*history*, *key*, ***context*)
Return a dictionary keyed to possible dataset states and valued with lists containing the ids of each HDA in that state.

lddas Module

class `galaxy.managers.lddas.`**LDDAManager** (*app*)
Bases: `object`

A fairly sparse manager for LDDAs.

get (*trans*, *id*, *check_accessible=True*)

libraries Module

Manager and Serializer for libraries.

class `galaxy.managers.libraries.`**LibraryManager** (**args*, ***kwargs*)
Bases: `object`

Interface/service object for interacting with libraries.

check_accessible (*trans*, *library*)
Check whether the library is accessible to current user.

create (*trans*, *name*, *description=''*, *synopsis=''*)
Create a new library.

delete (*trans*, *library*, *undelete=False*)
Mark given library deleted/undeleted based on the flag.

get (*trans*, *decoded_library_id*, *check_accessible=True*)
Get the library from the DB.

Parameters

- **decoded_library_id** (int) – decoded library id
- **check_accessible** (bool) – flag whether to check that user can access item

Returns the requested library

Return type *Library*

get_access_roles (*trans*, *library*)
Load access roles for all library permissions

get_add_roles (*trans*, *library*)
Load add roles for all library permissions

get_current_roles (*trans*, *library*)
Load all permissions currently related to the given library.

Parameters **library** (Library) – the model object

Return type dictionary

Returns dict of current roles for all available permission types

get_library_dict (*trans*, *library*)
Return library data in the form of a dictionary.

Parameters **library** (Library) – library

Returns dict with data about the library

Return type dictionary

get_manage_roles (*trans*, *library*)
Load manage roles for all library permissions

get_modify_roles (*trans*, *library*)
Load modify roles for all library permissions

is_public (*trans*, *library*)
Return true if lib is public.

list (*trans*, *deleted=False*)
Return a list of libraries from the DB.

Parameters **deleted** (*boolean (optional)*) – if True, show only `deleted` libraries, if False show only `non-deleted`

Returns query that will emit all accessible libraries

Return type sqlalchemy query

make_public (*trans*, *library*)
Makes the given library public (removes all access roles)

secure (*trans*, *library*, *check_accessible=True*)
Check if library is accessible to user.

Parameters

- **folder** (Library) – library
- **check_accessible** (bool) – flag whether to check that user can access library

Returns the original folder

Return type *LibraryFolder*

set_permission_roles(*trans*, *library*, *access_roles*, *modify_roles*, *manage_roles*, *add_roles*)
Set permissions on the given library.

update(*trans*, *library*, *name=None*, *description=None*, *synopsis=None*)
Update the given library

roles Module Manager and Serializer for Roles.

class galaxy.managers.roles.**RoleManager**(*app*)
Bases: *galaxy.managers.base.ModelManager*

Business logic for roles.

foreign_key_name = 'role'

get(*trans*, *decoded_role_id*)
Method loads the role from the DB based on the given role id.

Parameters **decoded_role_id** (int) – id of the role to load from the DB

Returns the loaded Role object

Return type Role

Raises InconsistentDatabase, RequestParameterInvalidException, InternalServerError

group_assoc
alias of GroupRoleAssociation

model_class
alias of Role

user_assoc
alias of UserRoleAssociation

tags Module

class galaxy.managers.tags.**CommunityTagManager**(*app*)
Bases: *galaxy.managers.tags.TagManager*

class galaxy.managers.tags.**GalaxyTagManager**(*app*)
Bases: *galaxy.managers.tags.TagManager*

class galaxy.managers.tags.**ItemTagAssocInfo**(*item_class*, *tag_assoc_class*, *item_id_col*)
Bases: object

class galaxy.managers.tags.**TagManager**(*app*)
Bases: object

Manages CRUD operations related to tagging objects.

apply_item_tag(*user*, *item*, *name*, *value=None*)

apply_item_tags(*user*, *item*, *tags_str*)
Apply tags to an item.

delete_item_tags(*user*, *item*)
Delete tags from an item.

get_community_tags(*item=None*, *limit=None*)
Returns community tags for an item.

get_id_col_in_item_tag_assoc_table(*item_class*)
Returns item id column in class' item-tag association table.

get_tag_assoc_class(*item_class*)
Returns tag association class for item class.

get_tag_by_id(*tag_id*)
Get a Tag object from a tag id.

get_tag_by_name(*tag_name*)
Get a Tag object from a tag name (string).

get_tags_str(*tags*)
Build a string from an item's tags.

get_tool_tags()

item_has_tag(*user*, *item*, *tag*)
Returns true if item is has a given tag.

parse_tags(*tag_str*)
Returns a list of raw (tag-name, value) pairs derived from a string; method scrubs tag names and values as well. Return value is a dictionary where tag-names are keys.

remove_item_tag(*user*, *item*, *tag_name*)
Remove a tag from an item.

set_tags_from_list(*user*, *item*, *new_tags_list*)

workflows Module

class galaxy.managers.workflows.**CreatedWorkflow**(*stored_workflow*, *missing_tools*)
Bases: tuple

missing_tools
Alias for field number 1

stored_workflow
Alias for field number 0

class galaxy.managers.workflows.**MissingToolsException**(*workflow*, *errors*)
Bases: object

class galaxy.managers.workflows.**WorkflowContentsManager**
Bases: *galaxy.model.item_attrs.UsesAnnotations*

build_workflow_from_dict(*trans*, *data*, *source=None*, *add_to_menu=False*, *publish=False*)

update_workflow_from_dict(*trans*, *stored_workflow*, *workflow_data*, *from_editor=False*)

workflow_to_dict(*trans*, *stored*, *style='export'*)
Export the workflow contents to a dictionary ready for JSON-ification and to be sent out via API for instance. There are three styles of export allowed 'export', 'instance', and 'editor'. The Galaxy team will do it best to preserve the backward compatibility of the 'export' stye - this is the export method meant to be portable across Galaxy instances and over time. The 'editor' style is subject to rapid and unannounced changes. The 'instance' export option describes the workflow in a context more tied to the current Galaxy instance and includes fields like 'url' and 'url' and actual unencoded step ids instead of 'order_index'.

class galaxy.managers.workflows.**WorkflowsManager**(*app*)

Bases: object

Handle CRUD type operaitons related to workflows. More interesting stuff regarding workflow execution, step sorting, etc... can be found in the galaxy.workflow module.

build_invocations_query(*trans*, *decoded_stored_workflow_id*)

cancel_invocation(*trans*, *decoded_invocation_id*)

check_security(*trans*, *has_workflow*, *check_ownership=True*, *check_accessible=True*)

check accessibility or ownership of workflows, storedworkflows, and workflowinvocations. Throw an exception or returns True if user has needed level of access.

get_invocation(*trans*, *decoded_invocation_id*)

get_invocation_step(*trans*, *decoded_workflow_invocation_step_id*)

update_invocation_step(*trans*, *decoded_workflow_invocation_step_id*, *action*)

objectstore Package

objectstore Package objectstore package, abstraction for storing blobs of data for use in Galaxy, all providers ensure that data can be accessed on the filesystem for running tools

class galaxy.objectstore.**CachingObjectStore**(*path*, *backend*)

Bases: *galaxy.objectstore.ObjectStore*

Object store that uses a directory for caching files, but defers and writes back to another object store.

class galaxy.objectstore.**DiskObjectStore**(*config*, *config_xml=None*, *file_path=None*, *extra_dirs=None*)

Bases: *galaxy.objectstore.ObjectStore*

Standard Galaxy object store, stores objects in files under a specific directory on disk.

```
>>> from galaxy.util.bunch import Bunch
>>> import tempfile
>>> file_path=tempfile.mkdtemp()
>>> obj = Bunch(id=1)
>>> s = DiskObjectStore(Bunch(umask=077, job_working_directory=file_path, new_file_path=file_pat
>>> s.create(obj)
>>> s.exists(obj)
True
>>> assert s.get_filename(obj) == file_path + '/000/dataset_1.dat'
```

create(*obj*, ***kwargs*)

delete(*obj*, *entire_dir=False*, ***kwargs*)

empty(*obj*, ***kwargs*)

exists(*obj*, ***kwargs*)

get_data(*obj*, *start=0*, *count=-1*, ***kwargs*)

get_filename(*obj*, ***kwargs*)

get_object_url(*obj*, ***kwargs*)

get_store_usage_percent()

size(*obj*, ***kwargs*)

update_from_file(*obj*, *file_name=None*, *create=False*, ***kwargs*)
create parameter is not used in this implementation

class galaxy.objectstore.**DistributedObjectStore**(*config*, *config_xml=None*, *fsmon=False*)
Bases: galaxy.objectstore.NestedObjectStore

ObjectStore that defers to a list of backends, for getting objects the first store where the object exists is used, objects are created in a store selected randomly, but with weighting.

create(*obj*, ***kwargs*)
create() is the only method in which obj.object_store_id may be None

shutdown()

class galaxy.objectstore.**HierarchicalObjectStore**(*config*, *config_xml=None*, *fsmon=False*)
Bases: galaxy.objectstore.NestedObjectStore

ObjectStore that defers to a list of backends, for getting objects the first store where the object exists is used, objects are always created in the first store.

create(*obj*, ***kwargs*)
Create will always be called by the primary object_store

exists(*obj*, ***kwargs*)
Exists must check all child object stores

class galaxy.objectstore.**NestedObjectStore**(*config*, *config_xml=None*)
Bases: galaxy.objectstore.ObjectStore

Base for ObjectStores that use other ObjectStores (DistributedObjectStore, HierarchicalObjectStore)

create(*obj*, ***kwargs*)

delete(*obj*, ***kwargs*)

empty(*obj*, ***kwargs*)

exists(*obj*, ***kwargs*)

file_ready(*obj*, ***kwargs*)

get_data(*obj*, ***kwargs*)

get_filename(*obj*, ***kwargs*)

get_object_url(*obj*, ***kwargs*)

shutdown()

size(*obj*, ***kwargs*)

update_from_file(*obj*, ***kwargs*)

class galaxy.objectstore.**ObjectStore**(*config*, *config_xml=None*, ***kwargs*)
Bases: object

ObjectStore abstract interface

create(*obj*, *base_dir=None*, *dir_only=False*, *extra_dir=None*, *extra_dir_at_root=False*, *alt_name=None*, *obj_dir=False*)
Mark the object identified by *obj* as existing in the store, but with no content. This method will create a proper directory structure for the file if the directory does not already exist. See *exists* method for the description of other fields.

delete(*obj*, *entire_dir=False*, *base_dir=None*, *extra_dir=None*, *extra_dir_at_root=False*, *alt_name=None*, *obj_dir=False*)
Deletes the object identified by *obj*. See *exists* method for the description of other fields.

Parameters **entire_dir** (bool) – If True, delete the entire directory pointed to by extra_dir. For safety reasons, this option applies only for and in conjunction with the extra_dir or obj_dir options.

empty(*obj*, *base_dir=None*, *extra_dir=None*, *extra_dir_at_root=False*, *alt_name=None*, *obj_dir=False*)

Test if the object identified by *obj* has content. If the object does not exist raises *ObjectNotFound*. See *exists* method for the description of the fields.

exists(*obj*, *base_dir=None*, *dir_only=False*, *extra_dir=None*, *extra_dir_at_root=False*, *alt_name=None*)

Returns True if the object identified by *obj* exists in this file store, False otherwise.

FIELD DESCRIPTIONS (these apply to all the methods in this class):

Parameters

- **obj** (*object*) – A Galaxy object with an assigned database ID accessible via the .id attribute.
- **base_dir** (*string*) – A key in self.extra_dirs corresponding to the base directory in which this object should be created, or None to specify the default directory.
- **dir_only** (bool) – If True, check only the path where the file identified by *obj* should be located, not the dataset itself. This option applies to *extra_dir* argument as well.
- **extra_dir** (*string*) – Append *extra_dir* to the directory structure where the dataset identified by *obj* should be located. (e.g., 000/extra_dir/obj.id)
- **extra_dir_at_root** (bool) – Applicable only if *extra_dir* is set. If True, the *extra_dir* argument is placed at root of the created directory structure rather than at the end (e.g., extra_dir/000/obj.id vs. 000/extra_dir/obj.id)
- **alt_name** (*string*) – Use this name as the alternative name for the created dataset rather than the default.
- **obj_dir** (bool) – Append a subdirectory named with the object's ID (e.g. 000/obj.id)

file_ready(*obj*, *base_dir=None*, *dir_only=False*, *extra_dir=None*, *extra_dir_at_root=False*, *alt_name=None*, *obj_dir=False*)

A helper method that checks if a file corresponding to a dataset is ready and available to be used. Return True if so, False otherwise.

get_data(*obj*, *start=0*, *count=-1*, *base_dir=None*, *extra_dir=None*, *extra_dir_at_root=False*, *alt_name=None*, *obj_dir=False*)

Fetch *count* bytes of data starting at offset *start* from the object identified uniquely by *obj*. If the object does not exist raises *ObjectNotFound*. See *exists* method for the description of other fields.

Parameters

- **start** (int) – Set the position to start reading the dataset file
- **count** (int) – Read at most *count* bytes from the dataset

get_filename(*obj*, *base_dir=None*, *dir_only=False*, *extra_dir=None*, *extra_dir_at_root=False*, *alt_name=None*, *obj_dir=False*)

Get the expected filename (including the absolute path) which can be used to access the contents of the object uniquely identified by *obj*. See *exists* method for the description of the fields.

get_object_url(*obj*, *extra_dir=None*, *extra_dir_at_root=False*, *alt_name=None*, *obj_dir=False*)

If the store supports direct URL access, return a URL. Otherwise return None. Note: need to be careful to to bypass dataset security with this. See *exists* method for the description of the fields.

get_store_usage_percent()
Return the percentage indicating how full the store is

shutdown()

size(*obj*, *extra_dir=None*, *extra_dir_at_root=False*, *alt_name=None*, *obj_dir=False*)
Return size of the object identified by *obj*. If the object does not exist, return 0. See *exists* method for the description of the fields.

update_from_file(*obj*, *base_dir=None*, *extra_dir=None*, *extra_dir_at_root=False*, *alt_name=None*, *obj_dir=False*, *file_name=None*, *create=False*)
Inform the store that the file associated with the object has been updated. If *file_name* is provided, update from that file instead of the default. If the object does not exist raises *ObjectNotFound*. See *exists* method for the description of other fields.

Parameters

- **file_name** (*string*) – Use file pointed to by *file_name* as the source for updating the dataset identified by *obj*
- **create** (bool) – If True and the default dataset does not exist, create it first.

galaxy.objectstore.**build_object_store_from_config**(*config*, *fsmon=False*, *config_xml=None*)
Depending on the configuration setting, invoke the appropriate object store

galaxy.objectstore.**convert_bytes**(*bytes*)
A helper function used for pretty printing disk usage

galaxy.objectstore.**create_object_in_session**(*obj*)

galaxy.objectstore.**local_extra_dirs**(*func*)
A decorator for non-local plugins to utilize local directories for their extra_dirs (job_working_directory and temp).

s3_multipart_upload Module Split large file into multiple pieces for upload to S3. This parallelizes the task over available cores using multiprocessing. Code mostly taken form CloudBioLinux.

galaxy.objectstore.s3_multipart_upload.**map_wrap**(*f*)

galaxy.objectstore.s3_multipart_upload.**mp_from_ids**(*s3server*, *mp_id*, *mp_keyname*, *mp_bucketname*)
Get the multipart upload from the bucket and multipart IDs.

This allows us to reconstitute a connection to the upload from within multiprocessing functions.

galaxy.objectstore.s3_multipart_upload.**multimap**(**args*, ***kwds*)
Provide multiprocessing imap like function.

The context manager handles setting up the pool, worked around interrupt issues and terminating the pool on completion.

galaxy.objectstore.s3_multipart_upload.**multipart_upload**(*s3server*, *bucket*, *s3_key_name*, *tarball*, *mb_size*)
Upload large files using Amazon's multipart upload functionality.

galaxy.objectstore.s3_multipart_upload.**transfer_part**(**args*, ***kwargs*)
Transfer a part of a multipart upload. Designed to be run in parallel.

openid Package

openid Package OpenID functionality

providers Module Contains OpenID provider functionality

class galaxy.openid.providers.**OpenIDProvider**(*id*, *name*, *op_endpoint_url*, *sreg_required=None*, *sreg_optional=None*, *use_for=None*, *store_user_preference=None*, *never_associate_with_user=None*)

Bases: object

An OpenID Provider object.

classmethod from_elem(*xml_root*)

classmethod from_file(*filename*)

has_post_authentication_actions()

post_authentication(*trans*, *openid_manager*, *info*)

class galaxy.openid.providers.**OpenIDProviders**(*providers=None*)

Bases: object

Collection of OpenID Providers

NO_PROVIDER_ID = 'None'

classmethod from_elem(*xml_root*)

classmethod from_file(*filename*)

get(*name*, *default=None*)

new_provider_from_identifier(*identifier*)

quota Package

quota Package Galaxy Quotas

class galaxy.quota.**NoQuotaAgent**(*model*)

Bases: object

Base quota agent, always returns no quota

default_quota

get_percent(*trans=None*, *user=False*, *history=False*, *usage=False*, *quota=False*)

get_quota(*user*, *nice_size=False*)

get_usage(*trans=None*, *user=False*, *history=False*)

get_user_quotas(*user*)

class galaxy.quota.**QuotaAgent**(*model*)

Bases: galaxy.quota.NoQuotaAgent

Class that handles galaxy quotas

default_registered_quota

default_unregistered_quota

get_percent (*trans=None*, *user=False*, *history=False*, *usage=False*, *quota=False*)
Return the percentage of any storage quota applicable to the user/transaction.

get_quota (*user*, *nice_size=False*)
Calculated like so:

1. Anonymous users get the default quota.
2. Logged in users start with the highest of their associated '=' quotas or the default quota, if there are no associated '=' quotas. If an '=' unlimited (-1 in the database) quota is found during this process, the user has no quota (aka unlimited).
3. Quota is increased or decreased by any corresponding '+' or '-' quotas.

get_user_quotas (*user*)

set_default_quota (*default_type*, *quota*)

set_entity_quota_associations (*quotas=[]*, *users=[]*, *groups=[]*, *delete_existing_assocs=True*)

sample_tracking Package

data_transfer Module

class galaxy.sample_tracking.data_transfer.**DataTransferFactory**
Bases: object

parse ()

type = None

class galaxy.sample_tracking.data_transfer.**FtpDataTransferFactory**
Bases: *galaxy.sample_tracking.data_transfer.DataTransferFactory*

parse (*elem*)

type = 'ftp'

class galaxy.sample_tracking.data_transfer.**HttpDataTransferFactory**
Bases: *galaxy.sample_tracking.data_transfer.DataTransferFactory*

parse (*config_file*, *elem*)

type = 'http'

class galaxy.sample_tracking.data_transfer.**ScpDataTransferFactory**
Bases: *galaxy.sample_tracking.data_transfer.DataTransferFactory*

parse (*config_file*, *elem*)

type = 'scp'

galaxy.sample_tracking.data_transfer.**data_transfer**
alias of *FtpDataTransferFactory*

external_service_types Module

class galaxy.sample_tracking.external_service_types.**ExternalServiceType** (*external_service_type_xml_con root*, *visible=True*)
Bases: object

parse (*root*)

parse_data_transfer_settings(*root*)

parse_run_details(*root*)

parse_run_details_results(*root*)

exception galaxy.sample_tracking.external_service_types.**ExternalServiceTypeNotFoundException**

Bases: exceptions.Exception

class galaxy.sample_tracking.external_service_types.**ExternalServiceTypesCollection**(*config_filenam root_dir, app*)

Bases: object

load_all(*config_filename*)

load_external_service_type(*config_file, visible=True*)

reload(*external_service_type_id*)

Attempt to reload the external_service_type identified by 'external_service_type_id', if successful replace the old external_service_type.

request_types Module RequestType

class galaxy.sample_tracking.request_types.**RequestTypeFactory**(*sample_state_factory, re-name_dataset_options*)

Bases: object

from_elem(*elem, request_form, sample_form, external_service*)

Return RequestType created from an xml string.

new(*name, request_form, sample_form, external_service, description=None, sample_states=None*)

Return new RequestType.

sample Module Sample

class galaxy.sample_tracking.sample.**SampleStateFactory**

Bases: object

from_elem(*request_type, elem*)

Return SampleState created from an xml string.

new(*request_type, name, description=None*)

Return new SampleState.

security Package

security Package Galaxy Security

class galaxy.security.**Action**(*action, description, model*)

Bases: object

class galaxy.security.**GalaxyRBACAgent**(*model, permitted_actions=None*)

Bases: *galaxy.security.RBACAgent*

allow_action(*roles, action, item*)

Method for checking a permission for the current user (based on roles) to perform a specific action on an item, which must be one of: Dataset, Library, LibraryFolder, LibraryDataset, LibraryDatasetDatasetAs-sociation

allow_action_on_libitems(*trans*, *user_roles*, *action*, *items*)
This should be the equivalent of allow_action defined on multiple items. It is meant to specifically replace allow_action for multiple LibraryDatasets, but it could be reproduced or modified for allow_action's permitted classes - Dataset, Library, LibraryFolder, and LDDAs.

associate_action_dataset_role(*action*, *dataset*, *role*)

associate_components(***kwd*)

associate_group_role(*group*, *role*)

associate_user_group(*user*, *group*)

associate_user_role(*user*, *role*)

can_access_dataset(*user_roles*, *dataset*)

can_access_library(*roles*, *library*)

can_access_library_item(*roles*, *item*, *user*)

can_access_request_type(*roles*, *request_type*)

can_add_library_item(*roles*, *item*)

can_manage_dataset(*roles*, *dataset*)

can_manage_library_item(*roles*, *item*)

can_modify_library_item(*roles*, *item*)

check_folder_contents(*user*, *roles*, *folder*, *hidden_folder_ids='*')
This method must always be sent an instance of LibraryFolder(). Recursive execution produces a comma-separated string of folder ids whose folders do NOT meet the criteria for showing. Along with the string, True is returned if the current user has permission to access folder. Otherwise, cycle through all subfolders in folder until one is found that meets this criteria, if it exists. This method does not necessarily scan the entire library as it returns when it finds the first folder that is accessible to user.

copy_dataset_permissions(*src*, *dst*)

copy_library_permissions(*trans*, *source_library_item*, *target_library_item*, *user=None*)

create_private_user_role(*user*)

dataset_access_mapping(*trans*, *user_roles*, *datasets*)
For the given list of datasets, return a mapping of the datasets' ids to whether they can be accessed by the user or not. The datasets input is expected to be a simple list of Dataset objects.

dataset_is_private_to_user(*trans*, *dataset*)
If the LibraryDataset object has exactly one access role and that is the current user's private role then we consider the dataset private.

dataset_is_public(*dataset*)
A dataset is considered public if there are no "access" actions associated with it. Any other actions ('manage permissions', 'edit metadata') are irrelevant. Accessing dataset.actions will cause a query to be emitted.

dataset_is_unrestricted(*trans*, *dataset*)
Different implementation of the method above with signature: def dataset_is_public(self, dataset)

dataset_permission_map_for_access(*trans*, *user_roles*, *libitems*)
For a given list of library items (e.g., Datasets), return a map of the datasets' ids to whether they can have permission to use that action (e.g., "access" or "modify") on the dataset. The libitems input is expected to be a simple list of library items, such as Datasets or LibraryDatasets. NB: This is currently only usable for Datasets; it was intended to be used for any library item.

datasets_are_public(*trans*, *datasets*)
Given a transaction object and a list of Datasets, return a mapping from Dataset ids to whether the Dataset is public or not. All Dataset ids should be returned in the mapping's keys.

derive_roles_from_access(*trans*, *item_id*, *cntrller*, *library=False*, ***kwd*)

folder_is_public(*folder*)

folder_is_unrestricted(*folder*)

get_accessible_libraries(*trans*, *user*)
Return all data libraries that the received user can access

get_accessible_request_types(*trans*, *user*)
Return all RequestTypes that the received user has permission to access.

get_actions_for_items(*trans*, *action*, *permission_items*)

get_all_roles(*trans*, *cntrller*)

get_component_associations(***kwd*)

get_item_actions(*action*, *item*)

get_legitimate_roles(*trans*, *item*, *cntrller*)
Return a sorted list of legitimate roles that can be associated with a permission on item where item is a Library or a Dataset. The cntrller param is the controller from which the request is sent. We cannot use trans.user_is_admin() because the controller is what is important since admin users do not necessarily have permission to do things on items outside of the admin view.

If cntrller is from the admin side (e.g., library_admin):

- if item is public, all roles, including private roles, are legitimate.
- if item is restricted, legitimate roles are derived from the users and groups associated with each role that is associated with the access permission (i.e., DATASET_MANAGE_PERMISSIONS or LIBRARY_MANAGE) on item. Legitimate roles will include private roles.

If cntrller is not from the admin side (e.g., root, library):

- if item is public, all non-private roles, except for the current user's private role, are legitimate.
- if item is restricted, legitimate roles are derived from the users and groups associated with each role that is associated with the access permission on item. Private roles, except for the current user's private role, will be excluded.

get_permissions(*item*)
Return a dictionary containing the actions and associated roles on item where item is one of Library, LibraryFolder, LibraryDatasetDatasetAssociation, LibraryDataset, Dataset. The dictionary looks like: { Action : [Role, Role] }.

get_permitted_libraries(*trans*, *user*, *actions*)
This method is historical (it is not currently used), but may be useful again at some point. It returns a dictionary whose keys are library objects and whose values are a comma-separated string of folder ids. This method works with the show_library_item() method below, and it returns libraries for which the received user has permission to perform the received actions. Here is an example call to this method to return all libraries for which the received user has LIBRARY_ADD permission:

```
libraries = trans.app.security_agent.get_permitted_libraries( trans, user,
    [ trans.app.security_agent.permitted_actions.LIBRARY_ADD ] )
```

get_private_user_role(*user*, *auto_create=False*)

get_roles_for_action(*item*, *action*)
Return a list containing the roles associated with given action on given item where item is one of Library, LibraryFolder, LibraryDatasetDatasetAssociation, LibraryDataset, Dataset.

get_sharing_roles(*user*)

get_showable_folders(*user*, *roles*, *library_item*, *actions_to_check*, *hidden_folder_ids=[]*, *showable_folders=[]*)
This method must be sent an instance of Library(), all the folders of which are scanned to determine if user is allowed to perform any action in actions_to_check. The param hidden_folder_ids, if passed, should contain a list of folder IDs which was generated when the library was previously scanned using the same actions_to_check. A list of showable folders is generated. This method scans the entire library.

get_valid_roles(*trans*, *item*, *query=None*, *page=None*, *page_limit=None*, *is_library_access=False*)
This method retrieves the list of possible roles that user can select in the item permissions form. Admins can select any role so the results are paginated in order to save the bandwidth and to speed things up. Standard users can select their own private role, any of their sharing roles and any public role (not private and not sharing).

guess_derived_permissions_for_datasets(*datasets=[]*)
Returns a dict of { action : [role, role, ...] } for the output dataset based upon provided datasets

has_accessible_folders(*trans*, *folder*, *user*, *roles*, *search_downward=True*)

has_accessible_library_datasets(*trans*, *folder*, *user*, *roles*, *search_downward=True*)

history_get_default_permissions(*history*)

history_set_default_permissions(*history*, *permissions={}*, *dataset=False*, *bypass_manage_permission=False*)

item_permission_map_for_add(*trans*, *user_roles*, *libitems*)

item_permission_map_for_manage(*trans*, *user_roles*, *libitems*)

item_permission_map_for_modify(*trans*, *user_roles*, *libitems*)

library_is_public(*library*, *contents=False*)

library_is_unrestricted(*library*)

make_dataset_public(*dataset*)

make_folder_public(*folder*)

make_library_public(*library*, *contents=False*)

ok_to_display(*user*, *role*)
Method for checking if: - a role is private and is the current user's private role - a role is a sharing role and belongs to the current user

privately_share_dataset(*dataset*, *users=[]*)

sa_session
Returns a SQLAlchemy session

set_all_dataset_permissions(*dataset*, *permissions={}*)
Set new full permissions on a dataset, eliminating all current permissions. Permission looks like: { Action : [Role, Role] }

set_all_library_permissions(*trans*, *library_item*, *permissions={}*)

set_dataset_permission(*dataset*, *permission={}*)
Set a specific permission on a dataset, leaving all other current permissions on the dataset alone. Permission looks like: { Action.action : [Role, Role] }

set_entity_group_associations(*groups=[]*, *users=[]*, *roles=[]*, *delete_existing_assocs=True*)

set_entity_role_associations(*roles=[]*, *users=[]*, *groups=[]*, *delete_existing_assocs=True*)

set_entity_user_associations(*users=[]*, *roles=[]*, *groups=[]*, *delete_existing_assocs=True*)

set_library_item_permission(*library_item*, *permission={}*)
Set a specific permission on a library item, leaving all other current permissions on the item alone. Permission looks like: { Action.action : [Role, Role] }

set_request_type_permissions(*request_type*, *permissions={}*)

show_library_item(*user*, *roles*, *library_item*, *actions_to_check*, *hidden_folder_ids=''*)
This method must be sent an instance of Library() or LibraryFolder(). Recursive execution produces a comma-separated string of folder ids whose folders do NOT meet the criteria for showing. Along with the string, True is returned if the current user has permission to perform any 1 of actions_to_check on library_item. Otherwise, cycle through all sub-folders in library_item until one is found that meets this criteria, if it exists. This method does not necessarily scan the entire library as it returns when it finds the first library_item that allows user to perform any one action in actions_to_check.

sort_by_attr(*seq*, *attr*)
Sort the sequence of objects by object's attribute Arguments: seq - the list or any sequence (including immutable one) of objects to sort. attr - the name of attribute to sort by

user_get_default_permissions(*user*)

user_set_default_permissions(*user*, *permissions={}*, *history=False*, *dataset=False*, *bypass_manage_permission=False*, *default_access_private=False*)

class galaxy.security.**HostAgent**(*model*, *permitted_actions=None*)
Bases: galaxy.security.RBACAgent

A simple security agent which allows access to datasets based on host. This exists so that externals sites such as UCSC can gain access to datasets which have permissions which would normally prevent such access.

allow_action(*addr*, *action*, ***kwd*)

sa_session
Returns a SQLAlchemy session

set_dataset_permissions(*hda*, *user*, *site*)

sites = <galaxy.util.bunch.Bunch object>

class galaxy.security.**RBACAgent**
Class that handles galaxy security

associate_components(***kwd*)

can_access_dataset(*roles*, *dataset*)

can_access_library(*roles*, *library*)

can_add_library_item(*roles*, *item*)

can_manage_dataset(*roles*, *dataset*)

can_manage_library_item(*roles*, *item*)

can_modify_library_item(*roles*, *item*)

components_are_associated(***kwd*)

convert_permitted_action_strings(*permitted_action_strings*)
When getting permitted actions from an untrusted source like a form, ensure that they match our actual permitted actions.

create_private_user_role(*user*)

dataset_is_public(*dataset*)

derive_roles_from_access(*trans*, *item_id*, *cntrller*, *library=False*, ***kwd*)

folder_is_public(*library*)

get_accessible_libraries(*trans*, *user*)

get_accessible_request_types(*trans*, *user*)

get_action(*name*, *default=None*)
Get a permitted action by its dict key or action name

get_actions()
Get all permitted actions as a list of Action objects

get_all_roles(*trans*, *cntrller*)

get_component_associations(***kwd*)

get_item_actions(*action*, *item*)

get_legitimate_roles(*trans*, *item*, *cntrller*)

get_permissions(*library_dataset*)

get_permitted_libraries(*trans*, *user*, *actions*)

get_private_user_role(*user*)

guess_derived_permissions_for_datasets(*datasets=[]*)

history_set_default_permissions(*history*, *permissions=None*, *dataset=False*, *bypass_manage_permission=False*)

library_is_public(*library*)

make_dataset_public(*dataset*)

make_folder_public(*folder*, *count=0*)

make_library_public(*library*)

permitted_actions = <galaxy.util.bunch.Bunch object>

set_all_dataset_permissions(*dataset*, *permissions*)

set_all_library_permissions(*trans*, *dataset*, *permissions*)

set_dataset_permission(*dataset*, *permission*)

set_library_item_permission(*library_item*, *permission*)

user_set_default_permissions(*user*, *permissions={}*, *history=False*, *dataset=False*)

galaxy.security.**get_permitted_actions**(*filter=None*)
Utility method to return a subset of RBACAgent's permitted actions

validate_user_input Module Utilities for validating inputs related to user objects.

The validate_* methods in this file return simple messages that do not contain user inputs - so these methods do not need to be escaped.

galaxy.security.validate_user_input.**transform_publicname**(*trans*, *publicname*, *user=None*)

galaxy.security.validate_user_input.**validate_email**(*trans*, *email*, *user=None*, *check_dup=True*)

Validates the email format, also checks whether the domain is blacklisted in the disposable domains configuration.

galaxy.security.validate_user_input.**validate_password**(*trans*, *password*, *confirm*)

galaxy.security.validate_user_input.**validate_publicname**(*trans*, *publicname*, *user=None*)

tags Package

tags Package Galaxy tagging classes and methods.

tag_handler Module

tool_shed Package

tool_shed Package

common_util Module

encoding_util Module

install_manager Module

tool_shed_registry Module

update_manager Module

Subpackages

migrate Package

check Module

common Module

tool_dependencies Package

common_util Module

fabric_util Module

install_util Module

tools Package

tools Package Classes encapsulating galaxy tools and tool configuration.

class galaxy.tools.**AsyncDataSourceTool**(*config_file*, *tool_source*, *app*, *guid=None*, *repository_id=None*, *allow_code_files=True*)

Bases: galaxy.tools.DataSourceTool

tool_type = 'data_source_async'

class galaxy.tools.**BadValue**(*value*)

Bases: object

class galaxy.tools.**DataDestinationTool**(*config_file*, *tool_source*, *app*, *guid=None*, *repository_id=None*, *allow_code_files=True*)

Bases: galaxy.tools.Tool

tool_type = 'data_destination'

class galaxy.tools.**DataManagerTool**(*config_file*, *root*, *app*, *guid=None*, *data_manager_id=None*, ***kwds*)

Bases: galaxy.tools.OutputParameterJSONTool

allow_user_access(*user*, *attempting_access=True*)

Parameters

- **user** (galaxy.model.User) – model object representing user.
- **attempting_access** (bool) – is the user attempting to do something with the the tool (set false for incidental checks like toolbox listing)

Returns bool – Whether the user is allowed to access the tool.

Data Manager tools are only accessible to admins.

default_tool_action

alias of DataManagerToolAction

exec_after_process(*app*, *inp_data*, *out_data*, *param_dict*, *job=None*, ***kwds*)

get_default_history_by_trans(*trans*, *create=False*)

tool_type = 'manage_data'

class galaxy.tools.**DataSourceTool**(*config_file*, *tool_source*, *app*, *guid=None*, *repository_id=None*, *allow_code_files=True*)

Bases: galaxy.tools.OutputParameterJSONTool

Alternate implementation of Tool for data_source tools – those that allow the user to query and extract data from another web site.

default_tool_action

alias of DataSourceToolAction

exec_before_job(*app*, *inp_data*, *out_data*, *param_dict=None*)

parse_inputs(*tool_source*)

tool_type = 'data_source'

class galaxy.tools.**DefaultToolState**
Bases: object

Keeps track of the state of a users interaction with a tool between requests. The default tool state keeps track of the current page (for multipage "wizard" tools) and the values of all

copy()
WARNING! Makes a shallow copy, *SHOULD* rework to have it make a deep copy.

decode(*value*, *tool*, *app*, *secure=True*)
Restore the state from a string

encode(*tool*, *app*, *secure=True*)
Convert the data to a string

class galaxy.tools.**ExportHistoryTool**(*config_file*, *tool_source*, *app*, *guid=None*, *repository_id=None*, *allow_code_files=True*)
Bases: galaxy.tools.Tool

tool_type = 'export_history'

class galaxy.tools.**GenomeIndexTool**(*config_file*, *tool_source*, *app*, *guid=None*, *repository_id=None*, *allow_code_files=True*)
Bases: galaxy.tools.Tool

tool_type = 'index_genome'

class galaxy.tools.**ImportHistoryTool**(*config_file*, *tool_source*, *app*, *guid=None*, *repository_id=None*, *allow_code_files=True*)
Bases: galaxy.tools.Tool

tool_type = 'import_history'

exception galaxy.tools.**InterruptedUpload**
Bases: exceptions.Exception

class galaxy.tools.**OutputParameterJSONTool**(*config_file*, *tool_source*, *app*, *guid=None*, *repository_id=None*, *allow_code_files=True*)
Bases: galaxy.tools.Tool

Alternate implementation of Tool that provides parameters and other values JSONified within the contents of an output dataset

exec_before_job(*app*, *inp_data*, *out_data*, *param_dict=None*)

tool_type = 'output_parameter_json'

class galaxy.tools.**SetMetadataTool**(*config_file*, *tool_source*, *app*, *guid=None*, *repository_id=None*, *allow_code_files=True*)
Bases: galaxy.tools.Tool

Tool implementation for special tool that sets metadata on an existing dataset.

exec_after_process(*app*, *inp_data*, *out_data*, *param_dict*, *job=None*)

job_failed(*job_wrapper*, *message*, *exception=False*)

requires_setting_metadata = False

tool_type = 'set_metadata'

class galaxy.tools.**SetParamAction**(*name*, *output_name*)
Set parameter action.

static parse (*elt*)
Parse action from element.

class galaxy.tools.**Tool** (*config_file*, *tool_source*, *app*, *guid=None*, *repository_id=None*, *allow_code_files=True*)
Bases: object, galaxy.model.item_attrs.Dictifiable

Represents a computational tool that can be executed through Galaxy.

allow_user_access (*user*, *attempting_access=True*)

Returns bool – Whether the user is allowed to access the tool.

build_dependency_shell_commands ()
Return a list of commands to be run to populate the current environment to include this tools requirements.

build_redirect_url_params (*param_dict*)
Substitute parameter values into self.redirect_url_params

call_hook (*hook_name*, **args*, ***kwargs*)
Call the custom code hook function identified by 'hook_name' if any, and return the results

check_and_update_param_values (*values*, *trans*, *update_values=True*, *allow_workflow_parameters=False*)
Check that all parameters have values, and fill in with default values where necessary. This could be called after loading values from a database in case new parameters have been added.

check_and_update_param_values_helper (*inputs*, *values*, *trans*, *messages*, *context=None*, *prefix=''*, *update_values=True*, *allow_workflow_parameters=False*)
Recursive helper for *check_and_update_param_values_helper*

check_workflow_compatible (*tool_source*)
Determine if a tool can be used in workflows. External tools and the upload tool are currently not supported by workflows.

collect_child_datasets (*output*, *job_working_directory*)
Look for child dataset files, create HDA and attach to parent.

collect_dynamic_collections (*output*, ***kwds*)
Find files corresponding to dynamically structured collections.

collect_primary_datasets (*output*, *job_working_directory*, *input_ext*)
Find any additional datasets generated by a tool and attach (for cases where number of outputs is not known in advance).

default_template = 'tool_form.mako'

default_tool_action
alias of DefaultToolAction

dict_collection_visible_keys = ('id', 'name', 'version', 'description')

exec_after_process (*app*, *inp_data*, *out_data*, *param_dict*, *job=None*)

exec_before_job (*app*, *inp_data*, *out_data*, *param_dict={}*)

execute (*trans*, *incoming={}*, *set_output_hid=True*, *history=None*, ***kwargs*)
Execute the tool using parameter values in *incoming*. This just dispatches to the *ToolAction* instance specified by *self.tool_action*. In general this will create a *Job* that when run will build the tool's outputs, e.g. *DefaultToolAction*.

fill_in_new_state(*trans*, *inputs*, *state*, *context=None*, *history=None*)
Fill in a tool state dictionary with default values for all parameters in the dictionary *inputs*. Grouping elements are filled in recursively.

find_fieldstorage(*x*)

find_output_def(*name*)

get_default_history_by_trans(*trans*, *create=False*)

classmethod get_externally_referenced_paths(*path*)
Return relative paths to externally referenced files by the tool described by file at *path*. External components should not assume things about the structure of tool xml files (this is the tool's responsibility).

get_hook(*name*)
Returns an object from the code file referenced by *code_namespace* (this will normally be a callable object)

get_job_destination(*job_params=None*)

Returns galaxy.jobs.JobDestination – The destination definition and runner parameters.

get_job_handler(*job_params=None*)
Get a suitable job handler for this *Tool* given the provided *job_params*. If multiple handlers are valid for combination of *Tool* and *job_params* (e.g. the defined handler is a handler tag), one will be selected at random.

Parameters **job_params** (*dict or None*) – Any params specific to this job (e.g. the job source)

Returns str – The id of a job handler for a job run of this *Tool*

get_panel_section()

get_param(*key*)
Returns the parameter named *key* or None if there is no such parameter.

get_param_html_map(*trans*, *page=0*, *other_values={}*)
Return a dictionary containing the HTML representation of each parameter. This is used for rendering display elements. It is currently not compatible with grouping constructs.

NOTE: This should be considered deprecated, it is only used for tools with *display* elements. These should be eliminated.

get_static_param_values(*trans*)
Returns a map of parameter names and values if the tool does not require any user input. Will raise an exception if any parameter does require input.

handle_input(*trans*, *incoming*, *history=None*, *old_errors=None*, *process_state='update'*, *source='html'*)
Process incoming parameters for this tool from the dict *incoming*, update the tool state (or create if none existed), and either return to the form or execute the tool (only if 'execute' was clicked and there were no errors).

process_state can be either 'update' (to incrementally build up the state over several calls - one repeat per handle for instance) or 'populate' force a complete build of the state and submission all at once (like from API). May want an incremental version of the API also at some point, that is why this is not just called for_api.

handle_interrupted(*trans*, *inputs*)
Upon handling inputs, if it appears that we have received an incomplete form, do some cleanup or anything else deemed necessary. Currently this is only likely during file uploads, but this method could be generalized and a method standardized for handling other tools.

handle_job_failure_exception(*e*)
Called by job.fail when an exception is generated to allow generation of a better error message (returning None yields the default behavior)

handle_single_execution(*trans*, *rerun_remap_job_id*, *params*, *history*, *mapping_over_collection*)
Return a pair with whether execution is successful as well as either resulting output data or an error message indicating the problem.

handle_unvalidated_param_values(*input_values*, *app*)
Find any instances of *UnvalidatedValue* within input_values and validate them (by calling *ToolParameter.from_html* and *ToolParameter.validate*).

handle_unvalidated_param_values_helper(*inputs*, *input_values*, *app*, *context=None*, *prefix=''*)
Recursive helper for *handle_unvalidated_param_values*

help

help_by_page

installed_tool_dependencies

job_failed(*job_wrapper*, *message*, *exception=False*)
Called when a job has failed

new_state(*trans*, *all_pages=False*, *history=None*)
Create a new *DefaultToolState* for this tool. It will be initialized with default values for inputs.

Only inputs on the first page will be initialized unless *all_pages* is True, in which case all inputs regardless of page are initialized.

params_from_strings(*params*, *app*, *ignore_errors=False*)

params_to_strings(*params*, *app*)

params_with_missing_data_table_entry
Return all parameters that are dynamically generated select lists whose options require an entry not currently in the tool_data_table_conf.xml file.

params_with_missing_index_file
Return all parameters that are dynamically generated select lists whose options refer to a missing .loc file.

parse(*tool_source*, *guid=None*)
Read tool configuration from the element *root* and fill in *self*.

parse_help(*tool_source*)
Parse the help text for the tool. Formatted in reStructuredText, but stored as Mako to allow for dynamic image paths. This implementation supports multiple pages.

parse_input_elem(*page_source*, *enctypes*, *context=None*)
Parse a parent element whose children are inputs – these could be groups (repeat, conditional) or param elements. Groups will be parsed recursively.

parse_input_page(*page_source*, *enctypes*)
Parse a page of inputs. This basically just calls 'parse_input_elem', but it also deals with possible 'display' elements which are supported only at the top/page level (not in groups).

parse_inputs(*tool_source*)
Parse the "<inputs>" element and create appropriate 'ToolParameter's. This implementation supports multiple pages and grouping constructs.

parse_outputs(*tool_source*)
Parse <outputs> elements and fill in self.outputs (keyed by name)

parse_param_elem(*input_source*, *enctypes*, *context*)
Parse a single "<param>" element and return a ToolParameter instance. Also, if the parameter has a 'required_enctype' add it to the set enctypes.

parse_redirect_url(*data*, *param_dict*)
Parse the REDIRECT_URL tool param. Tools that send data to an external application via a redirect must include the following 3 tool params:

1. REDIRECT_URL - the url to which the data is being sent
2. DATA_URL - the url to which the receiving application will send an http post to retrieve the Galaxy data
3. GALAXY_URL - the url to which the external application may post data as a response

parse_stdio(*tool_source*)
Parse <stdio> element(s) and fill in self.return_codes, self.stderr_rules, and self.stdout_rules. Return codes have a range and an error type (fault or warning). Stderr and stdout rules have a regular expression and an error level (fault or warning).

populate_state(*trans*, *inputs*, *state*, *incoming*, *history=None*, *source='html'*, *prefix=''*, *context=None*)

populate_tool_shed_info()

produces_collections

requires_setting_metadata = **True**

sa_session
Returns a SQLAlchemy session

tests

to_dict(*trans*, *link_details=False*, *io_details=False*)
Returns dict of tool.

to_json(*trans*, *kwd={}*, *is_workflow=False*)
Recursively creates a tool dictionary containing repeats, dynamic options and updated states.

tool_shed_repository

tool_type = **'default'**

tool_version
Return a ToolVersion if one exists for our id

tool_versions

update_state(*trans*, *inputs*, *state*, *incoming*, *source='html'*, *prefix=''*, *context=None*, *update_only=False*, *old_errors={}*, *item_callback=None*)
Update the tool state in *state* using the user input in *incoming*. This is designed to be called recursively: *inputs* contains the set of inputs being processed, and *prefix* specifies a prefix to add to the name of each input to extract its value from *incoming*.

If *update_only* is True, values that are not in *incoming* will not be modified. In this case *old_errors* can be provided, and any errors for parameters which were *not* updated will be preserved.

visit_inputs(*value*, *callback*)
Call the function *callback* on each parameter of this tool. Visits grouping parameters recursively and constructs unique prefixes for each nested set of The callback method is then called as:

callback(level_prefix, parameter, parameter_value)

class galaxy.tools.**ToolBox**(*config_filenames*, *tool_root_dir*, *app*)
Bases: galaxy.tools.toolbox.base.AbstractToolBox

A derivative of AbstractToolBox with knowledge about Tool internals - how to construct them, action types, dependency management, etc....

create_tool(*config_file*, *repository_id=None*, *guid=None*, ***kwds*)

handle_datatypes_changed()
Refresh upload tools when new datatypes are added.

tools_by_id

exception galaxy.tools.**ToolNotFoundException**
Bases: exceptions.Exception

class galaxy.tools.**ToolOutput**(*name*, *format=None*, *format_source=None*, *metadata_source=None*, *parent=None*, *label=None*, *filters=None*, *actions=None*, *hidden=False*, *implicit=False*)
Bases: *galaxy.tools.ToolOutputBase*

Represents an output datasets produced by a tool. For backward compatibility this behaves as if it were the tuple:

```
(format, metadata_source, parent)
```

dict_collection_visible_keys = ('name', 'format', 'label', 'hidden')

class galaxy.tools.**ToolOutputBase**(*name*, *label=None*, *filters=None*, *hidden=False*)
Bases: object, *galaxy.model.item_attrs.Dictifiable*

class galaxy.tools.**ToolOutputCollection**(*name*, *structure*, *label=None*, *filters=None*, *hidden=False*, *default_format='data'*, *default_format_source=None*, *default_metadata_source=None*, *inherit_format=False*, *inherit_metadata=False*)
Bases: *galaxy.tools.ToolOutputBase*

Represents a HistoryDatasetCollectionAssociation of output datasets produced by a tool. <outputs>
<dataset_collection type="list" label="${tool.name} on ${on_string} fasta">
<discover_datasets pattern="__name__" ext="fasta" visible="True" directory="outputFiles" />

</dataset_collection> <dataset_collection type="paired" label="${tool.name} on ${on_string} paired reads">

<data name="forward" format="fastqsanger" /> <data name="reverse" format="fastqsanger"/>

</dataset_collection>
<outputs>

dataset_collectors

dynamic_structure

known_outputs(*inputs*, *type_registry*)

class galaxy.tools.**ToolOutputCollectionPart**(*output_collection_def*, *element_identifier*, *output_def*)
Bases: object

effective_output_name

static is_named_collection_part_name(*name*)

static split_output_name(*name*)

class galaxy.tools.**ToolOutputCollectionStructure**(*collection_type*, *structured_like*, *dataset_collectors*)

Bases: object

class galaxy.tools.**TracksterConfig**(*actions*)

Trackster configuration encapsulation.

static parse(*root*)

galaxy.tools.**check_param_from_incoming**(*trans*, *state*, *input*, *incoming*, *key*, *context*, *source*)

Unlike "update" state, this preserves default if no incoming value found. This lets API user specify just a subset of params and allow defaults to be used when available.

galaxy.tools.**get_incoming_value**(*incoming*, *key*, *default*)

Fetch value from incoming dict directly or check special nginx upload created variants of this key.

galaxy.tools.**json_fix**(*val*)

galaxy.tools.**tool_class**

alias of *DataDestinationTool*

exception_handling Module Exceptions and handlers for tools.

FIXME: These are used by tool scripts, not the framework, and should not live in this package.

exception galaxy.tools.exception_handling.**UCSCLimitException**

Bases: exceptions.Exception

class galaxy.tools.exception_handling.**UCSCOutWrapper**(*other*)

Bases: object

File-like object that throws an exception if it encounters the UCSC limit error lines

next()

readline()

test Module

class galaxy.tools.test.**ParamContext**(*name*, *index=None*, *parent_context=None*)

Bases: object

extract_value(*raw_inputs*)

for_state()

param_names()

class galaxy.tools.test.**RootParamContext**

Bases: object

for_state()

get_index()

param_names()

class galaxy.tools.test.**ToolTestBuilder**(*tool*, *test_dict*, *i*, *default_interactor*)

Bases: object

Encapsulates information about a tool test, and allows creation of a dynamic TestCase class (the unittest framework is very class oriented, doing dynamic tests in this way allows better integration)

test_data()
Iterator over metadata representing the required files for upload.

galaxy.tools.test.**nottest**(*x*)

galaxy.tools.test.**parse_tests**(*tool*, *tests_source*)
Build ToolTestBuilder objects for each "<test>" elements and return default interactor (if any).

galaxy.tools.test.**require_file**(*name*, *value*, *extra*, *required_files*)

galaxy.tools.test.**test_data_iter**(*required_files*)

Subpackages

actions Package

actions Package

class galaxy.tools.actions.**DefaultToolAction**
Bases: object

Default tool action is to run an external command

collect_input_dataset_collections(*tool*, *param_values*)

collect_input_datasets(*tool*, *param_values*, *trans*)
Collect any dataset inputs from incoming. Returns a mapping from parameter name to Dataset instance for each tool parameter that is of the DataToolParameter type.

execute(*tool*, *trans*, *incoming={}*, *return_job=False*, *set_output_hid=True*, *set_output_history=True*, *history=None*, *job_params=None*, *rerun_remap_job_id=None*, *mapping_over_collection=False*)
Executes a tool, creating job and tool outputs, associating them, and submitting the job to the job queue. If history is not specified, use trans.history as destination for tool's output datasets.

get_output_name(*output*, *dataset*, *tool*, *on_text*, *trans*, *incoming*, *history*, *params*, *job_params*)

class galaxy.tools.actions.**ObjectStorePopulator**(*app*)
Bases: object

Small helper for interacting with the object store and making sure all datasets from a job end up with the same object_store_id.

set_object_store_id(*data*)

class galaxy.tools.actions.**ToolAction**
Bases: object

The actions to be taken when a tool is run (after parameters have been converted and validated).

execute(*tool*, *trans*, *incoming={}*, *set_output_hid=True*)

galaxy.tools.actions.**determine_output_format**(*output*, *parameter_context*, *input_datasets*, *random_input_ext*)
Determines the output format for a dataset based on an abstract description of the output (galaxy.tools.ToolOutput), the parameter wrappers, a map of the input datasets (name => HDA), and the last input extensions in the tool form.

TODO: Don't deal with XML here - move this logic into ToolOutput. TODO: Make the input extension used deterministic instead of random.

galaxy.tools.actions.**filter_output**(*output*, *incoming*)

galaxy.tools.actions.**on_text_for_names**(*input_names*)

history_imp_exp Module

class galaxy.tools.actions.history_imp_exp.**ExportHistoryToolAction**

Bases: *galaxy.tools.actions.ToolAction*

Tool action used for exporting a history to an archive.

execute(*tool*, *trans*, *incoming={}*, *set_output_hid=False*, *overwrite=True*, *history=None*, ***kwargs*)

class galaxy.tools.actions.history_imp_exp.**ImportHistoryToolAction**

Bases: *galaxy.tools.actions.ToolAction*

Tool action used for importing a history to an archive.

execute(*tool*, *trans*, *incoming={}*, *set_output_hid=False*, *overwrite=True*, *history=None*, ***kwargs*)

index_genome Module

metadata Module

class galaxy.tools.actions.metadata.**SetMetadataToolAction**

Bases: galaxy.tools.actions.__init__.ToolAction

Tool action used for setting external metadata on an existing dataset

execute(*tool*, *trans*, *incoming={}*, *set_output_hid=False*, *overwrite=True*, *history=None*, *job_params=None*, ***kwargs*)

Execute using a web transaction.

execute_via_app(*tool*, *app*, *session_id*, *history_id*, *user=None*, *incoming={}*, *set_output_hid=False*, *overwrite=True*, *history=None*, *job_params=None*)

Execute using application.

upload Module

class galaxy.tools.actions.upload.**UploadToolAction**

Bases: galaxy.tools.actions.__init__.ToolAction

execute(*tool*, *trans*, *incoming={}*, *set_output_hid=True*, *history=None*, ***kwargs*)

upload_common Module

galaxy.tools.actions.upload_common.**active_folders**(*trans*, *folder*)

galaxy.tools.actions.upload_common.**cleanup_unused_precreated_datasets**(*precreated_datasets*)

galaxy.tools.actions.upload_common.**create_job**(*trans*, *params*, *tool*, *json_file_path*, *data_list*, *folder=None*, *history=None*)

Create the upload job.

galaxy.tools.actions.upload_common.**create_paramfile**(*trans*, *uploaded_datasets*)

Create the upload tool's JSON "param" file.

galaxy.tools.actions.upload_common.**get_precreated_dataset**(*precreated_datasets*, *name*)

Return a dataset matching a name from the list of precreated (via async upload) datasets. If there's more than one upload with the exact same name, we need to pop one (the first) so it isn't chosen next time.

galaxy.tools.actions.upload_common.**get_precreated_datasets**(*trans*, *params*, *data_obj*, *controller='root'*)

Get any precreated datasets (when using asynchronous uploads).

galaxy.tools.actions.upload_common.**get_uploaded_datasets**(*trans*, *cntrller*, *params*, *precreated_datasets*, *dataset_upload_inputs*, *library_bunch=None*, *history=None*)

galaxy.tools.actions.upload_common.**handle_library_params**(*trans*, *params*, *folder_id*, *replace_dataset=None*)

galaxy.tools.actions.upload_common.**new_upload**(*trans*, *cntrller*, *uploaded_dataset*, *library_bunch=None*, *history=None*, *state=None*)

galaxy.tools.actions.upload_common.**persist_uploads**(*params*)

Turn any uploads in the submitted form to persisted files.

data Package

data Package Manage tool data tables, which store (at the application level) data that is used by tools, for example in the generation of dynamic options. Tables are loaded and stored by names which tools use to refer to them. This allows users to configure data tables for a local Galaxy instance without needing to modify the tool configurations.

class galaxy.tools.data.**TabularToolDataField**(*data*)

Bases: *galaxy.model.item_attrs.Dictifiable*, object

clean_base_dir(*path*)

dict_collection_visible_keys = []

get_base_dir()

get_base_path()

get_files()

get_filesize_map(*rm_base_dir=False*)

get_fingerprint()

to_dict()

class galaxy.tools.data.**TabularToolDataTable**(*config_element*, *tool_data_path*, *from_shed_config=False*, *filename=None*)

Bases: *galaxy.tools.data.ToolDataTable*, *galaxy.model.item_attrs.Dictifiable*

Data stored in a tabular / separated value format on disk, allows multiple files to be merged but all must have the same column definitions:

```
<table type="tabular" name="test">
    <column name='...' index = '...' />
    <file path="..." />
    <file path="..." />
</table>
```

configure_and_load(*config_element*, *tool_data_path*, *from_shed_config=False*, *url_timeout=10*)

Configure and load table from an XML element.

dict_collection_visible_keys = ['name']

extend_data_with(*filename*, *errors=None*)

filter_file_fields(*loc_file*, *values*)
Reads separated lines from file and print back only the lines that pass a filter.

get_column_name_list()

get_entries(*query_attr*, *query_val*, *return_attr*, *default=None*, *limit=None*)
Returns table entry associated with a col/val pair.

get_entry(*query_attr*, *query_val*, *return_attr*, *default=None*)
Returns table entry associated with a col/val pair.

get_field(*value*)

get_fields()

get_filename_for_source(*source*, *default=None*)

get_named_fields_list()

get_version_fields()

handle_found_index_file(*filename*)

merge_tool_data_table(*other_table*, *allow_duplicates=True*, *persist=False*, *persist_on_error=False*, *entry_source=None*, ***kwd*)

parse_column_spec(*config_element*)
Parse column definitions, which can either be a set of 'column' elements with a name and index (as in dynamic options config), or a shorthand comma separated list of names in order as the text of a 'column_names' element.

A column named 'value' is required.

parse_file_fields(*reader*, *errors=None*, *here='__HERE__'*)
Parse separated lines from file and return a list of tuples.

TODO: Allow named access to fields using the column names.

to_dict(*view='collection'*)

type_key = 'tabular'

xml_string

class galaxy.tools.data.**ToolDataTable**(*config_element*, *tool_data_path*, *from_shed_config=False*, *filename=None*)
Bases: object

add_entries(*entries*, *allow_duplicates=True*, *persist=False*, *persist_on_error=False*, *entry_source=None*, ***kwd*)

add_entry(*entry*, *allow_duplicates=True*, *persist=False*, *persist_on_error=False*, *entry_source=None*, ***kwd*)

classmethod from_elem(*table_elem*, *tool_data_path*, *from_shed_config*, *filename*)

get_empty_field_by_name(*name*)

is_current_version(*other_version*)

merge_tool_data_table(*other_table*, *allow_duplicates=True*, *persist=False*, *persist_on_error=False*, *entry_source=None*, ***kwd*)

reload_from_files()

remove_entry(*values*)

class galaxy.tools.data.**ToolDataTableManager**(*tool_data_path*, *config_filename=None*)

Bases: object

Manages a collection of tool data tables

add_new_entries_from_config_file(*config_filename*, *tool_data_path*, *shed_tool_data_table_config*, *persist=False*)

This method is called when a tool shed repository that includes a tool_data_table_conf.xml.sample file is being installed into a local galaxy instance. We have 2 cases to handle, files whose root tag is <tables>, for example:

```
<tables>
    <!-- Location of Tmap files -->
    <table name="tmap_indexes" comment_char="#">
        <columns>value, dbkey, name, path</columns>
        <file path="tool-data/tmap_index.loc" />
    </table>
</tables>
```

and files whose root tag is <table>, for example:

```
<!-- Location of Tmap files -->
<table name="tmap_indexes" comment_char="#">
    <columns>value, dbkey, name, path</columns>
    <file path="tool-data/tmap_index.loc" />
</table>
```

get(*name*, *default=None*)

get_tables()

load_from_config_file(*config_filename*, *tool_data_path*, *from_shed_config=False*)

This method is called under 3 conditions:

1. When the ToolDataTableManager is initialized (see __init__ above).
2. Just after the ToolDataTableManager is initialized and the additional entries defined by shed_tool_data_table_conf.xml are being loaded into the ToolDataTableManager.data_tables.
3. When a tool shed repository that includes a tool_data_table_conf.xml.sample file is being installed into a local Galaxy instance. In this case, we have 2 entry types to handle, files whose root tag is <tables>, for example:

reload_tables(*table_names=None*)

set(*name*, *value*)

to_xml_file(*shed_tool_data_table_config*, *new_elems=None*, *remove_elems=None*)

Write the current in-memory version of the shed_tool_data_table_conf.xml file to disk. remove_elems are removed before new_elems are added.

galaxy.tools.data.**cls**

alias of *TabularToolDataTable*

galaxy.tools.data.**expand_here_template**(*content*, *here=None*)

deps Package

deps Package Dependency management for tools.

class galaxy.tools.deps.**DependencyManager**(*default_base_path*, *conf_file=None*)
Bases: object

A DependencyManager attempts to resolve named and versioned dependencies by searching for them under a list of directories. Directories should be of the form:
$BASE/name/version/...
and should each contain a file 'env.sh' which can be sourced to make the dependency available in the current shell environment.

dependency_shell_commands(*requirements*, ***kwds*)

find_dep(*name*, *version=None*, *type='package'*, ***kwds*)

uses_tool_shed_dependencies()

class galaxy.tools.deps.**NullDependencyManager**
Bases: object

dependency_shell_commands(*requirements*, ***kwds*)

find_dep(*name*, *version=None*, *type='package'*, ***kwds*)

uses_tool_shed_dependencies()

galaxy.tools.deps.**build_dependency_manager**(*config*)

tests Module

genome_index Package

genome_index Package

index_genome Module

imp_exp Package

imp_exp Package

class galaxy.tools.imp_exp.**JobExportHistoryArchiveWrapper**(*job_id*)
Bases: object, *galaxy.model.item_attrs.UsesAnnotations*

Class provides support for performing jobs that export a history to an archive.

cleanup_after_job(*db_session*)
Remove temporary directory and attribute files generated during setup for this job.

get_history_datasets(*trans*, *history*)
Returns history's datasets.

setup_job(*trans*, *jeha*, *include_hidden=False*, *include_deleted=False*)
Perform setup for job to export a history into an archive. Method generates attribute files for export, sets the corresponding attributes in the jeha object, and returns a command line for running the job. The command line includes the command, inputs, and options; it does not include the output file because it must be set at runtime.

class galaxy.tools.imp_exp.**JobImportHistoryArchiveWrapper**(*app*, *job_id*)
Bases: object, *galaxy.model.item_attrs.UsesAnnotations*

Class provides support for performing jobs that import a history from an archive.

cleanup_after_job()
Set history, datasets, and jobs' attributes and clean up archive directory.

galaxy.tools.imp_exp.**load_history_imp_exp_tools**(*toolbox*)
Adds tools for importing/exporting histories to archives.

export_history Module Export a history to an archive file using attribute files.

usage: %prog history_attrs dataset_attrs job_attrs out_file -G, –gzip: gzip archive file

galaxy.tools.imp_exp.export_history.**create_archive**(*history_attrs_file*, *datasets_attrs_file*, *jobs_attrs_file*, *out_file*, *gzip=False*)
Create archive from the given attribute/metadata files and save it to out_file.

galaxy.tools.imp_exp.export_history.**get_dataset_filename**(*name*, *ext*)
Builds a filename for a dataset using its name an extension.

galaxy.tools.imp_exp.export_history.**main**()

unpack_tar_gz_archive Module Unpack a tar or tar.gz archive into a directory.

usage: %prog archive_source dest_dir –[url|file] source type, either a URL or a file.

galaxy.tools.imp_exp.unpack_tar_gz_archive.**unpack_archive**(*archive_file*, *dest_dir*)
Unpack a tar and/or gzipped archive into a destination directory.

galaxy.tools.imp_exp.unpack_tar_gz_archive.**url_to_file**(*url*, *dest_file*)
Transfer a file from a remote URL to a temporary file.

parameters Package

parameters Package Classes encapsulating Galaxy tool parameters.

galaxy.tools.parameters.**check_param**(*trans*, *param*, *incoming_value*, *param_values*, *source='html'*)
Check the value of a single parameter *param*. The value in *incoming_value* is converted from its HTML encoding and validated. The *param_values* argument contains the processed values of previous parameters (this may actually be an ExpressionContext when dealing with grouping scenarios).

galaxy.tools.parameters.**params_from_strings**(*params*, *param_values*, *app*, *ignore_errors=False*)
Convert a dictionary of strings as produced by *params_to_strings* back into parameter values (decode the json representation and then allow each parameter to convert the basic types into the parameters preferred form).

galaxy.tools.parameters.**params_to_incoming**(*incoming*, *inputs*, *input_values*, *app*, *name_prefix=''*, *to_html=True*)
Given a tool's parameter definition (*inputs*) and a specific set of parameter *input_values* objects, populate *incoming* with the html values.

Useful for e.g. the rerun function.

galaxy.tools.parameters.**params_to_strings**(*params*, *param_values*, *app*)
Convert a dictionary of parameter values to a dictionary of strings suitable for persisting. The *value_to_basic* method of each parameter is called to convert its value to basic types, the result of which is then json encoded (this allowing complex nested parameters and such).

galaxy.tools.parameters.**visit_input_values**(*inputs*, *input_values*, *callback*, *name_prefix=''*, *label_prefix=''*)

Given a tools parameter definition (*inputs*) and a specific set of parameter *values*, call *callback* for each non-grouping parameter, passing the parameter object, value, a constructed unique name, and a display label.

If the callback returns a value, it will be replace the old value.

FIXME: There is redundancy between this and the visit_inputs methods of Repeat and Group. This tracks labels and those do not. It would be nice to unify all the places that recursively visit inputs.

basic Module Basic tool parameters.

class galaxy.tools.parameters.basic.**BaseDataToolParameter**(*tool*, *input_source*, *trans*)

Bases: *galaxy.tools.parameters.basic.ToolParameter*

class galaxy.tools.parameters.basic.**BaseURLToolParameter**(*tool*, *input_source*)

Bases: *galaxy.tools.parameters.basic.HiddenToolParameter*

Returns a parameter that contains its value prepended by the current server base url. Used in all redirects.

from_html(*value=None*, *trans=None*, *context={}*)

class galaxy.tools.parameters.basic.**BooleanToolParameter**(*tool*, *input_source*)

Bases: *galaxy.tools.parameters.basic.ToolParameter*

Parameter that takes one of two values.

```
>>> p = BooleanToolParameter( None, XML( '<param name="blah" type="boolean" checked="yes" trueva
>>> print p.name
blah
>>> print p.get_html()
<input type="checkbox" id="blah" name="blah" value="true" checked="checked"><input type="hidden"
>>> print p.from_html( ["true","true"] )
True
>>> print p.to_param_dict_string( True )
bulletproof vests
>>> print p.from_html( ["true"] )
False
>>> print p.to_param_dict_string( False )
cellophane chests
```

from_html(*value*, *trans=None*, *other_values={}*)

from_json(*value*, *trans=None*, *other_values={}*)

get_html_field(*trans=None*, *value=None*, *other_values={}*)

get_initial_value(*trans*, *context*, *history=None*)

legal_values

to_dict(*trans*, *view='collection'*, *value_mapper=None*, *other_values={}*)

to_html_value(*value*, *app*)

to_param_dict_string(*value*, *other_values={}*)

to_python(*value*, *app*)

class galaxy.tools.parameters.basic.**ColorToolParameter**(*tool*, *input_source*)

Bases: *galaxy.tools.parameters.basic.ToolParameter*

Parameter that stores a color.

```
>>> p = ColorToolParameter( None, XML( '<param name="blah" type="color" value="#ffffff"/>' ) )
>>> print p.name
blah
```

get_html_field (*trans=None*, *value=None*, *other_values={}*)

get_initial_value (*trans*, *context*, *history=None*)

class galaxy.tools.parameters.basic.**ColumnListParameter** (*tool*, *input_source*)

Bases: *galaxy.tools.parameters.basic.SelectToolParameter*

Select list that consists of either the total number of columns or only those columns that contain numerical values in the associated DataToolParameter.

TODO: we need better testing here, but not sure how to associate a DatatoolParameter with a ColumnListParameter # from a twill perspective...

```
>>> # Mock up a history (not connected to database)
>>> from galaxy.model import History, HistoryDatasetAssociation
>>> from galaxy.util.bunch import Bunch
>>> from galaxy.model.mapping import init
>>> sa_session = init( "/tmp", "sqlite:///:memory:", create_tables=True ).session
>>> hist = History()
>>> sa_session.add( hist )
>>> sa_session.flush()
>>> hda = hist.add_dataset( HistoryDatasetAssociation( id=1, extension='interval', create_datase
>>> dtp =  DataToolParameter( None, XML( '<param name="blah" type="data" format="interval"/>' )
>>> print dtp.name
blah
>>> clp = ColumnListParameter ( None, XML( '<param name="numerical_column" type="data_column" da
>>> print clp.name
numerical_column
```

from_html (*value*, *trans=None*, *context={}*)

Label convention prepends column number with a 'c', but tool uses the integer. This removes the 'c' when entered into a workflow.

get_column_list (*trans*, *other_values*)

Generate a select list containing the columns of the associated dataset (if found).

get_dependencies ()

get_initial_value (*trans*, *context*, *history=None*)

get_legal_values (*trans*, *other_values*)

get_options (*trans*, *other_values*)

show column labels rather than c1..cn if use_header_names=True

to_dict (*trans*, *view='collection'*, *value_mapper=None*, *other_values={}*)

galaxy.tools.parameters.basic.**DEFAULT_VALUE_MAP** (*x*)

class galaxy.tools.parameters.basic.**DataCollectionToolParameter** (*tool*, *input_source*, *trans=None*)

Bases: *galaxy.tools.parameters.basic.BaseDataToolParameter*

collection_type

from_html (*value*, *trans*, *other_values={}*)

get_html_field(*trans=None*, *value=None*, *other_values={}*)

match_collections(*trans*, *history*, *dataset_matcher*)

match_multirun_collections(*trans*, *history*, *dataset_matcher*)

to_dict(*trans*, *view='collection'*, *value_mapper=None*, *other_values=None*)

to_python(*value*, *app*)

to_string(*value*, *app*)

validate(*value*, *history=None*)

value_to_display_text(*value*, *app*)

class galaxy.tools.parameters.basic.**DataToolParameter**(*tool*, *input_source*, *trans=None*)

Bases: galaxy.tools.parameters.basic.BaseDataToolParameter

Parameter that takes on one (or many) or a specific set of values.

TODO: There should be an alternate display that allows single selects to be displayed as radio buttons and multiple selects as a set of checkboxes

TODO: The following must be fixed to test correctly for the new security_check tag in the DataToolParameter (the last test below is broken) Nate's next pass at the dataset security stuff will dramatically alter this anyway.

converter_safe(*other_values*, *trans*)

from_html(*value*, *trans*, *other_values={}*)

get_dependencies()

Get the *names* of the other params this param depends on.

get_html_field(*trans=None*, *value=None*, *other_values={}*)

get_initial_value(*trans*, *context*, *history=None*)

get_initial_value_from_history_prevent_repeats(*trans*, *context*, *already_used*, *history=None*)

NOTE: This is wasteful since dynamic options and dataset collection happens twice (here and when generating HTML).

match_collections(*history*, *dataset_matcher*, *reduction=True*)

match_datasets(*history*, *dataset_matcher*)

to_dict(*trans*, *view='collection'*, *value_mapper=None*, *other_values=None*)

to_param_dict_string(*value*, *other_values={}*)

to_python(*value*, *app*)

to_string(*value*, *app*)

validate(*value*, *history=None*)

value_to_display_text(*value*, *app*)

class galaxy.tools.parameters.basic.**DrillDownSelectToolParameter**(*tool*, *input_source*, *context=None*)

Bases: galaxy.tools.parameters.basic.SelectToolParameter

Parameter that takes on one (or many) of a specific set of values. Creating a hierarchical select menu, which allows users to 'drill down' a tree-like set of options.

```
>>> p = DrillDownSelectToolParameter( None, XML(
... '''
... <param name="some_name" type="drill_down" display="checkbox" hierarchy="recurse" multiple="t
...   <options>
...    <option name="Heading 1" value="heading1">
...        <option name="Option 1" value="option1"/>
...        <option name="Option 2" value="option2"/>
...        <option name="Heading 1" value="heading1">
...          <option name="Option 3" value="option3"/>
...          <option name="Option 4" value="option4"/>
...        </option>
...    </option>
...    <option name="Option 5" value="option5"/>
...   </options>
... </param>
... ''' ) )
>>> print p.get_html()
<div class="form-row drilldown-container" id="drilldown--736f6d655f6e616d65">
<div class="form-row-input">
<div><span class="form-toggle icon-button toggle-expand" id="drilldown--736f6d655f6e616d65-68656
<input type="checkbox" name="some_name" value="heading1" >Heading 1
</div><div class="form-row" id="drilldown--736f6d655f6e616d65-68656164696e6731-container" style=
<div class="form-row-input">
<input type="checkbox" name="some_name" value="option1" >Option 1
</div>
<div class="form-row-input">
<input type="checkbox" name="some_name" value="option2" >Option 2
</div>
<div class="form-row-input">
<div><span class="form-toggle icon-button toggle-expand" id="drilldown--736f6d655f6e616d65-68656
<input type="checkbox" name="some_name" value="heading1" >Heading 1
</div><div class="form-row" id="drilldown--736f6d655f6e616d65-68656164696e6731-68656164696e6731-
<div class="form-row-input">
<input type="checkbox" name="some_name" value="option3" >Option 3
</div>
<div class="form-row-input">
<input type="checkbox" name="some_name" value="option4" >Option 4
</div>
</div>
</div>
</div>
</div>
<div class="form-row-input">
<input type="checkbox" name="some_name" value="option5" >Option 5
</div>
</div>
>>> p = DrillDownSelectToolParameter( None, XML(
... '''
... <param name="some_name" type="drill_down" display="radio" hierarchy="recurse" multiple="fals
...   <options>
...    <option name="Heading 1" value="heading1">
...        <option name="Option 1" value="option1"/>
...        <option name="Option 2" value="option2"/>
...        <option name="Heading 1" value="heading1">
...          <option name="Option 3" value="option3"/>
...          <option name="Option 4" value="option4"/>
...        </option>
...    </option>
```

```
...     <option name="Option 5" value="option5"/>
...    </options>
... </param>
... ''' ) )
>>> print p.get_html()
<div class="form-row drilldown-container" id="drilldown--736f6d655f6e616d65">
<div class="form-row-input">
<div><span class="form-toggle icon-button toggle-expand" id="drilldown--736f6d655f6e616d65-68656
<input type="radio" name="some_name" value="heading1" >Heading 1
</div><div class="form-row" id="drilldown--736f6d655f6e616d65-68656164696e6731-container" style=
<div class="form-row-input">
<input type="radio" name="some_name" value="option1" >Option 1
</div>
<div class="form-row-input">
<input type="radio" name="some_name" value="option2" >Option 2
</div>
<div class="form-row-input">
<div><span class="form-toggle icon-button toggle-expand" id="drilldown--736f6d655f6e616d65-68656
<input type="radio" name="some_name" value="heading1" >Heading 1
</div><div class="form-row" id="drilldown--736f6d655f6e616d65-68656164696e6731-68656164696e6731-
<div class="form-row-input">
<input type="radio" name="some_name" value="option3" >Option 3
</div>
<div class="form-row-input">
<input type="radio" name="some_name" value="option4" >Option 4
</div>
</div>
</div>
</div>
</div>
<div class="form-row-input">
<input type="radio" name="some_name" value="option5" >Option 5
</div>
</div>
>>> print sorted(p.options[1].items())
[('name', 'Option 5'), ('options', []), ('selected', False), ('value', 'option5')]
>>> p.options[0]["name"]
'Heading 1'
>>> p.options[0]["selected"]
False
```

from_html (*value*, *trans=None*, *other_values={}*)

get_dependencies ()

Get the *names* of the other params this param depends on.

get_html (*trans=None*, *value=None*, *other_values={}*)

Returns the html widget corresponding to the paramter. Optionally attempt to retain the current value specific by 'value'

get_html_field (*trans=None*, *value=None*, *other_values={}*)

get_initial_value (*trans*, *context*, *history=None*)

get_legal_values (*trans*, *other_values*)

get_options (*trans=None*, *value=None*, *other_values={}*)

to_dict (*trans*, *view='collection'*, *value_mapper=None*, *other_values={}*)

to_param_dict_string (*value*, *other_values={}*, *value_map=<function <lambda>>*)

value_to_display_text(*value*, *app*)

class galaxy.tools.parameters.basic.**DummyDataset**
Bases: object

class galaxy.tools.parameters.basic.**FTPFileToolParameter**(*tool*, *input_source*)
Bases: *galaxy.tools.parameters.basic.ToolParameter*

Parameter that takes a file uploaded via FTP as a value.

from_html(*value*, *trans=None*, *other_values={}*)

get_html_field(*trans=None*, *value=None*, *other_values={}*)

get_initial_value(*trans*, *context*, *history=None*)

to_python(*value*, *app*)

to_string(*value*, *app*)

visible

class galaxy.tools.parameters.basic.**FileToolParameter**(*tool*, *input_source*)
Bases: *galaxy.tools.parameters.basic.ToolParameter*

Parameter that takes an uploaded file as a value.

```
>>> p = FileToolParameter( None, XML( '<param name="blah" type="file"/>' ) )
>>> print p.name
blah
>>> print p.get_html()
<input type="file" name="blah">
>>> p = FileToolParameter( None, XML( '<param name="blah" type="file" ajax-upload="true"/>' ) )
>>> print p.get_html()
<input type="file" name="blah" galaxy-ajax-upload="true">
```

from_html(*value*, *trans=None*, *other_values={}*)

get_html_field(*trans=None*, *value=None*, *other_values={}*)

get_initial_value(*trans*, *context*, *history=None*)

get_required_enctype()
File upload elements require the multipart/form-data encoding

to_python(*value*, *app*)

to_string(*value*, *app*)

class galaxy.tools.parameters.basic.**FloatToolParameter**(*tool*, *input_source*)
Bases: *galaxy.tools.parameters.basic.TextToolParameter*

Parameter that takes a real number value.

```
>>> p = FloatToolParameter( None, XML( '<param name="blah" type="float" size="4" value="3.141592
>>> print p.name
blah
>>> print p.get_html()
<input type="text" name="blah" size="4" value="3.141592">
>>> type( p.from_html( "36.1" ) )
<type 'float'>
>>> type( p.from_html( "bleh" ) )
Traceback (most recent call last):
```

```
    ...
ValueError: A real number is required
```

dict_collection_visible_keys = ('name', 'argument', 'type', 'label', 'help', 'min', 'max')

from_html(*value*, *trans=None*, *other_values={}*)

get_html_field(*trans=None*, *value=None*, *other_values={}*)

get_initial_value(*trans*, *context*, *history=None*)

to_python(*value*, *app*)

class galaxy.tools.parameters.basic.**GenomeBuildParameter**(**args*, ***kwds*)

Bases: *galaxy.tools.parameters.basic.SelectToolParameter*

Select list that sets the last used genome build for the current history as "selected".

```
>>> # Create a mock transaction with 'hg17' as the current build
>>> from galaxy.util.bunch import Bunch
>>> trans = Bunch( history=Bunch( genome_build='hg17' ), db_builds=util.read_dbnames( None ) )
```

```
>>> p = GenomeBuildParameter( None, XML(
... '''
... <param name="blah" type="genomebuild" />
... ''' ) )
>>> print p.name
blah
```

```
>>> # hg17 should be selected by default
>>> print p.get_html( trans )
<select name="blah" last_selected_value="hg17">
<option value="?">unspecified (?)</option>
...
<option value="hg18">Human Mar. 2006 (NCBI36/hg18) (hg18)</option>
<option value="hg17" selected>Human May 2004 (NCBI35/hg17) (hg17)</option>
...
</select>
```

```
>>> # If the user selected something else already, that should be used
>>> # instead
>>> print p.get_html( trans, value='hg18' )
<select name="blah" last_selected_value="hg18">
<option value="?">unspecified (?)</option>
...
<option value="hg18" selected>Human Mar. 2006 (NCBI36/hg18) (hg18)</option>
<option value="hg17">Human May 2004 (NCBI35/hg17) (hg17)</option>
...
</select>
```

```
>>> print p.filter_value( "hg17" )
hg17
```

get_legal_values(*trans*, *other_values*)

get_options(*trans*, *other_values*)

to_dict (*trans*, *view='collection'*, *value_mapper=None*, *other_values={}*)

class galaxy.tools.parameters.basic.**HiddenDataToolParameter** (*tool*, *elem*)
Bases: *galaxy.tools.parameters.basic.HiddenToolParameter*, *galaxy.tools.parameters.basic.DataToolParameter*

Hidden parameter that behaves as a DataToolParameter. As with all hidden parameters, this is a HACK.

get_html_field (*trans=None*, *value=None*, *other_values={}*)

get_initial_value (*trans*, *context*, *history=None*)

class galaxy.tools.parameters.basic.**HiddenToolParameter** (*tool*, *input_source*)
Bases: *galaxy.tools.parameters.basic.ToolParameter*

Parameter that takes one of two values.
FIXME: This seems hacky, parameters should only describe things the user might change. It is used for 'initializing' the UCSC proxy tool

```
>>> p = HiddenToolParameter( None, XML( '<param name="blah" type="hidden" value="wax so rockin"/
>>> print p.name
blah
>>> print p.get_html()
<input type="hidden" name="blah" value="wax so rockin">
```

get_html_field (*trans=None*, *value=None*, *other_values={}*)

get_initial_value (*trans*, *context*, *history=None*)

get_label ()

class galaxy.tools.parameters.basic.**IntegerToolParameter** (*tool*, *input_source*)
Bases: *galaxy.tools.parameters.basic.TextToolParameter*

Parameter that takes an integer value.

```
>>> p = IntegerToolParameter( None, XML( '<param name="blah" type="integer" size="4" value="10"
>>> print p.name
blah
>>> print p.get_html()
<input type="text" name="blah" size="4" value="10">
>>> type( p.from_html( "10" ) )
<type 'int'>
>>> type( p.from_html( "bleh" ) )
Traceback (most recent call last):
    ...
ValueError: An integer is required
```

dict_collection_visible_keys = ('name', 'argument', 'type', 'label', 'help', 'min', 'max')

from_html (*value*, *trans=None*, *other_values={}*)

get_html_field (*trans=None*, *value=None*, *other_values={}*)

get_initial_value (*trans*, *context*, *history=None*)

to_python (*value*, *app*)

class galaxy.tools.parameters.basic.**LibraryDatasetToolParameter** (*tool*, *elem*)
Bases: *galaxy.tools.parameters.basic.ToolParameter*

Parameter that lets users select a LDDA from a modal window, then use it within the wrapper.

from_html (*value*, *trans*, *other_values={}*)

get_html_field(*trans=None*, *value=None*, *other_values={}*)

get_initial_value(*trans*, *context*, *history=None*)

to_python(*value*, *app*)

to_string(*value*, *app*)

class galaxy.tools.parameters.basic.**RuntimeValue**

Bases: object

Wrapper to note a value that is not yet set, but will be required at runtime.

class galaxy.tools.parameters.basic.**SelectToolParameter**(*tool*, *input_source*, *context=None*)

Bases: *galaxy.tools.parameters.basic.ToolParameter*

Parameter that takes on one (or many) or a specific set of values.

```
>>> p = SelectToolParameter( None, XML(
... '''
... <param name="blah" type="select">
...     <option value="x">I am X</option>
...     <option value="y" selected="true">I am Y</option>
...     <option value="z">I am Z</option>
... </param>
... ''' ) )
>>> print p.name
blah
>>> print p.get_html()
<select name="blah" last_selected_value="y">
<option value="x">I am X</option>
<option value="y" selected>I am Y</option>
<option value="z">I am Z</option>
</select>
>>> print p.get_html( value="z" )
<select name="blah" last_selected_value="z">
<option value="x">I am X</option>
<option value="y">I am Y</option>
<option value="z" selected>I am Z</option>
</select>
>>> print p.filter_value( "y" )
y
```

```
>>> p = SelectToolParameter( None, XML(
... '''
... <param name="blah" type="select" multiple="true">
...     <option value="x">I am X</option>
...     <option value="y" selected="true">I am Y</option>
...     <option value="z" selected="true">I am Z</option>
... </param>
... ''' ) )
>>> print p.name
blah
>>> print p.get_html()
<select name="blah" multiple last_selected_value="z">
<option value="x">I am X</option>
<option value="y" selected>I am Y</option>
<option value="z" selected>I am Z</option>
</select>
```

```
>>> print p.get_html( value=["x","y"])
<select name="blah" multiple last_selected_value="y">
<option value="x" selected>I am X</option>
<option value="y" selected>I am Y</option>
<option value="z">I am Z</option>
</select>
>>> print p.to_param_dict_string( ["y", "z"] )
y,z
```

```
>>> p = SelectToolParameter( None, XML(
... '''
... <param name="blah" type="select" multiple="true" display="checkboxes">
...     <option value="x">I am X</option>
...     <option value="y" selected="true">I am Y</option>
...     <option value="z" selected="true">I am Z</option>
... </param>
... ''' ) )
>>> print p.name
blah
>>> print p.get_html()
<div class="checkUncheckAllPlaceholder" checkbox_name="blah"></div>
<div><input type="checkbox" name="blah" value="x" id="blah|x"><label class="inline" for="blah|x"
<div class="odd_row"><input type="checkbox" name="blah" value="y" id="blah|y" checked='checked'>
<div><input type="checkbox" name="blah" value="z" id="blah|z" checked='checked'><label class="in
>>> print p.get_html( value=["x","y"])
<div class="checkUncheckAllPlaceholder" checkbox_name="blah"></div>
<div><input type="checkbox" name="blah" value="x" id="blah|x" checked='checked'><label class="in
<div class="odd_row"><input type="checkbox" name="blah" value="y" id="blah|y" checked='checked'>
<div><input type="checkbox" name="blah" value="z" id="blah|z"><label class="inline" for="blah|z"
>>> print p.to_param_dict_string( ["y", "z"] )
y,z
```

from_html (*value*, *trans=None*, *context={}*)

get_dependencies ()

Get the *names* of the other params this param depends on.

get_html_field (*trans=None*, *value=None*, *context={}*)

get_initial_value (*trans*, *context*, *history=None*)

get_legal_values (*trans*, *other_values*)

get_options (*trans*, *other_values*)

to_dict (*trans*, *view='collection'*, *value_mapper=None*, *other_values={}*)

to_html_value (*value*, *app*)

to_param_dict_string (*value*, *other_values={}*, *value_map=<function <lambda>>*)

value_from_basic (*value*, *app*, *ignore_errors=False*)

value_to_basic (*value*, *app*)

value_to_display_text (*value*, *app*)

class galaxy.tools.parameters.basic.**TextToolParameter** (*tool*, *input_source*)

Bases: *galaxy.tools.parameters.basic.ToolParameter*

Parameter that can take on any text value.

```
>>> p = TextToolParameter( None, XML( '<param name="blah" type="text" size="4" value="default" /
>>> print p.name
blah
>>> print p.get_html()
<input type="text" name="blah" size="4" value="default">
>>> print p.get_html( value="meh" )
<input type="text" name="blah" size="4" value="meh">
```

get_html_field(*trans=None*, *value=None*, *other_values={}*)

get_initial_value(*trans*, *context*, *history=None*)

to_dict(*trans*, *view='collection'*, *value_mapper=None*, *other_values={}*)

to_html_value(*value*, *app*)

to_string(*value*, *app*)
Convert a value to a string representation suitable for persisting

class galaxy.tools.parameters.basic.**ToolParameter**(*tool*, *input_source*, *context=None*)
Bases: object, *galaxy.model.item_attrs.Dictifiable*

Describes a parameter accepted by a tool. This is just a simple stub at the moment but in the future should encapsulate more complex parameters (lists of valid choices, validation logic, ...)

classmethod build(*tool*, *param*)
Factory method to create parameter of correct type

dict_collection_visible_keys = ('name', 'argument', 'type', 'label', 'help')

filter_value(*value*, *trans=None*, *other_values={}*)
Parse the value returned by the view into a form usable by the tool OR raise a ValueError.

from_html(*value*, *trans=None*, *other_values={}*)
Convert a value from an HTML POST into the parameters preferred value format.

from_json(*value*, *trans=None*, *other_values={}*)

get_dependencies()
Return the names of any other parameters this parameter depends on

get_html(*trans=None*, *value=None*, *other_values={}*)
Returns the html widget corresponding to the parameter. Optionally attempt to retain the current value specific by 'value'

get_html_field(*trans=None*, *value=None*, *other_values={}*)

get_initial_value(*trans*, *context*, *history=None*)
Return the starting value of the parameter

get_initial_value_from_history_prevent_repeats(*trans*, *context*, *already_used*, *history=None*)
Get the starting value for the parameter, but if fetching from the history, try to find a value that has not yet been used. already_used is a list of objects that tools must manipulate (by adding to it) to store a memento that they can use to detect if a value has already been chosen from the history. This is to support the capability to choose each dataset once

get_label()
Return user friendly name for the parameter

get_required_enctype()
If this parameter needs the form to have a specific encoding return it, otherwise return None (indicating compatibility with any encoding)

classmethod parse_name (*input_source*)

to_dict (*trans*, *view='collection'*, *value_mapper=None*, *other_values={}*)
to_dict tool parameter. This can be overridden by subclasses.

to_html_value (*value*, *app*)
Convert an object value to the value expected from an html post

to_param_dict_string (*value*, *other_values={}*)
Called via __str__ when used in the Cheetah template

to_python (*value*, *app*)
Convert a value created with to_string back to an object representation

to_string (*value*, *app*)
Convert a value to a string representation suitable for persisting

validate (*value*, *history=None*)

value_from_basic (*value*, *app*, *ignore_errors=False*)

value_to_basic (*value*, *app*)

value_to_display_text (*value*, *app*)
Convert a value to a text representation suitable for displaying to the user

visible
Return true if the parameter should be rendered on the form

class galaxy.tools.parameters.basic.**UnvalidatedValue** (*value*)
Bases: object

Wrapper to mark a value that has not been validated

dynamic_options Module Support for generating the options for a SelectToolParameter dynamically (based on the values of other parameters or other aspects of the current state)

class galaxy.tools.parameters.dynamic_options.**AdditionalValueFilter** (*d_option*, *elem*)
Bases: *galaxy.tools.parameters.dynamic_options.Filter*

Adds a single static value to an options list.

Type: add_value
Required Attributes: value: value to appear in select list
Optional Attributes: name: Display name to appear in select list (value) index: Index of option list to add value (APPEND)
filter_options (*options*, *trans*, *other_values*)

class galaxy.tools.parameters.dynamic_options.**AttributeValueSplitterFilter** (*d_option*, *elem*)
Bases: *galaxy.tools.parameters.dynamic_options.Filter*

Filters a list of attribute-value pairs to be unique attribute names.

Type: attribute_value_splitter
Required Attributes: column: column in options to compare with
Optional Attributes: pair_separator: Split column by this (,) name_val_separator: Split name-value pair by this (whitespace)
filter_options (*options*, *trans*, *other_values*)

class galaxy.tools.parameters.dynamic_options.**DataMetaFilter** (*d_option*, *elem*)
Bases: *galaxy.tools.parameters.dynamic_options.Filter*

Filters a list of options on a column by a dataset metadata value.

Type: data_meta

When no 'from' source has been specified in the <options> tag, this will populate the options list with (meta_value, meta_value, False). Otherwise, options which do not match the metadata value in the column are discarded.

Required Attributes:
- •ref: Name of input dataset
- •key: Metadata key to use for comparison
- •column: column in options to compare with (not required when not associated with input options)

Optional Attributes:
- •multiple: Option values are multiple, split column by separator (True)
- •separator: When multiple split by this (,)

filter_options(*options*, *trans*, *other_values*)

get_dependency_name()

class galaxy.tools.parameters.dynamic_options.**DynamicOptions**(*elem*, *tool_param*)
Bases: object

Handles dynamically generated SelectToolParameter options

column_spec_to_index(*column_spec*)
Convert a column specification (as read from the config file), to an index. A column specification can just be a number, a column name, or a column alias.

get_dependency_names()
Return the names of parameters these options depend on – both data and other param types.

get_field_by_name_for_value(*field_name*, *value*, *trans*, *other_values*)
Get contents of field by name for specified value.

get_fields(*trans*, *other_values*)

get_fields_by_value(*value*, *trans*, *other_values*)
Return a list of fields with column 'value' matching provided value.

get_options(*trans*, *other_values*)

parse_column_definitions(*elem*)

parse_file_fields(*reader*)

class galaxy.tools.parameters.dynamic_options.**Filter**(*d_option*, *elem*)
Bases: object

A filter takes the current options list and modifies it.

filter_options(*options*, *trans*, *other_values*)
Returns a list of options after the filter is applied

classmethod from_element(*d_option*, *elem*)
Loads the proper filter by the type attribute of elem

get_dependency_name()
Returns the name of any depedencies, otherwise None

class galaxy.tools.parameters.dynamic_options.**MultipleSplitterFilter**(*d_option*, *elem*)
Bases: *galaxy.tools.parameters.dynamic_options.Filter*

Turns a single line of options into multiple lines, by splitting a column and creating a line for each item.

Type: multiple_splitter
Required Attributes: column: column in options to compare with
Optional Attributes: separator: Split column by this (,)
filter_options(*options*, *trans*, *other_values*)

class galaxy.tools.parameters.dynamic_options.**ParamValueFilter**(*d_option*, *elem*)
Bases: *galaxy.tools.parameters.dynamic_options.Filter*

Filters a list of options on a column by the value of another input.

Type: param_value

Required Attributes:
- ref: Name of input value
- column: column in options to compare with

Optional Attributes:
- **keep: Keep columns matching value (True)** Discard columns matching value (False)
- ref_attribute: Period (.) separated attribute chain of input (ref) to use as value for filter

filter_options(*options*, *trans*, *other_values*)

get_dependency_name()

class galaxy.tools.parameters.dynamic_options.**RemoveValueFilter**(*d_option*, *elem*)
Bases: *galaxy.tools.parameters.dynamic_options.Filter*

Removes a value from an options list.

Type: remove_value

Required Attributes:

```
value: value to remove from select list
    or
ref: param to refer to
    or
meta_ref: dataset to refer to
key: metadata key to compare to
```

filter_options(*options*, *trans*, *other_values*)

class galaxy.tools.parameters.dynamic_options.**SortByColumnFilter**(*d_option*, *elem*)
Bases: *galaxy.tools.parameters.dynamic_options.Filter*

Sorts an options list by a column

Type: sort_by
Required Attributes: column: column to sort by
filter_options(*options*, *trans*, *other_values*)

class galaxy.tools.parameters.dynamic_options.**StaticValueFilter**(*d_option*, *elem*)
Bases: *galaxy.tools.parameters.dynamic_options.Filter*

Filters a list of options on a column by a static value.

Type: static_value
Required Attributes: value: static value to compare to column: column in options to compare with
Optional Attributes:

keep: Keep columns matching value (True) Discard columns matching value (False)

filter_options(*options*, *trans*, *other_values*)

class galaxy.tools.parameters.dynamic_options.**UniqueValueFilter**(*d_option*, *elem*)
Bases: *galaxy.tools.parameters.dynamic_options.Filter*

Filters a list of options to be unique by a column value.

Type: unique_value
Required Attributes: column: column in options to compare with
filter_options(*options*, *trans*, *other_values*)

get_dependency_name()

grouping Module Constructs for grouping tool parameters

class galaxy.tools.parameters.grouping.**Conditional**
Bases: *galaxy.tools.parameters.grouping.Group*

get_current_case(*value*, *trans*)

get_initial_value(*trans*, *context*, *history=None*)

is_job_resource_conditional

label

to_dict(*trans*, *view='collection'*, *value_mapper=None*)

type = 'conditional'

value_from_basic(*value*, *app*, *ignore_errors=False*)

value_to_basic(*value*, *app*)

visit_inputs(*prefix*, *value*, *callback*)

class galaxy.tools.parameters.grouping.**ConditionalWhen**
Bases: object, *galaxy.model.item_attrs.Dictifiable*

dict_collection_visible_keys = ('value',)

to_dict(*trans*, *view='collection'*, *value_mapper=None*)

class galaxy.tools.parameters.grouping.**Group**
Bases: object, *galaxy.model.item_attrs.Dictifiable*

dict_collection_visible_keys = ('name', 'type')

get_initial_value(*trans*, *context*, *history=None*)
Return the initial state/value for this group

to_dict(*trans*, *view='collection'*, *value_mapper=None*)

value_from_basic(*value*, *app*, *ignore_errors=False*)
Convert a basic representation as produced by *value_to_basic* back into the preferred value form.

value_to_basic(*value*, *app*)
Convert value to a (possibly nested) representation using only basic types (dict, list, tuple, str, unicode, int, long, float, bool, None)

visible

class galaxy.tools.parameters.grouping.**Repeat**
Bases: *galaxy.tools.parameters.grouping.Group*

dict_collection_visible_keys = ('name', 'type', 'title', 'help', 'default', 'min', 'max')

get_initial_value(*trans*, *context*, *history=None*)

label()

title_plural

to_dict(*trans*, *view='collection'*, *value_mapper=None*)

type = 'repeat'

value_from_basic(*value*, *app*, *ignore_errors=False*)

value_to_basic(*value*, *app*)

visit_inputs(*prefix*, *value*, *callback*)

class galaxy.tools.parameters.grouping.**Section**

Bases: galaxy.tools.parameters.grouping.Group

dict_collection_visible_keys = ('name', 'type', 'title', 'help', 'expanded')

get_initial_value(*trans*, *context*, *history=None*)

label()

title_plural

to_dict(*trans*, *view='collection'*, *value_mapper=None*)

type = 'section'

value_from_basic(*value*, *app*, *ignore_errors=False*)

value_to_basic(*value*, *app*)

visit_inputs(*prefix*, *value*, *callback*)

class galaxy.tools.parameters.grouping.**UploadDataset**

Bases: galaxy.tools.parameters.grouping.Group

get_composite_dataset_name(*context*)

get_datatype(*trans*, *context*)

get_datatype_ext(*trans*, *context*)

get_file_base_name(*context*)

get_file_type(*context*)

get_initial_value(*trans*, *context*, *history=None*)

get_uploaded_datasets(*trans*, *context*, *override_name=None*, *override_info=None*)

group_title(*context*)

title_by_index(*trans*, *index*, *context*)

title_plural

type = 'upload_dataset'

value_from_basic(*value*, *app*, *ignore_errors=False*)

value_to_basic(*value*, *app*)

visit_inputs(*prefix*, *value*, *callback*)

input_translation Module Tool Input Translation.

class galaxy.tools.parameters.input_translation.**ToolInputTranslator**

Bases: object

Handles Tool input translation. This is used for data source tools

```
>>> from galaxy.util import Params
>>> from xml.etree.ElementTree import XML
>>> translator = ToolInputTranslator.from_element( XML(
... '''
... <request_param_translation>
...  <request_param galaxy_name="URL_method" remote_name="URL_method" missing="post" />
...  <request_param galaxy_name="URL" remote_name="URL" missing="" >
...     <append_param separator="&amp;" first_separator="?" join="=">
...         <value name="_export" missing="1" />
...         <value name="GALAXY_URL" missing="0" />
...     </append_param>
...  </request_param>
...  <request_param galaxy_name="dbkey" remote_name="db" missing="?" />
...  <request_param galaxy_name="organism" remote_name="org" missing="unknown species" />
...  <request_param galaxy_name="table" remote_name="hgta_table" missing="unknown table" />
...  <request_param galaxy_name="description" remote_name="hgta_regionType" missing="no descript
...  <request_param galaxy_name="data_type" remote_name="hgta_outputType" missing="tabular" >
...   <value_translation>
...    <value galaxy_value="tabular" remote_value="primaryTable" />
...    <value galaxy_value="tabular" remote_value="selectedFields" />
...    <value galaxy_value="wig" remote_value="wigData" />
...    <value galaxy_value="interval" remote_value="tab" />
...    <value galaxy_value="html" remote_value="hyperlinks" />
...    <value galaxy_value="fasta" remote_value="sequence" />
...   </value_translation>
...  </request_param>
... </request_param_translation>
... ''' ) )
>>> params = Params( { 'db':'hg17', 'URL':'URL_value', 'org':'Human', 'hgta_outputType':'primary
>>> translator.translate( params )
>>> print sorted(list(params.__dict__.keys()))
['URL', 'URL_method', 'data_type', 'db', 'dbkey', 'description', 'hgta_outputType', 'org', 'orga
>>> params.get('URL', None) in ['URL_value?GALAXY_URL=0&_export=1', 'URL_value?_export=1&GALAXY_
True
```

classmethod from_element (*elem*)

Loads the proper filter by the type attribute of elem

translate (*params*)

update params in-place

output Module Support for dynamically modifying output attributes.

class galaxy.tools.parameters.output.**BooleanFilter** (*parent*, *elem*)

Bases: *galaxy.tools.parameters.output.ToolOutputActionOptionFilter*

filter_options (*options*, *other_values*)

tag = 'boolean'

class galaxy.tools.parameters.output.**ColumnReplaceFilter** (*parent*, *elem*)

Bases: *galaxy.tools.parameters.output.ToolOutputActionOptionFilter*

filter_options (*options*, *other_values*)

tag = 'column_replace'

class galaxy.tools.parameters.output.**ColumnStripFilter** (*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputActionOptionFilter

filter_options (*options*, *other_values*)

tag = 'column_strip'

class galaxy.tools.parameters.output.**DatatypeIsInstanceToolOutputActionConditionalWhen** (*parent*, *config_elem value*)
Bases: galaxy.tools.parameters.output.ToolOutputActionConditionalWhen

is_case (*output_dataset*, *other_values*)

tag = 'when datatype_isinstance'

class galaxy.tools.parameters.output.**FormatToolOutputAction** (*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputAction

apply_action (*output_dataset*, *other_values*)

tag = 'format'

class galaxy.tools.parameters.output.**FromDataTableOutputActionOption** (*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputActionOption

get_value (*other_values*)

tag = 'from_data_table'

class galaxy.tools.parameters.output.**FromFileToolOutputActionOption** (*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputActionOption

get_value (*other_values*)

tag = 'from_file'

class galaxy.tools.parameters.output.**FromParamToolOutputActionOption** (*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputActionOption

get_value (*other_values*)

tag = 'from_param'

class galaxy.tools.parameters.output.**InsertColumnToolOutputActionOptionFilter** (*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputActionOptionFilter

filter_options (*options*, *other_values*)

tag = 'insert_column'

class galaxy.tools.parameters.output.**MetadataToolOutputAction** (*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputAction

apply_action (*output_dataset*, *other_values*)

tag = 'metadata'

class galaxy.tools.parameters.output.**MetadataValueFilter**(*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputActionOptionFilter

filter_options(*options*, *other_values*)

tag = 'metadata_value'

class galaxy.tools.parameters.output.**MultipleSplitterFilter**(*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputActionOptionFilter

filter_options(*options*, *other_values*)

tag = 'multiple_splitter'

class galaxy.tools.parameters.output.**NullToolOutputActionOption**(*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputActionOption

get_value(*other_values*)

tag = 'null_option'

class galaxy.tools.parameters.output.**ParamValueToolOutputActionOptionFilter**(*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputActionOptionFilter

filter_options(*options*, *other_values*)

tag = 'param_value'

class galaxy.tools.parameters.output.**StringFunctionFilter**(*parent*, *elem*)
Bases: galaxy.tools.parameters.output.ToolOutputActionOptionFilter

filter_options(*options*, *other_values*)

tag = 'string_function'

class galaxy.tools.parameters.output.**ToolOutputAction**(*parent*, *elem*)
Bases: object

apply_action(*output_dataset*, *other_values*)

classmethod from_elem(*parent*, *elem*)
Loads the proper action by the type attribute of elem

tag = 'action'

tool

class galaxy.tools.parameters.output.**ToolOutputActionConditional**(*parent*, *config_elem*)
Bases: object

apply_action(*output_dataset*, *other_values*)

tag = 'conditional'

tool

class galaxy.tools.parameters.output.**ToolOutputActionConditionalWhen**(*parent*, *config_elem*, *value*)
Bases: galaxy.tools.parameters.output.ToolOutputActionGroup

apply_action(*output_dataset*, *other_values*)

classmethod from_elem(*parent*, *when_elem*)
Loads the proper when by attributes of elem

get_ref (*output_dataset*, *other_values*)

is_case (*output_dataset*, *other_values*)

tag = 'when'

class galaxy.tools.parameters.output.**ToolOutputActionGroup** (*parent*, *config_elem*)
Bases: object

Manages a set of tool output dataset actions directives

apply_action (*output_dataset*, *other_values*)

tag = 'group'

tool

class galaxy.tools.parameters.output.**ToolOutputActionOption** (*parent*, *elem*)
Bases: object

classmethod from_elem (*parent*, *elem*)
Loads the proper action by the type attribute of elem

get_value (*other_values*)

tag = 'object'

tool

class galaxy.tools.parameters.output.**ToolOutputActionOptionFilter** (*parent*, *elem*)
Bases: object

filter_options (*options*, *other_values*)

classmethod from_elem (*parent*, *elem*)
Loads the proper action by the type attribute of elem

tag = 'filter'

tool

class galaxy.tools.parameters.output.**ValueToolOutputActionConditionalWhen** (*parent*, *config_elem*, *value*)
Bases: *galaxy.tools.parameters.output.ToolOutputActionConditionalWhen*

is_case (*output_dataset*, *other_values*)

tag = 'when value'

galaxy.tools.parameters.output.**action_type**
alias of *FormatToolOutputAction*

galaxy.tools.parameters.output.**compare_endswith** (*value1*, *value2*)

galaxy.tools.parameters.output.**compare_eq** (*value1*, *value2*)

galaxy.tools.parameters.output.**compare_gt** (*value1*, *value2*)

galaxy.tools.parameters.output.**compare_gte** (*value1*, *value2*)

galaxy.tools.parameters.output.**compare_in** (*value1*, *value2*)

galaxy.tools.parameters.output.**compare_lt** (*value1*, *value2*)

galaxy.tools.parameters.output.**compare_lte** (*value1*, *value2*)

galaxy.tools.parameters.output.**compare_neq** (*value1*, *value2*)

galaxy.tools.parameters.output.**compare_re_search**(*value1*, *value2*)

galaxy.tools.parameters.output.**compare_startswith**(*value1*, *value2*)

galaxy.tools.parameters.output.**filter_type**
alias of *ColumnReplaceFilter*

galaxy.tools.parameters.output.**option_type**
alias of *FromDataTableOutputActionOption*

galaxy.tools.parameters.output.**parse_cast_attribute**(*cast*)

galaxy.tools.parameters.output.**parse_compare_type**(*compare*)

sanitize Module Tool Parameter specific sanitizing.

class galaxy.tools.parameters.sanitize.**ToolParameterSanitizer**
Bases: object

Handles tool parameter specific sanitizing.

```
>>> from xml.etree.ElementTree import XML
>>> sanitizer = ToolParameterSanitizer.from_element( XML(
... '''
... <sanitizer invalid_char="">
...   <valid initial="string.letters"/>
... </sanitizer>
... ''' ) )
>>> sanitizer.sanitize_param( ''.join( sorted( [ c for c in string.printable ] ) ) ) == ''.join(
True
>>> slash = chr( 92 )
>>> sanitizer = ToolParameterSanitizer.from_element( XML(
... '''
... <sanitizer>
...   <valid initial="none">
...    <add preset="string.printable"/>
...    <remove value="&quot;"/>
...    <remove value="%s"/>
...   </valid>
...   <mapping initial="none">
...     <add source="&quot;" target="%s&quot;"/>
...     <add source="%s" target="%s%s"/>
...   </mapping>
... </sanitizer>
... ''' % ( slash, slash, slash, slash, slash ) ) )
>>> text = '%s"$rm&#!' % slash
>>> [ c for c in sanitizer.sanitize_param( text ) ] == [ slash, slash, slash, '"', '$', 'r', 'm'
True
```

DEFAULT_INVALID_CHAR = 'X'

MAPPING_PRESET = {'default': {'@': '__at__', '\t': '__tc__', '\n': '__cn__', '\r': '__cr__', '[': '__ob__', ']': '__cb__', '#'

VALID_PRESET = {'default': 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 -=_.(

classmethod from_element(*elem*)
Loads the proper filter by the type attribute of elem

classmethod get_mapping_by_name(*name*)

classmethod get_valid_by_name(*name*)

restore_param(*value*)

restore_text(*text*)
Restores sanitized text

sanitize_param(*value*)
Clean incoming parameters (strings or lists)

sanitize_text(*text*)
Restricts the characters that are allowed in a text

validation Module Classes related to parameter validation.

class galaxy.tools.parameters.validation.**DatasetOkValidator**(*message=None*)
Bases: *galaxy.tools.parameters.validation.Validator*

Validator that checks if a dataset is in an 'ok' state

classmethod from_element(*param*, *elem*)

validate(*value*, *history=None*)

class galaxy.tools.parameters.validation.**EmptyTextfieldValidator**(*message=None*)
Bases: *galaxy.tools.parameters.validation.Validator*

Validator that checks for empty text field

classmethod from_element(*param*, *elem*)

validate(*value*, *history=None*)

class galaxy.tools.parameters.validation.**ExpressionValidator**(*message*, *expression*, *substitute_value_in_message*)
Bases: *galaxy.tools.parameters.validation.Validator*

Validator that evaluates a python expression using the value

```
>>> from galaxy.tools.parameters import ToolParameter
>>> p = ToolParameter.build( None, XML( '''
... <param name="blah" type="text" size="10" value="10">
...     <validator type="expression" message="Not gonna happen">value.lower() == "foo"</validato
... </param>
... ''' ) )
>>> t = p.validate( "Foo" )
>>> t = p.validate( "foo" )
>>> t = p.validate( "Fop" )
Traceback (most recent call last):
    ...
ValueError: Not gonna happen
```

classmethod from_element(*param*, *elem*)

validate(*value*, *history=None*)

class galaxy.tools.parameters.validation.**InRangeValidator**(*message*, *range_min*, *range_max*)
Bases: *galaxy.tools.parameters.validation.Validator*

Validator that ensures a number is in a specific range

```
>>> from galaxy.tools.parameters import ToolParameter
>>> p = ToolParameter.build( None, XML( '''
... <param name="blah" type="integer" size="10" value="10">
...     <validator type="in_range" message="Not gonna happen" min="10" max="20"/>
... </param>
... ''' ) )
>>> t = p.validate( 10 )
>>> t = p.validate( 15 )
>>> t = p.validate( 20 )
>>> t = p.validate( 21 )
Traceback (most recent call last):
    ...
ValueError: Not gonna happen
```

classmethod **from_element** (*param*, *elem*)

validate (*value*, *history=None*)

exception galaxy.tools.parameters.validation.**LateValidationError** (*message*)

Bases: exceptions.Exception

class galaxy.tools.parameters.validation.**LengthValidator** (*message*, *length_min*, *length_max*)

Bases: *galaxy.tools.parameters.validation.Validator*

Validator that ensures the length of the provided string (value) is in a specific range

```
>>> from galaxy.tools.parameters import ToolParameter
>>> p = ToolParameter.build( None, XML( '''
... <param name="blah" type="text" size="10" value="foobar">
...     <validator type="length" min="2" max="8"/>
... </param>
... ''' ) )
>>> t = p.validate( "foo" )
>>> t = p.validate( "bar" )
>>> t = p.validate( "f" )
Traceback (most recent call last):
    ...
ValueError: Must have length of at least 2
>>> t = p.validate( "foobarbaz" )
Traceback (most recent call last):
    ...
ValueError: Must have length no more than 8
```

classmethod **from_element** (*param*, *elem*)

validate (*value*, *history=None*)

class galaxy.tools.parameters.validation.**MetadataInDataTableColumnValidator**(*tool_data_table*, *metadata_name*, *metadata_column*, *message='Value for metadata not found.'*, *line_startswith=None*)

Bases: galaxy.tools.parameters.validation.Validator

Validator that checks if the value for a dataset's metadata item exists in a file.

classmethod from_element(*param*, *elem*)

validate(*value*, *history=None*)

class galaxy.tools.parameters.validation.**MetadataInFileColumnValidator**(*filename*, *metadata_name*, *metadata_column*, *message='Value for metadata not found.'*, *line_startswith=None*)

Bases: galaxy.tools.parameters.validation.Validator

Validator that checks if the value for a dataset's metadata item exists in a file.

classmethod from_element(*param*, *elem*)

validate(*value*, *history=None*)

class galaxy.tools.parameters.validation.**MetadataValidator**(*message=None*, *check=''*, *skip=''*)

Bases: galaxy.tools.parameters.validation.Validator

Validator that checks for missing metadata

classmethod from_element(*param*, *elem*)

validate(*value*, *history=None*)

class galaxy.tools.parameters.validation.**NoOptionsValidator**(*message=None*)

Bases: galaxy.tools.parameters.validation.Validator

Validator that checks for empty select list

classmethod from_element(*param*, *elem*)

validate(*value*, *history=None*)

class galaxy.tools.parameters.validation.**RegexValidator**(*message*, *expression*)

Bases: galaxy.tools.parameters.validation.Validator

Validator that evaluates a regular expression

```
>>> from galaxy.tools.parameters import ToolParameter
>>> p = ToolParameter.build( None, XML( '''
... <param name="blah" type="text" size="10" value="10">
...     <validator type="regex" message="Not gonna happen">[Ff]oo</validator>
... </param>
... ''' ) )
>>> t = p.validate( "Foo" )
>>> t = p.validate( "foo" )
>>> t = p.validate( "Fop" )
Traceback (most recent call last):
    ...
ValueError: Not gonna happen
```

classmethod from_element(*param*, *elem*)

validate(*value*, *history=None*)

class galaxy.tools.parameters.validation.**UnspecifiedBuildValidator**(*message=None*)
Bases: *galaxy.tools.parameters.validation.Validator*

Validator that checks for dbkey not equal to '?'

classmethod from_element(*param*, *elem*)

validate(*value*, *history=None*)

class galaxy.tools.parameters.validation.**Validator**
Bases: object

A validator checks that a value meets some conditions OR raises ValueError

classmethod from_element(*param*, *elem*)

validate(*value*, *history=None*)

galaxy.tools.parameters.validation.**get_suite**()
Get unittest suite for this module

search Package

search Package Module for building and searching the index of tools installed within this Galaxy.

class galaxy.tools.search.**ToolBoxSearch**(*toolbox*, *index_help=True*)
Bases: object

Support searching tools in a toolbox. This implementation uses the Whoosh search library.

build_index(*index_help=True*)

search(*q*, *tool_name_boost*, *tool_section_boost*, *tool_description_boost*, *tool_help_boost*, *tool_search_limit*)
Perform search on the in-memory index. Weight in the given boosts.

util Package

util Package Utilities used by various Galaxy tools

FIXME: These are used by tool scripts, not the framework, and should not live in this package.

maf_utilities Module Provides wrappers and utilities for working with MAF files and alignments.

class galaxy.tools.util.maf_utilities.**GenomicRegionAlignment**(*start*, *end*, *species=[]*, *temp_file_handler=None*)

Bases: *galaxy.tools.util.maf_utilities.RegionAlignment*

class galaxy.tools.util.maf_utilities.**RegionAlignment**(*size*, *species=[]*, *temp_file_handler=None*)

Bases: object

DNA_COMPLEMENT = '\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x

MAX_SEQUENCE_SIZE = 9223372036854775807

add_species(*species*)

flush(*species=None*)

get_sequence(*species*)

get_sequence_reverse_complement(*species*)

get_species_names(*skip=[]*)

set_position(*index*, *species*, *base*)

set_range(*index*, *species*, *bases*)

class galaxy.tools.util.maf_utilities.**SplicedAlignment**(*exon_starts*, *exon_ends*, *species=[]*, *temp_file_handler=None*)

Bases: object

DNA_COMPLEMENT = '\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x

end

get_sequence(*species*)

get_sequence_reverse_complement(*species*)

get_species_names(*skip=[]*)

start

class galaxy.tools.util.maf_utilities.**TempFileHandler**(*max_open_files=None*, ***kwds*)

Bases: object

Handles creating, opening, closing, and deleting of Temp files, with a maximum number of files open at one time.

DEFAULT_MAX_OPEN_FILES = 512

close(*index*, *delete=False*)

flush(*index*)

get_open_tempfile(*index=None*, ***kwds*)

galaxy.tools.util.maf_utilities.**build_maf_index**(*maf_file*, *species=None*)

galaxy.tools.util.maf_utilities.**build_maf_index_species_chromosomes**(*filename*, *index_species=None*)

galaxy.tools.util.maf_utilities.**chop_block_by_region**(*block*, *src*, *region*, *species=None*, *mincols=0*)

galaxy.tools.util.maf_utilities.**component_overlaps_region**(*c*, *region*)

galaxy.tools.util.maf_utilities.**fill_region_alignment**(*alignment*, *index*, *primary_species*, *chrom*, *start*, *end*, *strand='+'*, *species=None*, *mincols=0*, *overwrite_with_gaps=True*)

galaxy.tools.util.maf_utilities.**get_attributes_from_fasta_header**(*header*)

galaxy.tools.util.maf_utilities.**get_chopped_blocks_for_region**(*index*, *src*, *region*, *species=None*, *mincols=0*)

galaxy.tools.util.maf_utilities.**get_chopped_blocks_with_index_offset_for_region**(*index*, *src*, *region*, *species=None*, *mincols=0*)

galaxy.tools.util.maf_utilities.**get_components_by_src**(*block*, *src*)

galaxy.tools.util.maf_utilities.**get_components_by_src_start**(*block*, *src*)

galaxy.tools.util.maf_utilities.**get_fasta_header**(*component*, *attributes={}*, *suffix=None*)

galaxy.tools.util.maf_utilities.**get_oriented_chopped_blocks_for_region**(*index*, *src*, *region*, *species=None*, *mincols=0*, *force_strand=None*)

galaxy.tools.util.maf_utilities.**get_oriented_chopped_blocks_with_index_offset_for_region**(*in*
sr
re
gi
sp
m
co
fo

galaxy.tools.util.maf_utilities.**get_region_alignment**(*index*, *primary_species*, *chrom*, *start*, *end*, *strand='+'*, *species=None*, *mincols=0*, *overwrite_with_gaps=True*, *temp_file_handler=None*)

galaxy.tools.util.maf_utilities.**get_species_in_block**(*block*)

galaxy.tools.util.maf_utilities.**get_species_in_maf**(*maf_filename*)

galaxy.tools.util.maf_utilities.**get_spliced_region_alignment**(*index, primary_species, chrom, starts, ends, strand='+', species=None, mincols=0, overwrite_with_gaps=True, temp_file_handler=None*)

galaxy.tools.util.maf_utilities.**get_starts_ends_fields_from_gene_bed**(*line*)

galaxy.tools.util.maf_utilities.**iter_blocks_split_by_species**(*block, species=None*)

galaxy.tools.util.maf_utilities.**iter_blocks_split_by_src**(*block, src*)

galaxy.tools.util.maf_utilities.**iter_components_by_src**(*block, src*)

galaxy.tools.util.maf_utilities.**iter_components_by_src_start**(*block, src*)

galaxy.tools.util.maf_utilities.**iter_fasta_alignment**(*filename*)

galaxy.tools.util.maf_utilities.**line_enumerator**(*lines, comment_start='#'*)

galaxy.tools.util.maf_utilities.**maf_index_by_uid**(*maf_uid, index_location_file*)

galaxy.tools.util.maf_utilities.**open_or_build_maf_index**(*maf_file, index_filename, species=None*)

galaxy.tools.util.maf_utilities.**orient_block_by_region**(*block, src, region, force_strand=None*)

galaxy.tools.util.maf_utilities.**parse_species_option**(*species*)

galaxy.tools.util.maf_utilities.**reduce_block_by_primary_genome**(*block, species, chromosome, region_start*)

galaxy.tools.util.maf_utilities.**remove_temp_index_file**(*index_filename*)

galaxy.tools.util.maf_utilities.**sort_block_components_by_block**(*block1, block2*)

galaxy.tools.util.maf_utilities.**src_merge**(*spec, chrom, contig=None*)

galaxy.tools.util.maf_utilities.**src_split**(*src*)

galaxy.tools.util.maf_utilities.**tool_fail**(*msg='Unknown Error'*)

Subpackages

galaxyops Package

galaxyops Package Utility functions for galaxyops

galaxy.tools.util.galaxyops.**default_printer**(*stream, exc, obj*)

galaxy.tools.util.galaxyops.**fail**(*msg*)

galaxy.tools.util.galaxyops.**parse_cols_arg**(*cols*)
Parse a columns command line argument into a four-tuple

galaxy.tools.util.galaxyops.**skipped**(*reader, filedesc=''*)

galaxy.tools.util.galaxyops.**warn**(*msg*)

util Package

util Package Utility functions used systemwide.

class galaxy.util.**ExecutionTimer**
Bases: object

class galaxy.util.**Params**(*params*, *sanitize=True*)
Bases: object

Stores and 'sanitizes' parameters. Alphanumeric characters and the non-alphanumeric ones that are deemed safe are let to pass through (see L{valid_chars}). Some non-safe characters are escaped to safe forms for example C{>} becomes C{__lt__} (see L{mapped_chars}). All other characters are replaced with C{X}.

Operates on string or list values only (HTTP parameters).

```
>>> values = { 'status':'on', 'symbols':[  'alpha', '<>', '$rm&#!' ]  }
>>> par = Params(values)
>>> par.status
'on'
>>> par.value == None      # missing attributes return None
True
>>> par.get('price', 0)
0
>>> par.symbols            # replaces unknown symbols with X
['alpha', '__lt____gt__', 'XrmX__pd__!']
>>> sorted(par.flatten())  # flattening to a list
[('status', 'on'), ('symbols', 'XrmX__pd__!'), ('symbols', '__lt____gt__'), ('symbols', 'alpha')
```

NEVER_SANITIZE = ['file_data', 'url_paste', 'URL', 'filesystem_paths']

flatten()
Creates a tuple list from a dict with a tuple/value pair for every value that is a list

get(*key*, *default*)

update(*values*)

class galaxy.util.**ParamsWithSpecs**(*specs=None*, *params=None*)
Bases: collections.defaultdict

galaxy.util.**asbool**(*obj*)

galaxy.util.**commaify**(*amount*)

galaxy.util.**compare_urls**(*url1*, *url2*, *compare_scheme=True*, *compare_hostname=True*, *compare_path=True*)

galaxy.util.**docstring_trim**(*docstring*)
Trimming python doc strings. Taken from: http://www.python.org/dev/peps/pep-0257/

galaxy.util.**file_iter**(*fname*, *sep=None*)
This generator iterates over a file and yields its lines splitted via the C{sep} parameter. Skips empty lines and lines starting with the C{#} character.

```
>>> lines = [ line for line in file_iter(__file__) ]
>>> len(lines) !=  0
True
```

galaxy.util.**file_reader**(*fp*, *chunk_size=65536*)
This generator yields the open fileobject in chunks (default 64k). Closes the file at the end

galaxy.util.**force_symlink**(*source*, *link_name*)

galaxy.util.**galaxy_directory**()

galaxy.util.**get_charset_from_http_headers**(*headers*, *default=None*)

galaxy.util.**get_file_size**(*value*, *default=None*)

galaxy.util.**in_directory**(*file*, *directory*, *local_path_module=<module 'posixpath' from '/home/docs/checkouts/readthedocs.org/user_builds/jmchilton-galaxy/envs/latest/lib/python2.7/posixpath.pyc'>*)

Return true, if the common prefix of both is equal to directory e.g. /a/b/c/d.rst and directory is /a/b, the common prefix is /a/b

galaxy.util.**is_binary**(*value*, *binary_chars=None*)

File is binary if it contains a null-byte by default (e.g. behavior of grep, etc.). This may fail for utf-16 files, but so would ASCII encoding. >>> is_binary(string.printable) False >>> is_binary('xcex94') False >>> is_binary('000') True

galaxy.util.**is_multi_byte**(*chars*)

galaxy.util.**is_uuid**(*value*)

This method returns True if value is a UUID, otherwise False. >>> is_uuid("123e4567-e89b-12d3-a456-426655440000") True >>> is_uuid("0x3242340298902834") False

galaxy.util.**listify**(*item*, *do_strip=False*)

Make a single item a single item list, or return a list if passed a list. Passing a None returns an empty list.

galaxy.util.**mask_password_from_url**(*url*)

Masks out passwords from connection urls like the database connection in galaxy.ini

```
>>> mask_password_from_url( 'sqlite+postgresql://user:password@localhost/' )
'sqlite+postgresql://user:********@localhost/'
>>> mask_password_from_url( 'amqp://user:amqp@localhost' )
'amqp://user:********@localhost'
>>> mask_password_from_url( 'amqp://localhost')
'amqp://localhost'
```

galaxy.util.**merge_sorted_iterables**(*operator*, **iterables*)

```
>>> operator = lambda x: x
>>> list( merge_sorted_iterables( operator, [1,2,3], [4,5] ) )
[1, 2, 3, 4, 5]
>>> list( merge_sorted_iterables( operator, [4, 5], [1,2,3] ) )
[1, 2, 3, 4, 5]
>>> list( merge_sorted_iterables( operator, [1, 4, 5], [2], [3] ) )
[1, 2, 3, 4, 5]
```

galaxy.util.**mkstemp_ln**(*src*, *prefix='mkstemp_ln_'*)

From tempfile._mkstemp_inner, generate a hard link in the same dir with a random name. Created so we can persist the underlying file of a NamedTemporaryFile upon its closure.

galaxy.util.**move_merge**(*source*, *target*)

galaxy.util.**nice_size**(*size*)

Returns a readably formatted string with the size

```
>>> nice_size(100)
'100 bytes'
>>> nice_size(10000)
'9.8 KB'
>>> nice_size(1000000)
'976.6 KB'
>>> nice_size(100000000)
'95.4 MB'
```

galaxy.util.**object_to_string**(*obj*)

galaxy.util.**parse_xml**(*fname*)
Returns a parsed xml tree

galaxy.util.**parse_xml_string**(*xml_string*)

galaxy.util.**pretty_print_json**(*json_data*, *is_json_string=False*)

galaxy.util.**pretty_print_time_interval**(*time=False*, *precise=False*)
Get a datetime object or a int() Epoch timestamp and return a pretty string like 'an hour ago', 'Yesterday', '3 months ago', 'just now', etc credit: http://stackoverflow.com/questions/1551382/user-friendly-time-format-in-python

galaxy.util.**pretty_print_xml**(*elem*, *level=0*)

galaxy.util.**read_build_sites**(*filename*, *check_builds=True*)
read db names to ucsc mappings from file, this file should probably be merged with the one above

galaxy.util.**read_dbnames**(*filename*)
Read build names from file

galaxy.util.**ready_name_for_url**(*raw_name*)
General method to convert a string (i.e. object name) to a URL-ready slug.

```
>>> ready_name_for_url( "My Cool Object" )
'My-Cool-Object'
>>> ready_name_for_url( "!My Cool Object!" )
'My-Cool-Object'
>>> ready_name_for_url( "Hello₩" )
'Hello'
```

galaxy.util.**recursively_stringify_dictionary_keys**(*d*)

galaxy.util.**relativize_symlinks**(*path*, *start=None*, *followlinks=False*)

galaxy.util.**restore_text**(*text*, *character_map={'@': '__at__', '\t': '__tc__', '\n': '__cn__', '\r': '__cr__', '[': '__ob__', ']': '__cb__', '#': '__pd__', '"': '__dq__', '"': '__sq__', '{': '__oc__', '}': '__cc__', '<': '__lt__', '>': '__gt__'}*)
Restores sanitized text

galaxy.util.**roundify**(*amount*, *sfs=2*)
Take a number in string form and truncate to 'sfs' significant figures.

galaxy.util.**rst_to_html**(*s*)
Convert a blob of reStructuredText to HTML

galaxy.util.**safe_str_cmp**(*a*, *b*)
safely compare two strings in a timing-attack-resistant manner

galaxy.util.**sanitize_for_filename**(*text*, *default=None*)
Restricts the characters that are allowed in a filename portion; Returns default value or a unique id string if result

is not a valid name. Method is overly aggressive to minimize possible complications, but a maximum length is not considered.

galaxy.util.**sanitize_lists_to_string**(*values*, *valid_characters=set(['!', ' ', ')', '(', '+', '*', '-', ',', '.', '/', ':', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8', ':', '=', '?', 'A', 'C', 'B', 'E', 'D', 'G', 'F', 'I', 'H', 'K', 'J', 'M', 'L', 'O', 'N', 'Q', 'P', 'S', 'R', 'U', 'T', 'W', 'V', 'Y', 'X', 'Z', '_', '^', 'a', 'c', 'b', 'e', 'd', 'g', 'f', 'i', 'h', 'k', 'j', 'm', 'l', 'o', 'n', 'q', 'p', 's', 'r', 'u', 't', 'w', 'v', 'y', 'x', 'z'])*, *character_map={'@': '__at__', '\t': '__tc__', '\n': '__cn__', '\r': '__cr__', '[': '__ob__', ']': '__cb__', '#': '__pd__', '"': '__dq__', '''': '__sq__', '{': '__oc__', '}': '__cc__', '<': '__lt__', '>': '__gt__'}*, *invalid_character='X'*)

galaxy.util.**sanitize_param**(*value*, *valid_characters=set(['!', ' ', ')', '(', '+', '*', '-', ',', '.', '/', ':', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8', ':', '=', '?', 'A', 'C', 'B', 'E', 'D', 'G', 'F', 'I', 'H', 'K', 'J', 'M', 'L', 'O', 'N', 'Q', 'P', 'S', 'R', 'U', 'T', 'W', 'V', 'Y', 'X', 'Z', '_', '^', 'a', 'c', 'b', 'e', 'd', 'g', 'f', 'i', 'h', 'k', 'j', 'm', 'l', 'o', 'n', 'q', 'p', 's', 'r', 'u', 't', 'w', 'v', 'y', 'x', 'z'])*, *character_map={'@': '__at__', '\t': '__tc__', '\n': '__cn__', '\r': '__cr__', '[': '__ob__', ']': '__cb__', '#': '__pd__', '"': '__dq__', '''': '__sq__', '{': '__oc__', '}': '__cc__', '<': '__lt__', '>': '__gt__'}*, *invalid_character='X'*)

Clean incoming parameters (strings or lists)

galaxy.util.**sanitize_text**(*text*, *valid_characters=set(['!', ' ', ')', '(', '+', '*', '-', ',', '.', '/', ':', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8', ':', '=', '?', 'A', 'C', 'B', 'E', 'D', 'G', 'F', 'I', 'H', 'K', 'J', 'M', 'L', 'O', 'N', 'Q', 'P', 'S', 'R', 'U', 'T', 'W', 'V', 'Y', 'X', 'Z', '_', '^', 'a', 'c', 'b', 'e', 'd', 'g', 'f', 'i', 'h', 'k', 'j', 'm', 'l', 'o', 'n', 'q', 'p', 's', 'r', 'u', 't', 'w', 'v', 'y', 'x', 'z'])*, *character_map={'@': '__at__', '\t': '__tc__', '\n': '__cn__', '\r': '__cr__', '[': '__ob__', ']': '__cb__', '#': '__pd__', '"': '__dq__', '''': '__sq__', '{': '__oc__', '}': '__cc__', '<': '__lt__', '>': '__gt__'}*, *invalid_character='X'*)

Restricts the characters that are allowed in text; accepts both strings and lists of strings; non-string entities will be cast to strings.

galaxy.util.**send_mail**(*frm*, *to*, *subject*, *body*, *config*)

Sends an email.

galaxy.util.**shrink_stream_by_size**(*value*, *size*, *join_by='..'*, *left_larger=True*, *beginning_on_size_error=False*, *end_on_size_error=False*)

galaxy.util.**shrink_string_by_size**(*value*, *size*, *join_by='..'*, *left_larger=True*, *beginning_on_size_error=False*, *end_on_size_error=False*)

galaxy.util.**size_to_bytes**(*size*)

Returns a number of bytes if given a reasonably formatted string with the size

galaxy.util.**smart_str**(*s*, *encoding='utf-8'*, *strings_only=False*, *errors='strict'*)

Returns a bytestring version of 's', encoded as specified in 'encoding'.

If strings_only is True, don't convert (some) non-string-like objects.

Adapted from an older, simpler version of django.utils.encoding.smart_str.

galaxy.util.**string_as_bool**(*string*)

galaxy.util.**string_as_bool_or_none**(*string*)

Returns True, None or False based on the argument: True if passed True, 'True', 'Yes', or 'On' None if passed None or 'None' False otherwise

Note: string comparison is case-insensitive so lowecase versions of those function equivalently.

galaxy.util.**string_to_object**(*s*)

galaxy.util.**stringify_dictionary_keys**(*in_dict*)

galaxy.util.**synchronized**(*func*)

This wrapper will serialize access to 'func' to a single thread. Use it as a decorator.

galaxy.util.**umask_fix_perms**(*path*, *umask*, *unmasked_perms*, *gid=None*)

umask-friendly permissions fixing

galaxy.util.**unicodify**(*value*, *encoding='utf-8'*, *error='replace'*, *default=None*)

Returns a unicode string or None

galaxy.util.**unique_id**(*KEY_SIZE=128*)

Generates an unique id

```
>>> ids = [ unique_id() for i in range(1000) ]
>>> len(set(ids))
1000
```

galaxy.util.**xml_element_compare**(*elem1*, *elem2*)

galaxy.util.**xml_element_list_compare**(*elem_list1*, *elem_list2*)

galaxy.util.**xml_element_to_dict**(*elem*)

galaxy.util.**xml_text**(*root*, *name=None*)

Returns the text inside an element

galaxy.util.**xml_to_string**(*elem*, *pretty=False*)

Returns a string from an xml tree

aliaspickler Module

class galaxy.util.aliaspickler.**AliasPickleModule**(*aliases*)

Bases: object

dump(*obj*, *fileobj*, *protocol=0*)

dumps(*obj*, *protocol=0*)

load(*fileobj*)

loads(*string*)

class galaxy.util.aliaspickler.**AliasUnpickler**(*aliases*, **args*, ***kw*)

Bases: pickle.Unpickler

find_class(*module*, *name*)

bunch Module

class galaxy.util.bunch.**Bunch**(***kwds*)

Bases: object

http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/52308

Often we want to just collect a bunch of stuff together, naming each item of the bunch; a dictionary's OK for that, but a small do-nothing class is even handier, and prettier to use.

get(*key*, *default=None*)

items()

keys()

values()

debugging Module

class galaxy.util.debugging.**SimpleProfiler**(*log=None*)

Bases: object

Simple profiler that captures the duration between calls to *report* and stores the results in a list.

REPORT_FORMAT = '%20f: %s'

get_reports()

report(*msg*)

start(*msg=None*)

galaxy.util.debugging.**stack_trace_string**(*max_depth=None*, *line_format='{index}:{file}:{function}:{line}'*)

Returns a string representation of the current stack.

Parameters **depth** – positive integer to control how many levels of the stack are returned. max_depth=None returns the entire stack (default).

expressions Module Expression evaluation support.

For the moment this depends on python's eval. In the future it should be replaced with a "safe" parser.

class galaxy.util.expressions.**ExpressionContext**(*dict*, *parent=None*)

Bases: object, UserDict.DictMixin

hash_util Module Utility functions for bi-directional Python version compatibility. Python 2.5 introduced hashlib which replaced sha in Python 2.4 and previous versions.

galaxy.util.hash_util.**hmac_new**(*key*, *value*)

galaxy.util.hash_util.**is_hashable**(*value*)

galaxy.util.hash_util.**new_secure_hash**(*text_type=None*)

Returns either a sha1 hash object (if called with no arguments), or a hexdigest of the sha1 hash of the argument *text_type*.

heartbeat Module

class galaxy.util.heartbeat.**Heartbeat**(*name='Heartbeat Thread'*, *period=20*, *fname='heartbeat.log'*)

Bases: threading.Thread

Thread that periodically dumps the state of all threads to a file

get_interesting_stack_frame(*stack_frames*)

Scans a given backtrace stack frames, returns a single quadraple of [filename, line, function-name, text] of the single, deepest, most interesting frame.

Interesting being:

```
inside the galaxy source code ("/lib/galaxy"),
prefreably not an egg.
```

print_nonsleeping(*threads_object_dict*)

run()

shutdown()

thread_is_sleeping(*last_stack_frame*)
Returns True if the given stack-frame represents a known sleeper function (at least in python 2.5)

galaxy.util.heartbeat.**get_current_thread_object_dict**()
Get a dictionary of all 'Thread' objects created via the threading module keyed by thread_id. Note that not all interpreter threads have a thread objects, only the main thread and any created via the 'threading' module. Threads created via the low level 'thread' module will not be in the returned dictionary.

HACK: This mucks with the internals of the threading module since that module does not expose any way to match 'Thread' objects with intepreter thread identifiers (though it should).

inflection Module

class galaxy.util.inflection.**Base**
Locale inflectors must inherit from this base class inorder to provide the basic Inflector functionality

camelize(*word*)
Returns given word as CamelCased Converts a word like "send_email" to "SendEmail". It will remove non alphanumeric character from the word, so "who's online" will be converted to "WhoSOnline"

classify(*table_name*)
Converts a table name to its class name according to rails naming conventions. Example: Converts "people" to "Person"

cond_plural(*number_of_records*, *word*)
Returns the plural form of a word if first parameter is greater than 1

demodulize(*module_name*)

foreignKey(*class_name*, *separate_class_name_and_id_with_underscore=1*)
Returns class_name in underscored form, with "_id" tacked on at the end. This is for use in dealing with the database.

humanize(*word*, *uppercase=''*)
Returns a human-readable string from word Returns a human-readable string from word, by replacing underscores with a space, and by upper-casing the initial character by default. If you need to uppercase all the words you just have to pass 'all' as a second parameter.

modulize(*module_description*)

ordinalize(*number*)
Converts number to its ordinal English form. This method converts 13 to 13th, 2 to 2nd ...

string_replace(*word*, *find*, *replace*)
This function returns a copy of word, translating all occurrences of each character in find to the corresponding character in replace

tableize(*class_name*)
Converts a class name to its table name according to rails naming conventions. Example. Converts "Person" to "people"

titleize(*word*, *uppercase=''*)
Converts an underscored or CamelCase word into a English sentence. The titleize function converts text like "WelcomePage", "welcome_page" or "welcome page" to this "Welcome Page". If second parameter is set to 'first' it will only capitalize the first character of the title.

unaccent(*text*)
Transforms a string to its unaccented version. This might be useful for generating "friendly" URLs

underscore(*word*)
Converts a word "into_it_s_underscored_version" Convert any "CamelCased" or "ordinary Word" into an "underscored_word". This can be really useful for creating friendly URLs.

urlize(*text*)
Transform a string its unaccented and underscored version ready to be inserted in friendly URLs

variablize(*word*)
Same as camelize but first char is lowercased Converts a word like "send_email" to "sendEmail". It will remove non alphanumeric character from the word, so "who's online" will be converted to "whoSOnline"

class galaxy.util.inflection.**English**
Bases: *galaxy.util.inflection.Base*

Inflector for pluralize and singularize English nouns.

This is the default Inflector for the Inflector obj

pluralize(*word*)
Pluralizes English nouns.

singularize(*word*)
Singularizes English nouns.

class galaxy.util.inflection.**Inflector**(*Inflector=<class galaxy.util.inflection.English>*)
Inflector for pluralizing and singularizing nouns.

It provides methods for helping on creating programs based on naming conventions like on Ruby on Rails.

camelize(*word*)
Returns given word as CamelCased Converts a word like "send_email" to "SendEmail". It will remove non alphanumeric character from the word, so "who's online" will be converted to "WhoSOnline"

classify(*table_name*)
Converts a table name to its class name according to rails naming conventions. Example: Converts "people" to "Person"

cond_plural(*number_of_records*, *word*)
Returns the plural form of a word if first parameter is greater than 1

demodulize(*module_name*)

foreignKey(*class_name*, *separate_class_name_and_id_with_underscore=1*)
Returns class_name in underscored form, with "_id" tacked on at the end. This is for use in dealing with the database.

humanize(*word*, *uppercase='*')
Returns a human-readable string from word Returns a human-readable string from word, by replacing underscores with a space, and by upper-casing the initial character by default. If you need to uppercase all the words you just have to pass 'all' as a second parameter.

modulize(*module_description*)

ordinalize(*number*)
Converts number to its ordinal form. This method converts 13 to 13th, 2 to 2nd ...

pluralize(*word*)
Pluralizes nouns.

singularize(*word*)
Singularizes nouns.

tableize(*class_name*)
Converts a class name to its table name according to rails naming conventions. Example. Converts "Person" to "people"

titleize(*word*, *uppercase=''*)
Converts an underscored or CamelCase word into a sentence. The titleize function converts text like "WelcomePage", "welcome_page" or "welcome page" to this "Welcome Page". If the "uppercase" parameter is set to 'first' it will only capitalize the first character of the title.

unaccent(*text*)
Transforms a string to its unaccented version. This might be useful for generating "friendly" URLs

underscore(*word*)
Converts a word "into_it_s_underscored_version" Convert any "CamelCased" or "ordinary Word" into an "underscored_word". This can be really useful for creating friendly URLs.

urlize(*text*)
Transform a string to its unaccented and underscored version ready to be inserted in friendly URLs

variablize(*word*)
Same as camelize but first char is lowercased Converts a word like "send_email" to "sendEmail". It will remove non alphanumeric character from the word, so "who's online" will be converted to "whoSOnline"

`json` Module

galaxy.util.json.**dumps**(*obj*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is false then `dict` keys that are not basic types (`str`, `unicode`, `int`, `long`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, all non-ASCII characters are not escaped, and the return value may be a `unicode` instance. See `dump` for details.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `OverflowError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

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

If `separators` is an `(item_separator, dict_separator)` tuple then it will be used instead of the default `(', ', ': ')` separators. `(',', ':')` is the most compact JSON representation.

`encoding` is the character encoding for str instances, default is UTF-8.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is `True` (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

galaxy.util.json.**loads**(*s*, *encoding=None*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str` or `unicode` instance containing a JSON document) to a Python object.

If `s` is a `str` instance and is encoded with an ASCII based encoding other than utf-8 (e.g. latin-1) then an appropriate `encoding` name must be specified. Encodings that are not ASCII based (such as UCS-2) are not allowed and should be decoded to `unicode` first.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders that rely on the order that the key and value pairs are decoded (for example, collections.OrderedDict will remember the order of insertion). If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN, null, true, false. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

galaxy.util.json.**safe_dumps**(**args*, ***kwargs*)

This is a wrapper around dumps that encodes Infinity and NaN values. It's a fairly rare case (which will be low in request volume). Basically, we tell json.dumps to blow up if it encounters Infinity/NaN, and we 'fix' it before re-encoding.

galaxy.util.json.**json_fix**(*val*)

galaxy.util.json.**validate_jsonrpc_request**(*request*, *regular_methods*, *notification_methods*)

galaxy.util.json.**validate_jsonrpc_response**(*response*, *id=None*)

galaxy.util.json.**jsonrpc_request**(*method*, *params=None*, *id=None*, *jsonrpc='2.0'*)

galaxy.util.json.**jsonrpc_response**(*request=None*, *id=None*, *result=None*, *error=None*, *jsonrpc='2.0'*)

`lrucache` Module

Kanwei Li, 03/2010

Simple LRU cache that uses a dictionary to store a specified number of objects at a time.

class galaxy.util.lrucache.**LRUCache**(*num_elements*)

clear()

Clears/initiates storage variables

`memdump` Module

none_like Module Objects with No values

class galaxy.util.none_like.**NoneDataset**(*datatypes_registry=None*, *ext='data'*, *dbkey='?'*)
Bases: galaxy.util.none_like.RecursiveNone

missing_meta()

class galaxy.util.none_like.**RecursiveNone**

odict Module Ordered dictionary implementation.

class galaxy.util.odict.**odict**(*dict=None*)
Bases: UserDict.UserDict

http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/107747

This dictionary class extends UserDict to record the order in which items are added. Calling keys(), values(), items(), etc. will return results in this order.

clear()

copy()

insert(*index*, *key*, *item*)

items()

iteritems()

iterkeys()

itervalues()

keys()

popitem()

reverse()

setdefault(*key*, *failobj=None*)

update(*dict*)

values()

sanitize_html Module HTML Sanitizer (ripped from feedparser)

galaxy.util.sanitize_html.**sanitize_html**(*htmlSource*, *encoding='utf-8'*, *type='text/html'*)

shed_util Module

shed_util_common Module

streamball Module A simple wrapper for writing tarballs as a stream.

class galaxy.util.streamball.**StreamBall**(*mode*, *members=None*)
Bases: object

add(*file*, *relpath*, *check_file=False*)

stream(*environ*, *start_response*)

class galaxy.util.streamball.**ZipBall**(*tmpf*, *tmpd*)
Bases: object

stream(*environ*, *start_response*)

template Module
galaxy.util.template.**fill_template**(*template_text*, *context=None*, ***kwargs*)

topsort Module Topological sort.

From Tim Peters, see: http://mail.python.org/pipermail/python-list/1999-July/006660.html

topsort takes a list of pairs, where each pair (x, y) is taken to mean that x <= y wrt some abstract partial ordering. The return value is a list, representing a total ordering that respects all the input constraints. E.g.,

topsort([(1,2), (3,3)])

Valid topological sorts would be any of (but nothing other than)

[3, 1, 2] [1, 3, 2] [1, 2, 3]

... however this variant ensures that 'key' order (first element of tuple) is preserved so the following will be result returned:

[1, 3, 2]

because those are the permutations of the input elements that respect the "1 precedes 2" and "3 precedes 3" input constraints. Note that a constraint of the form (x, x) is really just a trick to make sure x appears *somewhere* in the output list.

If there's a cycle in the constraints, say

topsort([(1,2), (2,1)])

then CycleError is raised, and the exception object supports many methods to help analyze and break the cycles. This requires a good deal more code than topsort itself!

exception galaxy.util.topsort.**CycleError**(*sofar*, *numpreds*, *succs*)
Bases: exceptions.Exception

get_elements()

get_pairlist()

get_partial()

get_pred_counts()

get_preds()

get_succs()

pick_a_cycle()

galaxy.util.topsort.**topsort**(*pairlist*)

galaxy.util.topsort.**topsort_levels**(*pairlist*)

Subpackages

backports Package

backports Package Modules for providing backward compatibility with future versions of Python

Subpackages

importlib Package

importlib Package Backport of importlib.import_module from 3.x.

galaxy.util.backports.importlib.**import_module**(*name*, *package=None*)
Import a module.

The 'package' argument is required when performing a relative import. It specifies the package to use as the anchor point from which to resolve the relative import to an absolute import.

visualization Package

visualization Package Package for Galaxy visualization plugins.

genomes Module

class galaxy.visualization.genomes.**Genome**(*key*, *description*, *len_file=None*, *twobit_file=None*)
Bases: object

Encapsulates information about a known genome/dbkey.

to_dict(*num=None*, *chrom=None*, *low=None*)
Returns representation of self as a dictionary.

class galaxy.visualization.genomes.**GenomeRegion**(*chrom=None*, *start=0*, *end=0*, *sequence=None*)
Bases: object

A genomic region on an individual chromosome.

static from_dict(*obj_dict*)

static from_str(*obj_str*)

class galaxy.visualization.genomes.**Genomes**(*app*)
Bases: object

Provides information about available genome data and methods for manipulating that data.

check_and_reload()

chroms(*trans*, *dbkey=None*, *num=None*, *chrom=None*, *low=None*)
Returns a naturally sorted list of chroms/contigs for a given dbkey. Use either chrom or low to specify the starting chrom in the return list.

get_build(*dbkey*)
Returns build for the given key.

get_dbkeys(*trans*, *chrom_info=False*, ***kwd*)
Returns all known dbkeys. If chrom_info is True, only dbkeys with chromosome lengths are returned.

has_reference_data(*dbkey*, *dbkey_owner=None*)
Returns true if there is reference data for the specified dbkey. If dbkey is custom, dbkey_owner is needed to determine if there is reference data.

reference(*trans*, *dbkey*, *chrom*, *low*, *high*)
Return reference data for a build.

reload_genomes()

galaxy.visualization.genomes.**decode_dbkey**(*dbkey*)
Decodes dbkey and returns tuple (username, dbkey)

Subpackages

data_providers Package

data_providers Package Galaxy visualization/visual analysis data providers.

basic Module

class galaxy.visualization.data_providers.basic.**BaseDataProvider**(*converted_dataset=None*, *original_dataset=None*, *dependencies=None*, *error_max_vals='Only the first %i values are returned.'*)

Bases: object

Base class for data providers. Data providers (a) read and package data from datasets; and (b) write subsets of data to new datasets.

get_data(*chrom*, *start*, *end*, *start_val=0*, *max_vals=9223372036854775807*, ***kwargs*)
Returns data as specified by kwargs. start_val is the first element to return and max_vals indicates the number of values to return.

Return value must be a dictionary with the following attributes: dataset_type, data

get_iterator(***kwargs*)
Returns an iterator that provides data in the region chrom:start-end

has_data(***kwargs*)
Returns true if dataset has data in the specified genome window, false otherwise.

process_data(*iterator*, *start_val=0*, *max_vals=None*, ***kwargs*)
Process data from an iterator to a format that can be provided to client.

write_data_to_file(*filename*, ***kwargs*)
Write data in region defined by chrom, start, and end to a file.

class galaxy.visualization.data_providers.basic.**ColumnDataProvider**(*original_dataset*, *max_lines_returned=30000*)

Bases: *galaxy.visualization.data_providers.basic.BaseDataProvider*

Data provider for columnar data

MAX_LINES_RETURNED = 30000

get_data(*columns=None*, *start_val=0*, *max_vals=None*, *skip_comments=True*, ***kwargs*)
Returns data from specified columns in dataset. Format is list of lists where each list is a line of data.

genome Module

registry Module

Subpackages

phyloviz Package

phyloviz Package Data providers code for PhyloViz

class galaxy.visualization.data_providers.phyloviz.**PhylovizDataProvider**(*original_dataset=None*)

Bases: galaxy.visualization.data_providers.basic.BaseDataProvider

dataset_type = 'phylo'

get_data(*tree_index=0*)

Returns trees. Trees are actually an array of JsonDicts. It's usually one tree, except in the case of Nexus

baseparser Module

class galaxy.visualization.data_providers.phyloviz.baseparser.**Base_Parser**

Bases: object

Base parsers contain all the methods to handle phylogeny tree creation and converting the data to json that all parsers should have

parseFile(*filePath*)

Base method that all phylogeny file parser should have

toJson(*jsonDict*)

Convenience method to get a json string from a python json dict

class galaxy.visualization.data_providers.phyloviz.baseparser.**Node**(*nodeName*, ***kwargs*)

Bases: object

Node class of PhyloTree, which represents a CLAUDE in a phylogenetic tree

addChildNode(*child*)

Adds a child node to the current node

addChildrenToJson(*jsonDict*)

Needs a special method to addChildren, such that the key does not appear in the Jsondict when the children is empty this requirement is due to the layout algorithm used by d3 layout for hiding subtree

addMiscToJson(*jsonDict*)

Adds other misc attributes to json if they are present

toJson()

Converts the data in the node to a dict representation of json

class galaxy.visualization.data_providers.phyloviz.baseparser.**PhyloTree**

Bases: object

Standardized python based class to represent the phylogenetic tree parsed from different phylogenetic file formats.

addAttributesToRoot(*attrDict*)

Adds attributes to root, but first we put it in a temp store and bind it with root when .toJson is called

addRoot (*root*)
Creates a root for phyloTree

generateJsonableDict ()
Changes itself into a dictonary by recurssively calling the tojson on all its nodes. Think of it as a dict in an array of dict in an array of dict and so on...

makeNode (*nodeName*, ***kwargs*)
Called to make a node within PhyloTree, arbitrary kwargs can be passed to annotate nodes Tracks the number of nodes via internally incremented id

newickparser Module

class galaxy.visualization.data_providers.phyloviz.newickparser.**Newick_Parser**
Bases: *galaxy.visualization.data_providers.phyloviz.baseparser.Base_Parser*

For parsing trees stored in the newick format (.nhx) It is necessarily more complex because this parser is later extended by Nexus for parsing newick as well..

cleanNewickString (*rawNewick*)
removing semi colon, and illegal json characters (,',") and white spaces

parseData (*newickString*)
To be called on a newickString directly to parse it. Returns: jsonableDict

parseFile (*filePath*)
Parses a newick file to obtain the string inside. Returns: jsonableDict

parseNode (*string*, *depth*)
Recursive method for parsing newick string, works by stripping down the string into substring of newick contained with brackers, which is used to call itself.

Eg ... (A, B, (D, E)C, F, G) ...

We will make the preceeding nodes first A, B, then the internal node C, its children D, E, and finally the succeeding nodes F, G

nexusparser Module

class galaxy.visualization.data_providers.phyloviz.nexusparser.**Nexus_Parser**
Bases: *galaxy.visualization.data_providers.phyloviz.newickparser.Newick_Parser*

checkComments (*line*)
Check to see if the line/lines is a comment.

parseFile (*filePath*)
passes a file and extracts its Nexus content.

parseNexus (*filename*)
Nexus data is stored in blocks between a line starting with begin and another line starting with end; Commends inside square brackets are to be ignored, For more information: http://wiki.christophchamp.com/index.php/NEXUS_file_format Nexus can store multiple trees

splitLinebyWhitespaces (*line*)
replace tabs and write spaces to a single write space, so we can properly split it.

phyloxmlparser Module

class galaxy.visualization.data_providers.phyloviz.phyloxmlparser.**Phyloxml_Parser**
Bases: galaxy.visualization.data_providers.phyloviz.baseparser.Base_Parser

Parses a phyloxml file into a json file that will be passed to PhyloViz for display

cleanTag(*tagString*)

parseFile(*filePath*)
passes a file and extracts its Phylogeny Tree content.

parseNode(*node*, *depth*)
Parses any node within a phyloxml tree and looks out for claude, which signals the creation of nodes - internal OR leaf

genome Package

genome Package Code for Galaxy genome visualizations.

visual_analytics Module

tracks Package

tracks Package Summary.py required to be in this module due to pickling.

summary Module

web Package

web Package The Galaxy web application framework

buildapp Module

form_builder Module Classes for generating HTML forms

class galaxy.web.form_builder.**AddressField**(*name*, *user=None*, *value=None*, *params=None*)
Bases: galaxy.web.form_builder.BaseField

static fields()

get_html(*disabled=False*)

class galaxy.web.form_builder.**BaseField**
Bases: object

get_disabled_str(*disabled=False*)

get_html(*prefix=''*)
Returns the html widget corresponding to the parameter

class galaxy.web.form_builder.**CheckboxField**(*name*, *checked=None*, *refresh_on_change=False*, *refresh_on_change_values=None*)

Bases: galaxy.web.form_builder.BaseField

A checkbox (boolean input)

```
>>> print CheckboxField( "foo" ).get_html()
<input type="checkbox" id="foo" name="foo" value="true"><input type="hidden" name="foo" value="t
>>> print CheckboxField( "bar", checked="yes" ).get_html()
<input type="checkbox" id="bar" name="bar" value="true" checked="checked"><input type="hidden" n
```

get_html(*prefix=''*, *disabled=False*)

static is_checked(*value*)

set_checked(*value*)

class galaxy.web.form_builder.**DrillDownField**(*name*, *multiple=None*, *display=None*, *refresh_on_change=False*, *options=[]*, *value=[]*, *refresh_on_change_values=[]*)

Bases: galaxy.web.form_builder.BaseField

A hierarchical select field, which allows users to 'drill down' a tree-like set of options.

```
>>> t = DrillDownField( "foo", multiple=True, display="checkbox", options=[{'name': 'Heading 1',
>>> print t.get_html()
<div class="form-row drilldown-container" id="drilldown--666f6f">
<div class="form-row-input">
<div><span class="form-toggle icon-button toggle-expand" id="drilldown--666f6f-68656164696e6731-
<input type="checkbox" name="foo" value="heading1" >Heading 1
</div><div class="form-row" id="drilldown--666f6f-68656164696e6731-container" style="float: left
<div class="form-row-input">
<input type="checkbox" name="foo" value="option1" >Option 1
</div>
<div class="form-row-input">
<input type="checkbox" name="foo" value="option2" >Option 2
</div>
<div class="form-row-input">
<div><span class="form-toggle icon-button toggle-expand" id="drilldown--666f6f-68656164696e6731-
<input type="checkbox" name="foo" value="heading1" >Heading 1
</div><div class="form-row" id="drilldown--666f6f-68656164696e6731-68656164696e6731-container" s
<div class="form-row-input">
<input type="checkbox" name="foo" value="option3" >Option 3
</div>
<div class="form-row-input">
<input type="checkbox" name="foo" value="option4" >Option 4
</div>
</div>
</div>
</div>
</div>
<div class="form-row-input">
<input type="checkbox" name="foo" value="option5" >Option 5
</div>
</div>
>>> t = DrillDownField( "foo", multiple=False, display="radio", options=[{'name': 'Heading 1', '
>>> print t.get_html()
<div class="form-row drilldown-container" id="drilldown--666f6f">
```

```
<div class="form-row-input">
<div><span class="form-toggle icon-button toggle-expand" id="drilldown--666f6f-68656164696e6731-
<input type="radio" name="foo" value="heading1" >Heading 1
</div><div class="form-row" id="drilldown--666f6f-68656164696e6731-container" style="float: left
<div class="form-row-input">
<input type="radio" name="foo" value="option1" >Option 1
</div>
<div class="form-row-input">
<input type="radio" name="foo" value="option2" >Option 2
</div>
<div class="form-row-input">
<div><span class="form-toggle icon-button toggle-expand" id="drilldown--666f6f-68656164696e6731-
<input type="radio" name="foo" value="heading1" >Heading 1
</div><div class="form-row" id="drilldown--666f6f-68656164696e6731-68656164696e6731-container" s
<div class="form-row-input">
<input type="radio" name="foo" value="option3" >Option 3
</div>
<div class="form-row-input">
<input type="radio" name="foo" value="option4" >Option 4
</div>
</div>
</div>
</div>
</div>
<div class="form-row-input">
<input type="radio" name="foo" value="option5" >Option 5
</div>
</div>
```

get_html (*prefix=''*)

class galaxy.web.form_builder.**FTPFileField** (*name*, *dir*, *ftp_site*, *value=None*)
Bases: *galaxy.web.form_builder.BaseField*

An FTP file upload input.

get_html (*prefix=''*)

tfoot = '\n </tbody>\n </table>\n '

thead = '\n <table id="grid-table" class="grid">\n <thead id="grid-table-header">\n <tr>\n <th id="select-header"></tl

trow = '\n <tr>\n <td><input type="checkbox" name="%s%s" value="%s"/></td>\n <td>%s</td>\n <td>%s</td>\n <t

class galaxy.web.form_builder.**FileField** (*name*, *value=None*, *ajax=False*)
Bases: *galaxy.web.form_builder.BaseField*

A file upload input.

```
>>> print FileField( "foo" ).get_html()
<input type="file" name="foo">
>>> print FileField( "foo", ajax = True ).get_html()
<input type="file" name="foo" galaxy-ajax-upload="true">
```

get_html (*prefix=''*)

class galaxy.web.form_builder.**HiddenField** (*name*, *value=None*)
Bases: *galaxy.web.form_builder.BaseField*

A hidden field.

```
>>> print HiddenField( "foo", 100 ).get_html()
<input type="hidden" name="foo" value="100">
```

get_html(*prefix=''*)

class galaxy.web.form_builder.**HistoryField**(*name*, *user=None*, *value=None*, *params=None*)
Bases: *galaxy.web.form_builder.BaseField*

get_display_text()

get_html(*disabled=False*)

class galaxy.web.form_builder.**LibraryField**(*name*, *value=None*, *trans=None*)
Bases: *galaxy.web.form_builder.BaseField*

get_display_text()

get_html(*prefix=''*, *disabled=False*)

class galaxy.web.form_builder.**PasswordField**(*name*, *size=None*, *value=None*)
Bases: *galaxy.web.form_builder.BaseField*

A password input box. text appears as "**"

```
>>> print PasswordField( "foo" ).get_html()
<input type="password" name="foo" size="10" value="">
>>> print PasswordField( "bins", size=4, value="default" ).get_html()
<input type="password" name="bins" size="4" value="default">
```

get_html(*prefix=''*, *disabled=False*)

set_size(*size*)

class galaxy.web.form_builder.**SelectField**(*name*, *multiple=None*, *display=None*, *refresh_on_change=False*, *refresh_on_change_values=None*, *size=None*)
Bases: *galaxy.web.form_builder.BaseField*

A select field.

```
>>> t = SelectField( "foo", multiple=True )
>>> t.add_option( "tuti", 1 )
>>> t.add_option( "fruity", "x" )
>>> print t.get_html()
<select name="foo" multiple>
<option value="1">tuti</option>
<option value="x">fruity</option>
</select>
```

```
>>> t = SelectField( "bar" )
>>> t.add_option( "automatic", 3 )
>>> t.add_option( "bazooty", 4, selected=True )
>>> print t.get_html()
<select name="bar" last_selected_value="4">
<option value="3">automatic</option>
<option value="4" selected>bazooty</option>
</select>
```

```
>>> t = SelectField( "foo", display="radio" )
>>> t.add_option( "tuti", 1 )
>>> t.add_option( "fruity", "x" )
>>> print t.get_html()
<div><input type="radio" name="foo" value="1" id="foo|1"><label class="inline" for="foo|1">tuti<
<div><input type="radio" name="foo" value="x" id="foo|x"><label class="inline" for="foo|x">fruit
```

```
>>> t = SelectField( "bar", multiple=True, display="checkboxes" )
>>> t.add_option( "automatic", 3 )
>>> t.add_option( "bazooty", 4, selected=True )
>>> print t.get_html()
<div class="checkUncheckAllPlaceholder" checkbox_name="bar"></div>
<div><input type="checkbox" name="bar" value="3" id="bar|3"><label class="inline" for="bar|3">au
<div><input type="checkbox" name="bar" value="4" id="bar|4" checked='checked'><label class="inli
```

add_option(*text*, *value*, *selected=False*)

get_html(*prefix=''*, *disabled=False*)

get_html_checkboxes(*prefix=''*, *disabled=False*)

get_html_default(*prefix=''*, *disabled=False*)

get_html_radio(*prefix=''*, *disabled=False*)

get_selected(*return_label=False*, *return_value=False*, *multi=False*)

Return the currently selected option's label, value or both as a tuple. For multi-select lists, a list is returned.

to_dict()

class galaxy.web.form_builder.**SwitchingSelectField**(*delegate_fields*, *default_field=None*)

Bases: *galaxy.web.form_builder.BaseField*

get_html(*prefix=''*, *disabled=False*)

primary_field

class galaxy.web.form_builder.**TextArea**(*name*, *size=None*, *value=None*)

Bases: *galaxy.web.form_builder.BaseField*

A standard text area box.

```
>>> print TextArea( "foo" ).get_html()
<textarea name="foo" rows="5" cols="25"></textarea>
>>> print TextArea( "bins", size="4x5", value="default" ).get_html()
<textarea name="bins" rows="4" cols="5">default</textarea>
```

get_html(*prefix=''*, *disabled=False*)

set_size(*rows*, *cols*)

class galaxy.web.form_builder.**TextField**(*name*, *size=None*, *value=None*)

Bases: *galaxy.web.form_builder.BaseField*

A standard text input box.

```
>>> print TextField( "foo" ).get_html()
<input type="text" name="foo" size="10" value="">
>>> print TextField( "bins", size=4, value="default" ).get_html()
<input type="text" name="bins" size="4" value="default">
```

get_html(*prefix=''*, *disabled=False*)

set_size(*size*)

class galaxy.web.form_builder.**WorkflowField**(*name*, *user=None*, *value=None*, *params=None*)
Bases: galaxy.web.form_builder.BaseField

get_html(*disabled=False*)

class galaxy.web.form_builder.**WorkflowMappingField**(*name*, *user=None*, *value=None*, *params=None*, ***kwd*)
Bases: galaxy.web.form_builder.BaseField

get_display_text()

get_html(*disabled=False*)

galaxy.web.form_builder.**build_select_field**(*trans*, *objs*, *label_attr*, *select_field_name*, *initial_value='none'*, *selected_value='none'*, *refresh_on_change=False*, *multiple=False*, *display=None*, *size=None*)
Build a SelectField given a set of objects. The received params are:

- objs: the set of objects used to populate the option list
- label_attr: the attribute of each obj (e.g., name, email, etc) whose value is used to populate each option label.
  - If the string 'self' is passed as label_attr, each obj in objs is assumed to be a string, so the obj itself is used
- select_field_name: the name of the SelectField
- initial_value: the value of the first option in the SelectField - allows for an option telling the user to select something
- selected_value: the value of the currently selected option
- refresh_on_change: True if the SelectField should perform a refresh_on_change

galaxy.web.form_builder.**get_suite**()
Get unittest suite for this module

params Module Mixins for parsing web form and API parameters

class galaxy.web.params.**BaseParamParser**
Bases: object

get_params(*kwargs*)

class galaxy.web.params.**QuotaParamParser**
Bases: galaxy.web.params.BaseParamParser

get_quota_params(*kwargs*)

Subpackages

base Package

controller Module Contains functionality needed in every web interface

class galaxy.web.base.controller.**BaseAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseController

get_object(*trans*, *id*, *class_name*, *check_ownership=False*, *check_accessible=False*, *deleted=None*)

not_implemented(*trans*, ***kwd*)

validate_in_users_and_groups(*trans*, *payload*)
For convenience, in_users and in_groups can be encoded IDs or emails/group names in the API.

class galaxy.web.base.controller.**BaseController**(*app*)
Bases: object

Base class for Galaxy web application controllers.

decode_id(*id*)

encode_all_ids(*trans*, *rval*, *recursive=False*)
Encodes all integer values in the dict rval whose keys are 'id' or end with '_id'

It might be useful to turn this in to a decorator

get_class(*class_name*)
Returns the class object that a string denotes. Without this method, we'd have to do eval(<class_name>).

get_group(*trans*, *id*, *check_ownership=False*, *check_accessible=False*, *deleted=None*)

get_object(*trans*, *id*, *class_name*, *check_ownership=False*, *check_accessible=False*, *deleted=None*)
Convenience method to get a model object with the specified checks.

get_role(*trans*, *id*, *check_ownership=False*, *check_accessible=False*, *deleted=None*)

get_toolbox()
Returns the application toolbox

get_user(*trans*, *id*, *check_ownership=False*, *check_accessible=False*, *deleted=None*)

parse_filter_params(*qdict*, *filter_attr_key='q'*, *filter_value_key='qv'*, *attr_op_split_char='-'*)

parse_limit_offset(*qdict*)

class galaxy.web.base.controller.**BaseUIController**(*app*)
Bases: *galaxy.web.base.controller.BaseController*

get_object(*trans*, *id*, *class_name*, *check_ownership=False*, *check_accessible=False*, *deleted=None*)

exception galaxy.web.base.controller.**ControllerUnavailable**
Bases: exceptions.Exception

Deprecated: *BaseController* used to be available under the name *Root*

class galaxy.web.base.controller.**CreatesApiKeysMixin**
Mixing centralizing logic for creating API keys for user objects.

Deprecated - please use api_keys.ApiKeyManager for new development.

create_api_key(*trans*, *user*)

class galaxy.web.base.controller.**CreatesUsersMixin**
Mixin centralizing logic for user creation between web and API controller.

Web controller handles additional features such e-mail subscription, activation, user forms, etc.... API created users are much more vanilla for the time being.

create_user(*trans*, *email*, *username*, *password*)

class galaxy.web.base.controller.**Datatype**(*extension*, *dtype*, *type_extension*, *mimetype*, *display_in_upload*)
Bases: object

Used for storing in-memory list of datatypes currently in the datatypes registry.

class galaxy.web.base.controller.**ExportsHistoryMixin**

queue_history_export(*trans*, *history*, *gzip=True*, *include_hidden=False*, *include_deleted=False*)

serve_ready_history_export(*trans*, *jeha*)

class galaxy.web.base.controller.**ImportsHistoryMixin**

queue_history_import(*trans*, *archive_type*, *archive_source*)

galaxy.web.base.controller.**Root**
alias of *BaseController*

class galaxy.web.base.controller.**SharableItemSecurityMixin**
Mixin for handling security for sharable items.

security_check(*trans*, *item*, *check_ownership=False*, *check_accessible=False*)
Security checks for an item: checks if (a) user owns item or (b) item is accessible to user.

class galaxy.web.base.controller.**SharableMixin**
Mixin for a controller that manages an item that can be shared.

create_item_slug(*sa_session*, *item*)
Create/set item slug. Slug is unique among user's importable items for item's class. Returns true if item's slug was set/changed; false otherwise.

display_by_username_and_slug(*trans*, *username*, *slug*)
Display item by username and slug.

get_item(*trans*, *id*)
Return item based on id.

get_item_content_async(*trans*, **args*, ***kwargs*)
Returns item content in HTML format.

get_name_and_link_async(*trans*, **args*, ***kwargs*)
Returns item's name and link.

set_public_username(*trans*, **args*, ***kwargs*)
Set user's public username and delegate to sharing()

set_slug_async(*trans*, **args*, ***kwargs*)

share(*trans*, **args*, ***kwargs*)
Handle sharing an item with a particular user.

sharing(*trans*, **args*, ***kwargs*)
Handle item sharing.

class galaxy.web.base.controller.**UsesExtendedMetadataMixin**
Bases: *galaxy.web.base.controller.SharableItemSecurityMixin*

Mixin for getting and setting item extended metadata.

create_extended_metadata(*trans*, *extmeta*)
Create/index an extended metadata object. The returned object is not associated with any items

delete_extended_metadata(*trans*, *item*)

get_item_extended_metadata_obj(*trans*, *item*)
Given an item object (such as a LibraryDatasetDatasetAssociation), find the object of the associated extended metadata

set_item_extended_metadata_obj (*trans*, *item*, *extmeta_obj*, *check_writable=False*)

unset_item_extended_metadata_obj (*trans*, *item*, *check_writable=False*)

class galaxy.web.base.controller.**UsesFormDefinitionsMixin**
Mixin for controllers that use Galaxy form objects.

add_template (*trans*, *cntrller*, *item_type*, *form_type*, ***kwd*)

build_form_id_select_field (*trans*, *forms*, *selected_value='none'*)

clean_field_contents (*widgets*, ***kwd*)

delete_template (*trans*, *cntrller*, *item_type*, *form_type*, ***kwd*)

edit_template (*trans*, *cntrller*, *item_type*, *form_type*, ***kwd*)

edit_template_info (*trans*, *cntrller*, *item_type*, *form_type*, ***kwd*)

field_param_values_ok (*widget_name*, *widget_type*, ***kwd*)

get_all_forms (*trans*, *all_versions=False*, *filter=None*, *form_type='All'*)
Return all the latest forms from the form_definition_current table if all_versions is set to True. Otherwise return all the versions of all the forms from the form_definition table.

get_all_forms_by_type (*trans*, *cntrller*, *form_type*)

get_form_values (*trans*, *user*, *form_definition*, ***kwd*)
Returns the name:value dictionary containing all the form values

get_item_and_stuff (*trans*, *item_type*, ***kwd*)

populate_widgets_from_kwd (*trans*, *widgets*, ***kwd*)

save_widget_field (*trans*, *field_obj*, *widget_name*, ***kwd*)

widget_fields_have_contents (*widgets*)

class galaxy.web.base.controller.**UsesLibraryMixin**

get_library (*trans*, *id*, *check_ownership=False*, *check_accessible=True*)

class galaxy.web.base.controller.**UsesLibraryMixinItems**
Bases: *galaxy.web.base.controller.SharableItemSecurityMixin*

can_current_user_add_to_library_item (*trans*, *item*)

check_user_can_add_to_library_item (*trans*, *item*, *check_accessible=True*)
Raise exception if user cannot add to the specified library item (i.e. Folder). Can set check_accessible to False if folder was loaded with this check.

copy_hda_to_library_folder (*trans*, *hda*, *library_folder*, *roles=None*, *ldda_message=''*)

get_library_dataset (*trans*, *id*, *check_ownership=False*, *check_accessible=True*)

get_library_dataset_dataset_association (*trans*, *id*, *check_ownership=False*, *check_accessible=True*)

get_library_folder (*trans*, *id*, *check_ownership=False*, *check_accessible=True*)

class galaxy.web.base.controller.**UsesQuotaMixin**
Bases: object

get_quota (*trans*, *id*, *check_ownership=False*, *check_accessible=False*, *deleted=None*)

class galaxy.web.base.controller.**UsesStoredWorkflowMixin**
Bases: galaxy.web.base.controller.SharableItemSecurityMixin, galaxy.model.item_attrs.UsesAnnotations

Mixin for controllers that use StoredWorkflow objects.

get_stored_workflow(*trans*, *id*, *check_ownership=True*, *check_accessible=False*)
Get a StoredWorkflow from the database by id, verifying ownership.

get_stored_workflow_steps(*trans*, *stored_workflow*)
Restores states for a stored workflow's steps.

class galaxy.web.base.controller.**UsesTagsMixin**
Bases: galaxy.web.base.controller.SharableItemSecurityMixin

get_tag_handler(*trans*)

get_user_tags_used(*trans*, *user=None*)
Return a list of distinct 'user_tname:user_value' strings that the given user has used.

user defaults to trans.user. Returns an empty list if no user is given and trans.user is anonymous.

set_tags_from_list(*trans*, *item*, *new_tags_list*, *user=None*)

class galaxy.web.base.controller.**UsesVisualizationMixin**
Bases: galaxy.web.base.controller.UsesLibraryMixinItems

Mixin for controllers that use Visualization objects.

add_visualization_revision(*trans*, *visualization*, *config*, *title*, *dbkey*)
Adds a new *VisualizationRevision* to the given *visualization* with the given parameters and set its parent visualization's *latest_revision* to the new revision.

create_visualization(*trans*, *type*, *title='Untitled Visualization'*, *slug=None*, *dbkey=None*, *annotation=None*, *config={}*, *save=True*)
Create visualiation and first revision.

get_hda(*trans*, *dataset_id*, *check_ownership=True*, *check_accessible=False*, *check_state=True*)
Get an HDA object by id performing security checks using the current transaction.

get_hda_or_ldda(*trans*, *hda_ldda*, *dataset_id*)
Returns either HDA or LDDA for hda/ldda and id combination.

get_new_track_config(*trans*, *dataset*)
Returns track configuration dict for a dataset.

get_published_visualizations(*trans*, *exclude_user=None*, *order_by=None*, *query_only=False*)
Return query or query results for published visualizations optionally excluding the user in *exclude_user*.

Set *order_by* to a column or list of columns to change the order returned. Defaults to *DEFAULT_ORDER_BY*. Set *query_only* to return just the query for further filtering or processing.

get_tool_def(*trans*, *hda*)
Returns definition of an interactive tool for an HDA.

get_visualization(*trans*, *id*, *check_ownership=True*, *check_accessible=False*)
Get a Visualization from the database by id, verifying ownership.

get_visualization_config(*trans*, *visualization*)
Returns a visualization's configuration. Only works for trackster visualizations right now.

get_visualization_dict(*visualization*)
Return a set of detailed attributes for a visualization in dictionary form. The visualization's latest_revision

is returned in its own sub-dictionary. NOTE: that encoding ids isn't done here should happen at the caller level.

get_visualization_revision_dict (*revision*)
Return a set of detailed attributes for a visualization in dictionary form. NOTE: that encoding ids isn't done here should happen at the caller level.

get_visualization_summary_dict (*visualization*)
Return a set of summary attributes for a visualization in dictionary form. NOTE: that encoding ids isn't done here should happen at the caller level.

get_visualizations_by_user (*trans*, *user*, *order_by=None*, *query_only=False*)
Return query or query results of visualizations filtered by a user.

Set *order_by* to a column or list of columns to change the order returned. Defaults to *DEFAULT_ORDER_BY*. Set *query_only* to return just the query for further filtering or processing.

get_visualizations_shared_with_user (*trans*, *user*, *order_by=None*, *query_only=False*)
Return query or query results for visualizations shared with the given user.

Set *order_by* to a column or list of columns to change the order returned. Defaults to *DEFAULT_ORDER_BY*. Set *query_only* to return just the query for further filtering or processing.

import_visualization (*trans*, *id*, *user=None*)
Copy the visualization with the given id and associate the copy with the given user (defaults to trans.user).

Raises *ItemAccessibilityException* if *user* is not passed and the current user is anonymous, and if the visualization is not *importable*. Raises *ItemDeletionException* if the visualization has been deleted.

save_visualization (*trans*, *config*, *type*, *id=None*, *title=None*, *dbkey=None*, *slug=None*, *annotation=None*)

viz_types = ['trackster']

galaxy.web.base.controller.**sort_by_attr** (*seq*, *attr*)
Sort the sequence of objects by object's attribute Arguments: seq - the list or any sequence (including immutable one) of objects to sort. attr - the name of attribute to sort by

Subpackages

controllers Package

admin Module

class galaxy.web.base.controllers.admin.**Admin**
Bases: object

center (*trans*, **args*, ***kwargs*)

create_group (*trans*, **args*, ***kwargs*)

create_new_user (*trans*, **args*, ***kwargs*)

create_role (*trans*, **args*, ***kwargs*)

delete_operation = None

group_list_grid = None

groups (*trans*, **args*, ***kwargs*)

index (*trans*, **args*, ***kwargs*)

job_info (*trans*, **args*, ***kwargs*)

jobs (*trans*, **args*, ***kwargs*)

manage_roles_and_groups_for_user (*trans*, **args*, ***kwargs*)

manage_users_and_groups_for_role (*trans*, **args*, ***kwargs*)

manage_users_and_roles_for_group (*trans*, **args*, ***kwargs*)

mark_group_deleted (*trans*, **args*, ***kwargs*)

mark_role_deleted (*trans*, **args*, ***kwargs*)

mark_user_deleted (*trans*, **args*, ***kwargs*)

name_autocomplete_data (*trans*, **args*, ***kwargs*)
Return autocomplete data for user emails

package_tool (*trans*, **args*, ***kwargs*)

purge_group (*trans*, **args*, ***kwargs*)

purge_operation = **None**

purge_role (*trans*, **args*, ***kwargs*)

purge_user (*trans*, **args*, ***kwargs*)

quota_list_grid = **None**

reload_tool (*trans*, **args*, ***kwargs*)

rename_group (*trans*, **args*, ***kwargs*)

rename_role (*trans*, **args*, ***kwargs*)

repository_list_grid = **None**

reset_user_password (*trans*, **args*, ***kwargs*)

role_list_grid = **None**

roles (*trans*, **args*, ***kwargs*)

tool_version_list_grid = **None**

tool_versions (*trans*, **args*, ***kwargs*)

undelete_group (*trans*, **args*, ***kwargs*)

undelete_operation = **None**

undelete_role (*trans*, **args*, ***kwargs*)

undelete_user (*trans*, **args*, ***kwargs*)

user_list_grid = **None**

users (*trans*, **args*, ***kwargs*)

galaxy.web.base.controllers.admin.**get_group** (*trans*, *id*)
Get a Group from the database by id.

galaxy.web.base.controllers.admin.**get_quota** (*trans*, *id*)
Get a Quota from the database by id.

galaxy.web.base.controllers.admin.**get_role** (*trans*, *id*)
Get a Role from the database by id.

galaxy.web.base.controllers.admin.**get_user**(*trans*, *user_id*)
Get a User from the database by id.

galaxy.web.base.controllers.admin.**get_user_by_username**(*trans*, *username*)
Get a user from the database by username

framework Package

framework Package Galaxy web application framework

base Module A simple WSGI application/framework.

class galaxy.web.framework.base.**DefaultWebTransaction**(*environ*)
Bases: object

Wraps the state of a single web transaction (request/response cycle).
TODO: Provide hooks to allow application specific state to be included in here.
session
Property that replaces itself with a calculated value the first time it is used.

class galaxy.web.framework.base.**FieldStorage**(*fp=None, headers=None, outerboundary=''', environ={'CELERY_LOG_REDIRECT_LEVEL': 'WARNING', '_MP_FORK_LOGFILE_': '/home/docs/log/celery_proc.log', 'NEW_RELIC_CONFIG_FILE': '/home/docs/newrelic.ini', 'CELERY_LOG_REDIRECT': '1', 'LOGNAME': 'docs', 'USER': 'docs', 'PATH': '/home/docs/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/*
'HOME': '/home/docs', 'PS1': '(docs)', 'TERM': 'linux', 'SHELL': '/bin/bash', 'TZ': 'America/Chicago', '_MP_FORK_LOGFORMAT_': '[%(asctime)s: %(levelname)s/%(processName)s] %(message)s', 'SHLVL': '1', 'SUPERVISOR_ENABLED': '1', 'EDITOR': 'vim', 'DJANGO_PROJECT_DIR': '/home/docs/checkouts/readthedocs.org/readthedocs', 'SUDO_USER': 'root', 'CELERY_LOG_FILE': '/home/docs/log/celery_proc.log', 'USERNAME': 'docs', 'READTHEDOCS': 'True', 'SUDO_UID': '0', 'VIRTUAL_ENV': '/home/docs', 'SUPERVISOR_PROCESS_NAME': 'celery', 'SUPERVISOR_SERVER_URL': 'unix:///home/docs/run/supervisord-docs.sock', '_': '/home/docs/bin/supervisord', 'SUDO_COMMAND': '/bin/bash -c /home/docs/bin/supervisord –nodaemon', 'SUDO_GID': '0', 'CELERY_LOADER': 'djcelery.loaders.DjangoLoader', 'OLDPWD': '/home/docs', 'PWD': '/home/docs/checkouts/readthedocs.org/readthedocs', '_MP_FORK_LOGLEVEL_': '20', 'MAIL': '/var/mail/docs', 'CELERY_LOG_LEVEL': '20', 'SUPERVISOR_GROUP_NAME': 'celery'}, keep_blank_values=0, strict_parsing=0)

Bases: cgi.FieldStorage

make_file(*binary=None*)

read_lines()

class galaxy.web.framework.base.**LazyProperty**(*func*)

Bases: object

Property that replaces itself with a calculated value the first time it is used.

class galaxy.web.framework.base.**Request**(*environ*)

Bases: webob.Request

Encapsulates an HTTP request.

base

Property that replaces itself with a calculated value the first time it is used.

browser_url
Property that replaces itself with a calculated value the first time it is used.

cookies
Property that replaces itself with a calculated value the first time it is used.

path
Property that replaces itself with a calculated value the first time it is used.

protocol
Descriptor that delegates a property to a key in the environ member of the associated object (provides property style access to keys in the WSGI environment)

remote_host
Property that replaces itself with a calculated value the first time it is used.

remote_hostname
Property that replaces itself with a calculated value the first time it is used.

remote_port
Descriptor that delegates a property to a key in the environ member of the associated object (provides property style access to keys in the WSGI environment)

class galaxy.web.framework.base.**Response**
Bases: object

Describes an HTTP response. Currently very simple since the actual body of the request is handled separately.

get_content_type()

send_redirect(*url*)
Send an HTTP redirect response to (target *url*)

set_content_type(*type*)
Sets the Content-Type header

wsgi_headeritems()
Return headers in format appropriate for WSGI *start_response*

wsgi_status()
Return status line in format appropriate for WSGI *start_response*

class galaxy.web.framework.base.**WSGIEnvironmentProperty**(*key*, *default='*')
Bases: object

Descriptor that delegates a property to a key in the environ member of the associated object (provides property style access to keys in the WSGI environment)

class galaxy.web.framework.base.**WebApplication**
Bases: object

A simple web application which maps requests to objects using routes, and to methods on those objects in the CherryPy style. Thus simple argument mapping in the CherryPy style occurs automatically, but more complicated encoding of arguments in the PATH_INFO can be performed with routes.

add_api_controller(*controller_name*, *controller*)

add_route(*route*, ***kwargs*)
Add a route to match a URL with a method. Accepts all keyword arguments of *routes.Mapper.connect*. Every route should result in at least a controller value which corresponds to one of the objects added with *add_controller*. It optionally may yield an *action* argument which will be used to locate the method to call on the controller. Additional arguments will be passed to the method as keyword args.

add_ui_controller(*controller_name*, *controller*)
Add a controller class to this application. A controller class has methods which handle web requests. To connect a URL to a controller's method use *add_route*.

finalize_config()
Call when application is completely configured and ready to serve requests

handle_controller_exception(*e*, *trans*, ***kwargs*)
Allow handling of exceptions raised in controller methods.

handle_request(*environ*, *start_response*)

make_body_iterable(*trans*, *body*)

set_transaction_factory(*transaction_factory*)
Use the callable *transaction_factory* to create the transaction which will be passed to requests.

trace(***fields*)

galaxy.web.framework.base.**flatten**(*seq*)
Flatten a possible nested set of iterables

galaxy.web.framework.base.**iterate_file**(*file*)
Progressively return chunks from *file*.

galaxy.web.framework.base.**lazy_property**
alias of *LazyProperty*

galaxy.web.framework.base.**send_file**(*start_response*, *trans*, *body*)

openid_manager Module Manage the OpenID consumer and related data stores.

class galaxy.web.framework.openid_manager.**OpenIDManager**(*cache_path*)
Bases: object

CANCEL = 'cancel'

FAILURE = 'failure'

SETUP_NEEDED = 'setup_needed'

SUCCESS = 'success'

add_sreg(*trans*, *request*, *required=None*, *optional=None*)

get_consumer(*trans*)

get_session(*trans*)

get_sreg(*info*)

persist_session(*trans*, *oidconsumer*)

galaxy.web.framework.openid_manager.**oidlog**(*message*, *level=0*)

Subpackages

helpers Package

helpers Package Galaxy web framework helpers

galaxy.web.framework.helpers.**css**(**args*)
Take a list of stylesheet names (no extension) and return appropriate string of link tags.

Cache-bust with time that server started running on

galaxy.web.framework.helpers.**iff**(*a*, *b*, *c*)
Ternary shortcut

galaxy.web.framework.helpers.**is_true**(*val*)
Returns true if input is a boolean and true or is a string and looks like a true value.

galaxy.web.framework.helpers.**js**(**args*)
Take a prefix and list of javascript names and return appropriate string of script tags.

galaxy.web.framework.helpers.**js_helper**(*prefix*, **args*)
Take a prefix and list of javascript names and return appropriate string of script tags.

Cache-bust with time that server started running on

galaxy.web.framework.helpers.**md5**(*s*)
Return hex encoded md5 hash of string s

galaxy.web.framework.helpers.**templates**(**args*)
Take a list of template names (no extension) and return appropriate string of script tags.

galaxy.web.framework.helpers.**time_ago**(*x*)
Convert a datetime to a string.

galaxy.web.framework.helpers.**to_unicode**(*a_string*)
Convert a string to unicode in utf-8 format; if string is already unicode, does nothing because string's encoding cannot be determined by introspection.

galaxy.web.framework.helpers.**truncate**(*content*, *length=100*, *suffix='...'*)
Smart string truncation

grids Module

class galaxy.web.framework.helpers.grids.**BooleanColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_single_filter(*user*, *a_filter*)

sort(*trans*, *query*, *ascending*, *column_name=None*)
Sort query using this column.

class galaxy.web.framework.helpers.grids.**CommunityRatingColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn, galaxy.model.item_attrs.UsesItemRatings

Column that displays community ratings for an item.

get_value(*trans*, *grid*, *item*)

sort(*trans*, *query*, *ascending*, *column_name=None*)

class galaxy.web.framework.helpers.grids.**CommunityTagsColumn**(*col_name*, *key*, *model_class=None*, *model_tag_association_class=None*, *filterable=None*, *grid_name=None*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

Column that supports community tags.

filter(*trans*, *user*, *query*, *column_filter*)

Modify query to filter model_class by tag. Multiple filters are ANDed.

get_filter(*trans*, *user*, *column_filter*)

get_value(*trans*, *grid*, *item*)

class galaxy.web.framework.helpers.grids.**DateTimeColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

sort(*trans*, *query*, *ascending*, *column_name=None*)

Sort query using this column.

class galaxy.web.framework.helpers.grids.**DeletedColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.GridColumn*

Column that tracks and filters for items with deleted attribute.

filter(*trans*, *user*, *query*, *column_filter*)
Modify query to filter self.model_class by state.

get_accepted_filters()
Returns a list of accepted filters for this column.

class galaxy.web.framework.helpers.grids.**DisplayByUsernameAndSlugGridOperation**(*label*, *key=None*, *condition=None*, *allow_multiple=True*, *allow_popup=True*, *target=None*, *url_args=None*, *async_compatible=*
confirm=None, *global_operation=N*
inbound=False)

Bases: *galaxy.web.framework.helpers.grids.GridOperation*

Operation to display an item by username and slug.

get_url_args(*item*)

class galaxy.web.framework.helpers.grids.**Grid**
Bases: object

Specifies the content and format of a grid (data table).

apply_query_filter(*trans*, *query*, ***kwargs*)

async_template = 'grid_base_async.mako'

build_initial_query(*trans*, ***kwargs*)

columns = []

cur_filter_pref_name = '.filter'

cur_sort_key_pref_name = '.sort_key'

default_filter = {}

default_sort_key = None

exposed = True

get_current_item(*trans*, ***kwargs*)

get_ids(***kwargs*)

global_actions = []

handle_operation(*trans*, *operation*, *ids*, ***kwargs*)

info_text = None

legend = None

model_class = None

num_page_links = 10

num_rows_per_page = 25

operations = []

pass_through_operations = {}

preserve_state = False

show_item_checkboxes = False

standard_filters = []

template = 'grid_base.mako'

title = ''

use_async = False

use_hide_message = True

use_paging = False

class galaxy.web.framework.helpers.grids.**GridAction**(*label=None*, *url_args=None*, *inbound=False*)

Bases: object

class galaxy.web.framework.helpers.grids.**GridColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: object

filter(*trans*, *user*, *query*, *column_filter*)

Modify query to reflect the column filter.

get_accepted_filters()

Returns a list of accepted filters for this column.

get_link(*trans*, *grid*, *item*)

get_value(*trans*, *grid*, *item*)

sort(*trans*, *query*, *ascending*, *column_name=None*)

Sort query using this column.

class galaxy.web.framework.helpers.grids.**GridColumnFilter**(*label*, *args=None*)

Bases: object

get_url_args()

class galaxy.web.framework.helpers.grids.**GridOperation**(*label*, *key=None*, *condition=None*, *allow_multiple=True*, *allow_popup=True*, *target=None*, *url_args=None*, *async_compatible=False*, *confirm=None*, *global_operation=None*, *inbound=False*)

Bases: object

allowed(*item*)

get_url_args(*item*)

class galaxy.web.framework.helpers.grids.**IndividualTagsColumn**(*col_name*, *key*, *model_class=None*, *model_tag_association_class=None*, *filterable=None*, *grid_name=None*)

Bases: *galaxy.web.framework.helpers.grids.CommunityTagsColumn*

Column that supports individual tags.

get_filter(*trans*, *user*, *column_filter*)

get_value(*trans*, *grid*, *item*)

class galaxy.web.framework.helpers.grids.**IntegerColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

Integer column that employs freetext, but checks that the text is an integer, so support filtering on integer values.

IMPORTANT NOTE: grids that use this column type should not include the column in the cols_to_filter list of MulticolFilterColumn (i.e., searching on this column type should not be performed in the grid's standard search - it won't throw exceptions, but it also will not find what you're looking for). Grids that search on this column should use 'filterable="advanced"' so that searching is only performed in the advanced search component, restricting the search to the specific column.

This is useful for searching on object ids or other integer columns. See the JobIdColumn column in the SpecifiedDateListGrid class in the jobs controller of the reports webapp for an example.

get_single_filter(*user*, *a_filter*)

sort(*trans*, *query*, *ascending*, *column_name=None*)

Sort query using this column.

class galaxy.web.framework.helpers.grids.**MulticolFilterColumn**(*col_name*, *cols_to_filter*, *key*, *visible*, *filterable='default'*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

Column that performs multicolumn filtering.

filter (*trans*, *user*, *query*, *column_filter*)
Modify query to filter model_class by tag. Multiple filters are ANDed.

class galaxy.web.framework.helpers.grids.**OwnerAnnotationColumn** (*col_name*, *key*, *model_class=None*, *model_annotation_association_class=None*, *filterable=None*)

Bases: galaxy.web.framework.helpers.grids.TextColumn, galaxy.model.item_attrs.UsesAnnotations

Column that displays and filters item owner's annotations.

get_single_filter (*user*, *a_filter*)
Filter by annotation and annotation owner.

get_value (*trans*, *grid*, *item*)
Returns first 150 characters of annotation.

class galaxy.web.framework.helpers.grids.**OwnerColumn** (*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

Column that lists item's owner.

get_value (*trans*, *grid*, *item*)

sort (*trans*, *query*, *ascending*, *column_name=None*)
Sort column using case-insensitive alphabetical sorting on item's username.

class galaxy.web.framework.helpers.grids.**PublicURLColumn** (*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

Column displays item's public URL based on username and slug.

get_link (*trans*, *grid*, *item*)

class galaxy.web.framework.helpers.grids.**ReverseSortColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

Column that reverses sorting; this is useful when the natural sort is descending.

sort(*trans*, *query*, *ascending*, *column_name=None*)

class galaxy.web.framework.helpers.grids.**SharingStatusColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

Grid column to indicate sharing status.

filter(*trans*, *user*, *query*, *column_filter*)
Modify query to filter histories by sharing status.

get_accepted_filters()
Returns a list of accepted filters for this column.

get_link(*trans*, *grid*, *item*)

get_value(*trans*, *grid*, *item*)

class galaxy.web.framework.helpers.grids.**StateColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

Column that tracks and filters for items with state attribute.

IMPORTANT NOTE: self.model_class must have a states Bunch or dict if this column type is used in the grid.

filter(*trans*, *user*, *query*, *column_filter*)
Modify query to filter self.model_class by state.

get_accepted_filters()
Returns a list of accepted filters for this column.

get_value(*trans*, *grid*, *item*)

class galaxy.web.framework.helpers.grids.**TextColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

Generic column that employs freetext and, hence, supports freetext, case-independent filtering.

filter(*trans*, *user*, *query*, *column_filter*)
Modify query to filter using free text, case independence.

get_filter(*trans*, *user*, *column_filter*)
Returns a SQLAlchemy criterion derived from column_filter.

get_single_filter(*user*, *a_filter*)
Returns a SQLAlchemy criterion derived for a single filter. Single filter is the most basic filter–usually a string–and cannot be a list.

sort(*trans*, *query*, *ascending*, *column_name=None*)
Sort column using case-insensitive alphabetical sorting.

middleware Package

middleware Package WSGI Middleware.

profile Module Middleware that profiles the request with cProfile and displays profiling information at the bottom of each page.

class galaxy.web.framework.middleware.profile.**ProfileMiddleware**(*app*, *global_conf=None*, *limit=40*)

Bases: object

Middleware that profiles all requests.

All HTML pages will have profiling information appended to them. The data is isolated to that single request, and does not include data from previous requests.

galaxy.web.framework.middleware.profile.**func_std_string**(*func_name*)
Match what old profile produced

galaxy.web.framework.middleware.profile.**get_func_list**(*stats*, *sel_list*)
Use 'sel_list' to select a list of functions to display.

galaxy.web.framework.middleware.profile.**pstats_as_html**(*stats*, **sel_list*)
Return an HTML representation of a pstats.Stats object.

remoteuser Module Middleware for handling $REMOTE_USER if use_remote_user is enabled.

class galaxy.web.framework.middleware.remoteuser.**RemoteUser**(*app*, *maildomain=None*, *display_servers=None*, *admin_users=None*, *remote_user_header=None*, *remote_user_secret_header=None*)

Bases: object

error(*start_response*, *title='Access denied'*, *message='Please contact your local Galaxy administrator.'*)

static Module

class galaxy.web.framework.middleware.static.**CacheableStaticURLParser**(*directory*, *cache_seconds=None*)

Bases: paste.urlparser.StaticURLParser

galaxy.web.framework.middleware.static.**make_static**(*global_conf*, *document_root*, *cache_seconds=None*)

translogger Module Middleware for logging requests, using Apache combined log format

class galaxy.web.framework.middleware.translogger.**TransLogger**(*application*, *logger=None*, *format=None*, *logging_level=20*, *logger_name='wsgi'*, *setup_console_handler=True*, *set_logger_level=10*)

Bases: object

This logging middleware will log all requests as they go through. They are, by default, sent to a logger named 'wsgi' at the INFO level.

If setup_console_handler is true, then messages for the named logger will be sent to the console.

format = '%(REMOTE_ADDR)s - %(REMOTE_USER)s [%(time)s] "%(REQUEST_METHOD)s %(REQUEST_URI)

write_log(*environ*, *method*, *req_uri*, *start*, *status*, *bytes*)

galaxy.web.framework.middleware.translogger.**make_filter**(*app*, *global_conf*, *logger_name='wsgi'*, *format=None*, *logging_level=20*, *setup_console_handler=True*, *set_logger_level=10*)

This logging middleware will log all requests as they go through. They are, by default, sent to a logger named 'wsgi' at the INFO level.

If setup_console_handler is true, then messages for the named logger will be sent to the console.

xforwardedhost Module

class galaxy.web.framework.middleware.xforwardedhost.**XForwardedHostMiddleware**(*app*, *global_conf=None*)

Bases: object

A WSGI middleware that changes the HTTP host header in the WSGI environ based on the X-Forwarded-Host header IF found

security Package

security Package

class galaxy.web.security.**SecurityHelper**(***config*)

Bases: object

decode_guid(*session_key*)

decode_id(*obj_id*, *kind=None*)

encode_all_ids(*rval*, *recursive=False*)

Encodes all integer values in the dict rval whose keys are 'id' or end with '_id' excluding *tool_id* which are consumed and produced as is via the API.

encode_dict_ids(*a_dict*, *kind=None*)

Encode all ids in dictionary. Ids are identified by (a) an 'id' key or (b) a key that ends with '_id'

encode_guid(*session_key*)

encode_id(*obj_id*, *kind=None*)

get_new_guid()

galaxy.web.security.**get_random_bytes**(*nbytes*)

webapps Package

webapps Package Galaxy webapps root package – this is a namespace package.

Subpackages

community Package

community Package

app Module

buildapp Module

config Module

Subpackages

controllers Package

controllers Package

admin Module

common Module

`hg` Module

`repository` Module

`repository_review` Module

`upload` Module

`user` Module

framework Package

`framework` Package

Subpackages

middleware Package

`middleware` Package

`hg` Module

`remoteuser` Module

model Package

`model` Package

`mapping` Module

Subpackages

migrate Package

`check` Module

security Package

`security` Package

util Package

`container_util` Module

`hgweb_config` Module

`shed_statistics` Module

`workflow_util` Module

galaxy Package

`buildapp` Module

Subpackages

Galaxy API Documentation

Background In addition to being accessible through a web interface, Galaxy can also be accessed programmatically, through shell scripts and other programs. The web interface is appropriate for things like exploratory analysis, visualization, construction of workflows, and rerunning workflows on new datasets.

The web interface is less suitable for things like

- Connecting a Galaxy instance directly to your sequencer and running workflows whenever data is ready.
- Running a workflow against multiple datasets (which can be done with the web interface, but is tedious).
- When the analysis involves complex control, such as looping and branching.

The Galaxy API addresses these and other situations by exposing Galaxy internals through an additional interface, known as an Application Programming Interface, or API.

Quickstart Log in as your user, navigate to the API Keys page in the User menu, and generate a new API key. Make a note of the API key, and then pull up a terminal. Now we'll use the display.py script in your galaxy/scripts/api directory for a short example:

```
% ./display.py my_key http://localhost:4096/api/histories
Collection Members
------------------
#1: /api/histories/8c49be448cfe29bc
  name: Unnamed history
  id: 8c49be448cfe29bc
#2: /api/histories/33b43b4e7093c91f
  name: output test
  id: 33b43b4e7093c91f
```

The result is a Collection of the histories of the user specified by the API key (you). To look at the details of a particular history, say #1 above, do the following:

```
% ./display.py my_key http://localhost:4096/api/histories/8c49be448cfe29bc
Member Information
------------------
state_details: {'ok': 1, 'failed_metadata': 0, 'upload': 0, 'discarded': 0, 'running': 0, 'setting_me
state: ok
contents_url: /api/histories/8c49be448cfe29bc/contents
id: 8c49be448cfe29bc
name: Unnamed history
```

This gives detailed information about the specific member in question, in this case the History. To view history contents, do the following:

```
% ./display.py my_key http://localhost:4096/api/histories/8c49be448cfe29bc/contents
Collection Members
------------------
#1: /api/histories/8c49be448cfe29bc/contents/6f91353f3eb0fa4a
  name: Pasted Entry
  type: file
  id: 6f91353f3eb0fa4a
```

What we have here is another Collection of items containing all of the datasets in this particular history. Finally, to view details of a particular dataset in this collection, execute the following:

```
% ./display.py my_key http://localhost:4096/api/histories/8c49be448cfe29bc/contents/6f91353f3eb0fa4a
Member Information
------------------
misc_blurb: 1 line
name: Pasted Entry
data_type: txt
deleted: False
file_name: /Users/yoplait/work/galaxy-stock/database/files/000/dataset_82.dat
state: ok
download_url: /datasets/6f91353f3eb0fa4a/display?to_ext=txt
visible: True
genome_build: ?
model_class: HistoryDatasetAssociation
file_size: 17
metadata_data_lines: 1
id: 6f91353f3eb0fa4a
misc_info: uploaded txt file
metadata_dbkey: ?
```

And now you've successfully used the API to request and select a history, browse the contents of that history, and then look at detailed information about a particular dataset.

For a more comprehensive Data Library example, set the following option in your galaxy.ini as well, and restart galaxy again:

```
admin_users = you@example.org
library_import_dir = /path/to/some/directory
```

In the directory you specified for 'library_import_dir', create some subdirectories, and put (or symlink) files to import into Galaxy into those subdirectories.

In Galaxy, create an account that matches the address you put in 'admin_users', then browse to that user's preferences and generate a new API Key. Copy the key to your clipboard and then use these scripts:

```
% ./display.py my_key http://localhost:4096/api/libraries
Collection Members
------------------

0 elements in collection

% ./library_create_library.py my_key http://localhost:4096/api/libraries api_test 'API Test Library'
Response
--------
/api/libraries/f3f73e481f432006
  name: api_test
  id: f3f73e481f432006

% ./display.py my_key http://localhost:4096/api/libraries
Collection Members
------------------
/api/libraries/f3f73e481f432006
  name: api_test
  id: f3f73e481f432006

% ./display.py my_key http://localhost:4096/api/libraries/f3f73e481f432006
Member Information
------------------
synopsis: None
contents_url: /api/libraries/f3f73e481f432006/contents
description: API Test Library
name: api_test

% ./display.py my_key http://localhost:4096/api/libraries/f3f73e481f432006/contents
Collection Members
------------------
/api/libraries/f3f73e481f432006/contents/28202595c0d2591f61ddda595d2c3670
  name: /
  type: folder
  id: 28202595c0d2591f61ddda595d2c3670

% ./library_create_folder.py my_key http://localhost:4096/api/libraries/f3f73e481f432006/contents 282
Response
--------
/api/libraries/f3f73e481f432006/contents/28202595c0d2591fa4f9089d2303fd89
  name: api_test_folder1
  id: 28202595c0d2591fa4f9089d2303fd89

% ./library_upload_from_import_dir.py my_key http://localhost:4096/api/libraries/f3f73e481f432006/con
Response
--------
/api/libraries/f3f73e481f432006/contents/e9ef7fdb2db87d7b
  name: 2.bed
  id: e9ef7fdb2db87d7b
/api/libraries/f3f73e481f432006/contents/3b7f6a31f80a5018
  name: 3.bed
  id: 3b7f6a31f80a5018

% ./display.py my_key http://localhost:4096/api/libraries/f3f73e481f432006/contents
Collection Members
------------------
/api/libraries/f3f73e481f432006/contents/28202595c0d2591f61ddda595d2c3670
  name: /
```

```
  type: folder
  id: 28202595c0d2591f61ddda595d2c3670
/api/libraries/f3f73e481f432006/contents/28202595c0d2591fa4f9089d2303fd89
  name: /api_test_folder1
  type: folder
  id: 28202595c0d2591fa4f9089d2303fd89
/api/libraries/f3f73e481f432006/contents/e9ef7fdb2db87d7b
  name: /api_test_folder1/2.bed
  type: file
  id: e9ef7fdb2db87d7b
/api/libraries/f3f73e481f432006/contents/3b7f6a31f80a5018
  name: /api_test_folder1/3.bed
  type: file
  id: 3b7f6a31f80a5018

% ./display.py my_key http://localhost:4096/api/libraries/f3f73e481f432006/contents/e9ef7fdb2db87d7b
Member Information
------------------
misc_blurb: 68 regions
metadata_endCol: 3
data_type: bed
metadata_columns: 6
metadata_nameCol: 4
uploaded_by: nate@...
metadata_strandCol: 6
name: 2.bed
genome_build: hg19
metadata_comment_lines: None
metadata_startCol: 2
metadata_chromCol: 1
file_size: 4272
metadata_data_lines: 68
message:
metadata_dbkey: hg19
misc_info: uploaded bed file
date_uploaded: 2010-06-22T17:01:51.266119
metadata_column_types: str, int, int, str, int, str
```

Other parameters are valid when uploading, they are the same parameters as are used in the web form, like 'link_data_only' and etc.

The request and response format should be considered alpha and are subject to change.

API Design Guidelines The following section outlines guidelines related to extending and/or modifing the Galaxy API. The Galaxy API has grown in an ad-hoc fashion over time by many contributors and so clients SHOULD NOT expect the API will conform to these guidelines - but developers contributing to the Galaxy API SHOULD follow these guidelines.

- API functionality should include docstring documentation for consumption by readthedocs.org.
- Developers should familiarize themselves with the HTTP status code definitions http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html. The API responses should properly set the status code according to the result - in particular 2XX responses should be used for successful requests, 4XX for various kinds of client errors, and 5XX for the errors on the server side.
- If there is an error processing some part of request (one item in a list for instance), the status code should be set to reflect the error and the partial result may or may not be returned depending on the controller - this behavior should be documented.

- API methods should throw a finite number of exceptions (defined in exceptions Package) and these should subclass *MessageException* and not paste/wsgi HTTP exceptions. When possible, the framework itself should be responsible catching these exceptions, setting the status code, and building an error response.
- Error responses should not consist of plain text strings - they should be dictionaries describing the error and containing the following:

```
{
  "status_code": 400,
  "err_code": 400007,
  "err_msg": "Request contained invalid parameter, action could not be completed.",
  "type": "error",
  "extra_error_info": "Extra information."
}
```

  Various error conditions (once a format has been chosen and framework to enforce it in place) should be spelled out in this document.
- Backward compatibility is important and should be maintained when possible. If changing behavior in a non-backward compatibile way please ensure one of the following holds - there is a strong reason to believe no consumers depend on a behavior, the behavior is effectively broken, or the API method being modified has not been part of a tagged dist release.

The following bullet points represent good practices more than guidelines, please consider them when modifying the API.

- Functionality should not be copied and pasted between controllers - consider refactoring functionality into associated classes or short of that into Mixins (http://en.wikipedia.org/wiki/Composition_over_inheritance) or into Managers (managers Package).
- API additions are more permanent changes to Galaxy than many other potential changes and so a second opinion on API changes should be sought. (Consider a pull request!)
- New API functionality should include functional tests. These functional tests should be implemented in Python and placed in *test/functional/api*. (Once such a framework is in place - it is not right now).
- Changes to reflect modifications to the API should be pushed upstream to the BioBlend project if possible.

Longer term goals/notes.

- It would be advantageous to have a clearer separation of anonymous and admin handling functionality.
- If at some point in the future, functionality needs to be added that breaks backward compatibility in a significant way to a component used by the community - a "dev" variant of the API will be established and the community should be alerted and given a timeframe for when the old behavior will be replaced with the new behavior.
- Consistent standards for range-based requests, batch requests, filtered requests, etc... should be established and documented here.

API Controllers Galaxy offers the following API controllers:

`annotations` Module API operations on annotations.

class `galaxy.webapps.galaxy.api.annotations.`**`BaseAnnotationsController`**(*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesStored*
galaxy.model.item_attrs.UsesAnnotations

`create`(*trans*, **args*, ***kwargs*)

`delete`(*trans*, **args*, ***kwargs*)

index (*trans*, **args*, ***kwargs*)

undelete (*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.api.annotations.**HistoryAnnotationsController** (*app*)
Bases: *galaxy.webapps.galaxy.api.annotations.BaseAnnotationsController*

controller_name = 'history_annotations'

tagged_item_id = 'history_id'

class galaxy.webapps.galaxy.api.annotations.**HistoryContentAnnotationsController** (*app*)
Bases: *galaxy.webapps.galaxy.api.annotations.BaseAnnotationsController*

controller_name = 'history_content_annotations'

tagged_item_id = 'history_content_id'

class galaxy.webapps.galaxy.api.annotations.**WorkflowAnnotationsController** (*app*)
Bases: *galaxy.webapps.galaxy.api.annotations.BaseAnnotationsController*

controller_name = 'workflow_annotations'

tagged_item_id = 'workflow_id'

authenticate Module API key retrieval through BaseAuth Sample usage:

curl –user zipzap@foo.com:password http://localhost:8080/api/authenticate/baseauth

Returns:

{ "api_key": "baa4d6e3a156d3033f05736255f195f9"

}

class galaxy.webapps.galaxy.api.authenticate.**AuthenticationController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

get_api_key (*trans*, **args*, ***kwargs*)
def get_api_key(self, trans, **kwd) * GET /api/authenticate/baseauth

returns an API key for authenticated user based on BaseAuth headers

Returns api_key in json format

Return type dict

Raises ObjectNotFound, HTTPBadRequest

configuration Module API operations allowing clients to determine Galaxy instance's capabilities and configuration settings.

class galaxy.webapps.galaxy.api.configuration.**ConfigurationController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

dynamic_tool_confs (*trans*, **args*, ***kwargs*)

get_config_dict (*trans*, *return_admin=False*, *view=None*, *keys=None*, *default_view='all'*)
Return a dictionary with (a subset of) current Galaxy settings.

If *return_admin* also include a subset of more sensitive keys. Pass in *view* (String) and comma seperated list of keys to control which configuration settings are returned.

index (*trans*, **args*, ***kwargs*)
GET /api/configuration Return an object containing exposable configuration settings.

Note: a more complete list is returned if the user is an admin.

tool_lineages (*trans*, **args*, ***kwargs*)

version (*trans*, **args*, ***kwargs*)
GET /api/version Return a description of the major version of Galaxy (e.g. 15.03).

Return type dict

Returns dictionary with major version keyed on 'version_major'

dataset_collections Module

class galaxy.webapps.galaxy.api.dataset_collections.**DatasetCollectionsController** (*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesLibrar

create (*trans*, **args*, ***kwargs*)

- **POST /api/dataset_collections:** create a new dataset collection instance.

Parameters **payload** (*dict*) – (optional) dictionary structure containing: * collection_type: dataset colltion type to create. * instance_type: Instance type - 'history' or 'library'. * name: the new dataset collections's name * datasets: object describing datasets for collection

Return type dict

Returns element view of new dataset collection

index (*trans*, **args*, ***kwargs*)

show (*trans*, **args*, ***kwargs*)

datasets Module

datatypes Module

API operations allowing clients to determine datatype supported by Galaxy.

class galaxy.webapps.galaxy.api.datatypes.**DatatypesController** (*app*)
Bases: galaxy.web.base.controller.BaseAPIController

converters (*trans*, **args*, ***kwargs*)

index (*trans*, **args*, ***kwargs*)
GET /api/datatypes Return an object containing upload datatypes.

mapping (*trans*, **args*, ***kwargs*)
GET /api/datatypes/mapping Return a dictionary of class to class mappings.

sniffers (*trans*, **args*, ***kwargs*)
GET /api/datatypes/sniffers Return a list of sniffers.

extended_metadata Module API operations on annotations.

class galaxy.webapps.galaxy.api.extended_metadata.**BaseExtendedMetadataController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesExtend
galaxy.web.base.controller.UsesLibraryMixinItems, galaxy.web.base.controller.UsesStoredW

create(*trans*, **args*, ***kwargs*)

delete(*trans*, **args*, ***kwargs*)

index(*trans*, **args*, ***kwargs*)

undelete(*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.api.extended_metadata.**HistoryDatasetExtendMetadataController**(*app*)
Bases: galaxy.webapps.galaxy.api.extended_metadata.BaseExtendedMetadataController

controller_name = 'history_dataset_extended_metadata'

exmeta_item_id = 'history_content_id'

class galaxy.webapps.galaxy.api.extended_metadata.**LibraryDatasetExtendMetadataController**(*app*)
Bases: galaxy.webapps.galaxy.api.extended_metadata.BaseExtendedMetadataController

controller_name = 'library_dataset_extended_metadata'

exmeta_item_id = 'library_content_id'

folder_contents Module API operations on the contents of a library folder.

class galaxy.webapps.galaxy.api.folder_contents.**FolderContentsController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesLibrar
galaxy.web.base.controller.UsesLibraryMixinItems

Class controls retrieval, creation and updating of folder contents.

build_path(*trans*, *folder*)
Search the path upwards recursively and load the whole route of names and ids for breadcrumb building purposes.

Parameters

- **folder** – current folder for navigating up
- **type** – Galaxy LibraryFolder

Returns list consisting of full path to the library

Type list

create(*self*, *trans*, *library_id*, *payload*, ***kwd*)

•**POST /api/folders/{encoded_id}/contents** create a new library file from an HDA

Parameters **payload** – dictionary structure containing:

Returns a dictionary containing the id, name, and 'show' url of the new item

Return type dict

Raises ObjectAttributeInvalidException, InsufficientPermissionsException, ItemAccessibilityException, InternalServerError

index (*trans*, **args*, ***kwargs*)
GET /api/folders/{encoded_folder_id}/contents

Displays a collection (list) of a folder's contents (files and folders). Encoded folder ID is prepended with 'F' if it is a folder as opposed to a data set which does not have it. Full path is provided in response as a separate object providing data for breadcrumb path building.

Parameters

- **folder_id** (*encoded string*) – encoded ID of the folder which contents should be library_dataset_dict
- **kwd** (*dict*) – keyword dictionary with other params

Returns dictionary containing all items and metadata

Type dict

Raises MalformedId, InconsistentDatabase, ObjectNotFound, InternalServerError

show (*trans*, **args*, ***kwargs*)
GET /api/folders/{encoded_folder_id}/

update (*trans*, **args*, ***kwargs*)
PUT /api/folders/{encoded_folder_id}/contents

folders Module API operations on library folders.

class galaxy.webapps.galaxy.api.folders.**FoldersController** (*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesLibra galaxy.web.base.controller.UsesLibraryMixinItems

create (*self*, *trans*, *encoded_parent_folder_id*, ***kwd*)
*POST /api/folders/{encoded_parent_folder_id}

Create a new folder object underneath the one specified in the parameters.

Parameters

- **encoded_parent_folder_id** (*an encoded id string (should be prefixed by 'F')*) – the parent folder's id (required)
- **name** (*str*) – the name of the new folder (required)
- **description** (*str*) – the description of the new folder

Returns information about newly created folder, notably including ID

Return type dictionary

Raises RequestParameterMissingException

delete (*self*, *trans*, *id*, ***kwd*)

- **DELETE /api/folders/{id}** marks the folder with the given `id` as *deleted* (or removes the *deleted* mark if the *undelete* param is true)

Note: Currently, only admin users can un/delete folders.

Parameters

- **id** (*an encoded id string*) – the encoded id of the folder to un/delete

- **undelete** (bool) – (optional) flag specifying whether the item should be deleted or undeleted, defaults to false:

Returns detailed folder information

Return type dictionary

Raises ItemAccessibilityException, MalformedId, ObjectNotFound

get_permissions (*trans*, **args*, ***kwargs*)

•GET /api/folders/{id}/permissions

Load all permissions for the given folder id and return it.

Parameters

- **encoded_folder_id** (*an encoded id string*) – the encoded id of the folder
- **scope** (*string*) – either 'current' or 'available'

Returns dictionary with all applicable permissions' values

Return type dictionary

Raises ObjectNotFound, InsufficientPermissionsException

index (*trans*, **args*, ***kwargs*)

*GET /api/folders/ This would normally display a list of folders. However, that would be across multiple libraries, so it's not implemented.

set_permissions (*trans*, **args*, ***kwargs*)

**def set_permissions(self, trans, encoded_folder_id, **kwd): ***POST /api/folders/{encoded_folder_id}/permissions

Parameters

- **encoded_folder_id** (*an encoded id string*) – the encoded id of the folder to set the permissions of
- **action** (*string*) – (required) describes what action should be performed available actions: set_permissions
- **add_ids[]** (*string or list*) – list of Role.id defining roles that should have add item permission on the folder
- **manage_ids[]** (*string or list*) – list of Role.id defining roles that should have manage permission on the folder
- **modify_ids[]** (*string or list*) – list of Role.id defining roles that should have modify permission on the folder

Return type dictionary

Returns dict of current roles for all available permission types.

Raises RequestParameterInvalidException, ObjectNotFound, InsufficientPermissionsException, InternalServerError RequestParameterMissingException

show (*self*, *trans*, *id*, ***kwd*)

*GET /api/folders/{encoded_folder_id}

Displays information about a folder.

Parameters **id** (*an encoded id string (has to be prefixed by 'F')*) – the folder's encoded id (required)

Returns dictionary including details of the folder

Return type dict

update(*trans*, **args*, ***kwargs*)
PUT /api/folders/{encoded_folder_id}

forms Module API operations on FormDefinition objects.

class galaxy.webapps.galaxy.api.forms.**FormDefinitionAPIController**(*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

create(*trans*, **args*, ***kwargs*)
POST /api/forms Creates a new form.

index(*trans*, **args*, ***kwargs*)
GET /api/forms Displays a collection (list) of forms.

show(*trans*, **args*, ***kwargs*)
GET /api/forms/{encoded_form_id} Displays information about a form.

ftp_files Module

genomes Module
class galaxy.webapps.galaxy.api.genomes.**GenomesController**(*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

RESTful controller for interactions with genome data.

index(*trans*, **args*, ***kwargs*)
GET /api/genomes: returns a list of installed genomes

show(*trans*, **args*, ***kwargs*)
GET /api/genomes/{id}

Returns information about build <id>
galaxy.webapps.galaxy.api.genomes.**get_id**(*base*, *format*)

group_roles Module API operations on Group objects.

class galaxy.webapps.galaxy.api.group_roles.**GroupRolesAPIController**(*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

delete(*trans*, **args*, ***kwargs*)
DELETE /api/groups/{encoded_group_id}/roles/{encoded_role_id} Removes a role from a group

index(*trans*, **args*, ***kwargs*)
GET /api/groups/{encoded_group_id}/roles Displays a collection (list) of groups.

show(*trans*, **args*, ***kwargs*)
GET /api/groups/{encoded_group_id}/roles/{encoded_role_id} Displays information about a group role.

update(*trans*, **args*, ***kwargs*)
PUT /api/groups/{encoded_group_id}/roles/{encoded_role_id} Adds a role to a group

group_users Module API operations on Group objects.

class galaxy.webapps.galaxy.api.group_users.**GroupUsersAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

delete(*trans*, **args*, ***kwargs*)
DELETE /api/groups/{encoded_group_id}/users/{encoded_user_id} Removes a user from a group

index(*trans*, **args*, ***kwargs*)
GET /api/groups/{encoded_group_id}/users Displays a collection (list) of groups.

show(*trans*, **args*, ***kwargs*)
GET /api/groups/{encoded_group_id}/users/{encoded_user_id} Displays information about a group user.

update(*trans*, **args*, ***kwargs*)
PUT /api/groups/{encoded_group_id}/users/{encoded_user_id} Adds a user to a group

groups Module API operations on Group objects.

class galaxy.webapps.galaxy.api.groups.**GroupAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

create(*trans*, **args*, ***kwargs*)
POST /api/groups Creates a new group.

index(*trans*, **args*, ***kwargs*)
GET /api/groups Displays a collection (list) of groups.

show(*trans*, **args*, ***kwargs*)
GET /api/groups/{encoded_group_id} Displays information about a group.

update(*trans*, **args*, ***kwargs*)
PUT /api/groups/{encoded_group_id} Modifies a group.

histories Module API operations on a history.

See also:

galaxy.model.History

class galaxy.webapps.galaxy.api.histories.**HistoriesController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.ExportsHis
galaxy.web.base.controller.ImportsHistoryMixin

archive_download(*trans*, **args*, ***kwargs*)
export_download(self, trans, id, jeha_id) * GET /api/histories/{id}/exports/{jeha_id}:

If ready and available, return raw contents of exported history. Use/poll "PUT /api/histories/{id}/exports" to initiate the creation of such an export - when ready that route will return 200 status code (instead of 202) with a JSON dictionary containing a *download_url*.

archive_export(*trans*, **args*, ***kwargs*)
export_archive(self, trans, id, payload) * PUT /api/histories/{id}/exports:

start job (if needed) to create history export for corresponding history.

Parameters **id** (*str*) – the encoded id of the history to export

Return type dict

Returns object containing url to fetch export from.

citations (*trans*, **args*, ***kwargs*)

create (*trans*, *payload*)

•**POST /api/histories:** create a new history

Parameters

- **payload** (*dict*) – (optional) dictionary structure containing: * name: the new history's name * history_id: the id of the history to copy * archive_source: the url that will generate the archive to import * archive_type: 'url' (default)
- **keys** – same as the use of *keys* in the *index* function above
- **view** – same as the use of *view* in the *index* function above

Return type dict

Returns element view of new history

delete (*self*, *trans*, *id*, ***kwd*)

•**DELETE /api/histories/{id}** delete the history with the given `id`

Note: Stops all active jobs in the history if purge is set.

Parameters

- **id** (*str*) – the encoded id of the history to delete
- **kwd** (*dict*) – (optional) dictionary structure containing extra parameters

You can purge a history, removing all it's datasets from disk (if unshared), by passing in `purge=True` in the url.

Parameters

- **keys** – same as the use of *keys* in the *index* function above
- **view** – same as the use of *view* in the *index* function above

Return type dict

Returns the deleted or purged history

index (*trans*, *deleted='False'*)

•**GET /api/histories:** return undeleted histories for the current user

•**GET /api/histories/deleted:** return deleted histories for the current user

Note: Anonymous users are allowed to get their current history

Parameters **deleted** (*boolean*) – if True, show only deleted histories, if False, non-deleted

Return type *list*

Returns list of dictionaries containing summary history information

The following are optional parameters:

view: string, one of ('summary','detailed'), defaults to 'summary' controls which set of properties to return

keys: comma separated strings, unused by default keys/names of individual properties to return

If neither keys or views are sent, the default view (set of keys) is returned. If both a view and keys are sent, the key list and the view's keys are combined. If keys are send and no view, only those properties in keys are returned.

For which properties are available see: galaxy/managers/histories/HistorySerializer

The list returned can be filtered by using two optional parameters:

q: string, generally a property name to filter by followed by an (often optional) hyphen and operator string.

qv: string, the value to filter by

..example: To filter the list to only those created after 2015-01-29, the query string would look like:

'?q=create_time-gt&qv=2015-01-29'

Multiple filters can be sent in using multiple q/qv pairs: '?q=create_time-gt&qv=2015-01-29&q=tag-has&qv=experiment-1'

The list returned can be paginated using two optional parameters:

limit: integer, defaults to no value and no limit (return all) how many items to return

offset: integer, defaults to 0 and starts at the beginning skip the first (offset - 1) items and begin returning at the Nth item

..example:

limit and offset can be combined. Skip the first two and return five: '?limit=5&offset=3'

show (*trans*, *id*, *deleted='False'*)

•**GET /api/histories/{id}:** return the history with `id`

•**GET /api/histories/deleted/{id}:** return the deleted history with `id`

•**GET /api/histories/most_recently_used:** return the most recently used history

Parameters

- **id** (*an encoded id string*) – the encoded id of the history to query or the string 'most_recently_used'
- **deleted** (*boolean*) – if True, allow information on a deleted history to be shown.
- **keys** – same as the use of *keys* in the *index* function above
- **view** – same as the use of *view* in the *index* function above

Return type dictionary

Returns detailed history information

undelete (*self*, *trans*, *id*, ***kwd*)

•**POST /api/histories/deleted/{id}/undelete:** undelete history (that hasn't been purged) with the given `id`

Parameters

- **id** (*str*) – the encoded id of the history to undelete

- **keys** – same as the use of *keys* in the *index* function above
- **view** – same as the use of *view* in the *index* function above

Return type str

Returns 'OK' if the history was undeleted

update (*self*, *trans*, *id*, *payload*, ***kwd*)

•**PUT /api/histories/{id}** updates the values for the history with the given `id`

Parameters

- **id** (*str*) – the encoded id of the history to update
- **payload** (*dict*) – a dictionary containing any or all the fields in `galaxy.model.History.to_dict()` and/or the following:
  - annotation: an annotation for the history
- **keys** – same as the use of *keys* in the *index* function above
- **view** – same as the use of *view* in the *index* function above

Return type dict

Returns an error object if an error occurred or a dictionary containing any values that were different from the original and, therefore, updated

history_contents Module API operations on the contents of a history.

class `galaxy.webapps.galaxy.api.history_contents.`**HistoryContentsController** (*app*)

Bases: `galaxy.web.base.controller.BaseAPIController`, `galaxy.web.base.controller.UsesLibra`
`galaxy.web.base.controller.UsesLibraryMixinItems`, `galaxy.web.base.controller.UsesTagsMi`

create (*self*, *trans*, *history_id*, *payload*, ***kwd*)

•**POST /api/histories/{history_id}/contents/{type}** create a new HDA by copying an accessible LibraryDataset

Parameters

- **history_id** (*str*) – encoded id string of the new HDA's History
- **type** (*str*) – Type of history content - 'dataset' (default) or 'dataset_collection'.
- **payload** (*dict*) – dictionary structure containing:: copy from library (for type 'dataset'): 'source' = 'library' 'content' = [the encoded id from the library dataset]

  copy from history dataset (for type 'dataset'): 'source' = 'hda' 'content' = [the encoded id from the HDA]

  copy from history dataset collection (for type 'dataset_collection') 'source' = 'hdca' 'content' = [the encoded id from the HDCA]

  create new history dataset collection (for type 'dataset_collection') 'source' = 'new_collection' (default 'source' if type is

  'dataset_collection' - no need to specify this)

'collection_type' = For example, "list", "paired", "list:paired". 'name' = Name of new dataset collection. 'element_identifiers' = Recursive list structure defining collection.

Each element must have 'src' which can be 'hda', 'ldda', 'hdca', or 'new_collection', as well as a 'name' which is the name of element (e.g. "forward" or "reverse" for paired datasets, or arbitrary sample names for instance for lists). For all src's except 'new_collection' - a encoded 'id' attribute must be included wiht element as well. 'new_collection' sources must defined a 'collection_type' and their own list of (potentially) nested 'element_identifiers'.

..note: Currently, a user can only copy an HDA from a history that the user owns.

Return type dict

Returns dictionary containing detailed information for the new HDA

delete (*self*, *trans*, *history_id*, *id*, ***kwd*)

•**DELETE /api/histories/{history_id}/contents/{id}** delete the HDA with the given `id`

Note: Currently does not stop any active jobs for which this dataset is an output.

Parameters

- **id** (*str*) – the encoded id of the history to delete
- **purge** (bool) – if True, purge the HDA
- **kwd** (*dict*) – (optional) dictionary structure containing:
  - **payload: a dictionary itself containing:**
    - purge: if True, purge the HDA

Note: that payload optionally can be placed in the query string of the request. This allows clients that strip the request body to still purge the dataset.

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

index (*self*, *trans*, *history_id*, *ids=None*, ***kwd*)

•**GET /api/histories/{history_id}/contents** return a list of HDA data for the history with the given `id`

Note: Anonymous users are allowed to get their current history contents

If Ids is not given, index returns a list of *summary* objects for every HDA associated with the given *history_id*.

If ids is given, index returns a *more complete* json object for each HDA in the ids list.

Parameters

- **history_id** (*str*) – encoded id string of the HDA's History
- **ids** (*str*) – (optional) a comma separated list of encoded *HDA* ids
- **types** (*str*) – (optional) kinds of contents to index (currently just dataset, but dataset_collection will be added shortly).

Return type *list*

Returns dictionaries containing summary or detailed HDA information

show (*self*, *trans*, *id*, *history_id*, ***kwd*)

•**GET /api/histories/{history_id}/contents/{id}** return detailed information about an HDA within a history

Note: Anonymous users are allowed to get their current history contents

Parameters

- **ids** – the encoded id of the HDA to return
- **history_id** (*str*) – encoded id string of the HDA's History

Return type dict

Returns dictionary containing detailed HDA information

update (*self*, *trans*, *history_id*, *id*, *payload*, ***kwd*)

•**PUT /api/histories/{history_id}/contents/{id}** updates the values for the HDA with the given `id`

Parameters

- **history_id** (*str*) – encoded id string of the HDA's History
- **id** (*str*) – the encoded id of the history to undelete
- **payload** (*dict*) – a dictionary containing any or all the fields in *galaxy.model.HistoryDatasetAssociation.to_dict()* and/or the following:
  - annotation: an annotation for the HDA

Return type dict

Returns an error object if an error occurred or a dictionary containing any values that were different from the original and, therefore, updated

item_tags Module API operations related to tagging items.

class `galaxy.webapps.galaxy.api.item_tags.`**BaseItemTagsController**(*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesTagsMi*

create (*trans*, **args*, ***kwargs*)

delete (*trans*, **args*, ***kwargs*)

index (*trans*, **args*, ***kwargs*)

show (*trans*, **args*, ***kwargs*)

update (*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.api.item_tags.**HistoryContentTagsController**(*app*)
Bases: galaxy.webapps.galaxy.api.item_tags.BaseItemTagsController

controller_name = 'history_content_tags'

tagged_item_class = 'HistoryDatasetAssociation'

tagged_item_id = 'history_content_id'

class galaxy.webapps.galaxy.api.item_tags.**HistoryTagsController**(*app*)
Bases: galaxy.webapps.galaxy.api.item_tags.BaseItemTagsController

controller_name = 'history_tags'

tagged_item_class = 'History'

tagged_item_id = 'history_id'

class galaxy.webapps.galaxy.api.item_tags.**WorkflowTagsController**(*app*)
Bases: galaxy.webapps.galaxy.api.item_tags.BaseItemTagsController

controller_name = 'workflow_tags'

tagged_item_class = 'StoredWorkflow'

tagged_item_id = 'workflow_id'

job_files Module API for asynchronous job running mechanisms can use to fetch or put files related to running and queued jobs.

class galaxy.webapps.galaxy.api.job_files.**JobFilesAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

This job files controller allows remote job running mechanisms to read and modify the current state of files for queued and running jobs. It is certainly not meant to represent part of Galaxy's stable, user facing API.

Furthermore, even if a user key corresponds to the user running the job, it should not be accepted for authorization - this API allows access to low-level unfiltered files and such authorization would break Galaxy's security model for tool execution.

create(*self*, *trans*, *job_id*, *payload*, ***kwargs*)

•**POST /api/jobs/{job_id}/files** Populate an output file (formal dataset, task split part, working directory file (such as those related to metadata)). This should be a multipart post with a 'file' parameter containing the contents of the actual file to create.

Parameters

- **job_id** (*str*) – encoded id string of the job
- **payload** (*dict*) – dictionary structure containing:: 'job_key' = Key authenticating 'path' = Path to file to create.

..note: This API method is intended only for consumption by job runners, not end users.

Return type dict

Returns an okay message

index(*self*, *trans*, *job_id*, ***kwargs*)

•**GET /api/jobs/{job_id}/files** Get a file required to staging a job (proper datasets, extra inputs, task-split inputs, working directory files).

Parameters

- **job_id** (*str*) – encoded id string of the job
- **path** (*str*) – Path to file.
- **job_key** (*str*) – A key used to authenticate this request as acting on behalf or a job runner for the specified job.

..note: This API method is intended only for consumption by job runners, not end users.

Return type *binary*

Returns contents of file

jobs Module API operations on a jobs.

See also:

`galaxy.model.Jobs`

class `galaxy.webapps.galaxy.api.jobs.JobController(app)`

Bases: `galaxy.web.base.controller.BaseAPIController`, `galaxy.web.base.controller.UsesLibrar`

create(*trans*, **args*, ***kwargs*)

See the create method in tools.py in order to submit a job.

index(*trans*, *state=None*, *tool_id=None*, *history_id=None*, *date_range_min=None*, *date_range_max=None*, *user_details=False*)

•GET /api/jobs: return jobs for current user

!! if user is admin and user_details is True, then return jobs for all galaxy users based on filtering - this is an extended service

Parameters **state** (*string or list*) – limit listing of jobs to those that match one of the included states. If none, all are returned.

Valid Galaxy job states include: 'new', 'upload', 'waiting', 'queued', 'running', 'ok', 'error', 'paused', 'deleted', 'deleted_new'

Parameters

- **tool_id** (*string or list*) – limit listing of jobs to those that match one of the included tool_ids. If none, all are returned.
- **user_details** (*boolean*) – if true, and requestor is an admin, will return external job id and user email.
- **date_range_min** (*string '2014-01-01'*) – limit the listing of jobs to those updated on or after requested date
- **date_range_max** (*string '2014-12-31'*) – limit the listing of jobs to those updated on or before requested date
- **history_id** (*string*) – limit listing of jobs to those that match the history_id. If none, all are returned.

Return type *list*

Returns list of dictionaries containing summary job information

inputs (*trans*, **args*, ***kwargs*)
show(trans, id) * GET /api/jobs/{job_id}/inputs

returns input datasets created by job

Parameters **id** (*string*) – Encoded job id

Return type dictionary

Returns dictionary containing input dataset associations

outputs (*trans*, **args*, ***kwargs*)
show(trans, id) * GET /api/jobs/{job_id}/outputs

returns output datasets created by job

Parameters **id** (*string*) – Encoded job id

Return type dictionary

Returns dictionary containing output dataset associations

search (*trans*, *payload*)

•**POST /api/jobs/search:** return jobs for current user

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

Return type *list*

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

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

show (*trans*, *id*)

•**GET /api/jobs/{job_id}:** return jobs for current user

Parameters

- **id** (*string*) – Specific job id
- **full** (*boolean*) – whether to return extra information

Return type dictionary

Returns dictionary containing full description of job data

lda_datasets Module API operations on the library datasets.

class galaxy.webapps.galaxy.api.lda_datasets.**LibraryDatasetsController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesVisual*

delete (*trans*, **args*, ***kwargs*)
delete(self, trans, encoded_dataset_id, **kwd): * DELETE /api/libraries/datasets/{encoded_dataset_id}

Marks the dataset deleted or undeleted based on the value of the undelete flag. If the flag is not present it is considered False and the item is marked deleted.

Parameters **encoded_dataset_id** (*an encoded id string*) – the encoded id of the dataset to change

Returns dict containing information about the dataset

Return type dictionary

download (*self*, *trans*, *format*, ***kwd*)

- GET /api/libraries/datasets/download/{format}
- **POST /api/libraries/datasets/download/{format}** Downloads requested datasets (identified by encoded IDs) in requested format.

example: `GET localhost:8080/api/libraries/datasets/download/tbz?ld_ids%255B%255D=a0d8`

Note: supported format values are: 'zip', 'tgz', 'tbz', 'uncompressed'

Parameters

- **format** (*string*) – string representing requested archive format
- **ld_ids[]** (*an array*) – an array of encoded ids

Return type file

Returns either archive with the requested datasets packed inside or a single uncompressed dataset

Raises MessageException, ItemDeletionException, ItemAccessibilityException, HTTPBadRequest, OSError, IOError, ObjectNotFound

load (*trans*, **args*, ***kwargs*)

load(self, trans, **kwd): * POST /api/libraries/datasets Load dataset from the given source into the library. Source can be:

user directory - root folder specified in galaxy.ini as "$user_library_import_dir"
example path: path/to/galaxy/$user_library_import_dir/user@example.com/{user can browse everything here} the folder with the user login has to be created beforehand

(admin)import directory - root folder specified in galaxy ini as "$library_import_dir"
example path: path/to/galaxy/$library_import_dir/{admin can browse everything here}

(admin)any absolute or relative path - option allowed with "allow_library_path_paste" in galaxy.ini

Parameters

- **encoded_folder_id** (*an encoded id string*) – the encoded id of the folder to import dataset(s) to
- **source** (*str*) – source the datasets should be loaded form
- **link_data** (bool) – flag whether to link the dataset to data or copy it to Galaxy, defaults to copy while linking is set to True all symlinks will be resolved _once_
- **preserve_dirs** (bool) – flag whether to preserve the directory structure when importing dir if False only datasets will be imported

- **file_type** (*str*) – file type of the loaded datasets, defaults to 'auto' (autodetect)
- **dbkey** (*str*) – dbkey of the loaded genome, defaults to '?' (unknown)

Returns dict containing information about the created upload job

Return type dictionary

show (*self*, *trans*, *id*, ***kwd*)

•**GET /api/libraries/datasets/{encoded_dataset_id}:** Displays information about the dataset identified by the encoded ID.

Parameters **id** (*an encoded id string*) – the encoded id of the dataset to query

Returns detailed dataset information from base controller

Return type dictionary

See also:

galaxy.web.base.controller.UsesLibraryMixinItems.get_library_dataset

show_roles (*trans*, **args*, ***kwargs*)

show_roles(self, trans, id, **kwd): * GET /api/libraries/datasets/{encoded_dataset_id}/permissions

Displays information about current or available roles for a given dataset permission.

Parameters

- **encoded_dataset_id** (*an encoded id string*) – the encoded id of the dataset to query
- **scope** (*string*) – either 'current' or 'available'

Return type dictionary

Returns either dict of current roles for all permission types or dict of available roles to choose from (is the same for any permission type)

show_version (*trans*, **args*, ***kwargs*)

show_version(self, trans, encoded_dataset_id, encoded_ldda_id, **kwd): * GET /api/libraries/datasets/:encoded_dataset_id/versions/:encoded_ldda_id

Displays information about specific version of the library_dataset (i.e. ldda).

Parameters

- **encoded_dataset_id** (*an encoded id string*) – the encoded id of the dataset to query
- **encoded_ldda_id** (*an encoded id string*) – the encoded id of the ldda to query

Return type dictionary

Returns dict of ldda's details

update_permissions (*trans*, **args*, ***kwargs*)

def update(self, trans, encoded_dataset_id, **kwd): *POST /api/libraries/datasets/{encoded_dataset_id}/permissions

Parameters

- **encoded_dataset_id** (*an encoded id string*) – the encoded id of the dataset to update permissions of
- **action** (*string*) – (required) describes what action should be performed available actions: make_private, remove_restrictions, set_permissions
- **access_ids[]** (*string or list*) – list of Role.name defining roles that should have access permission on the dataset
- **manage_ids[]** (*string or list*) – list of Role.name defining roles that should have manage permission on the dataset
- **modify_ids[]** (*string or list*) – list of Role.name defining roles that should have modify permission on the library dataset item

Return type dictionary

Returns dict of current roles for all available permission types

Raises RequestParameterInvalidException, ObjectNotFound, InsufficientPermissionsException, InternalServerError RequestParameterMissingException

libraries Module API operations on a data library.

class galaxy.webapps.galaxy.api.libraries.**LibrariesController**(*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*

create(*self*, *trans*, *payload*, ***kwd*)

•**POST /api/libraries:** Creates a new library. Only `name` parameter is required.

Note: Currently, only admin users can create libraries.

Parameters **payload** (*dict*) – dictionary structure containing:: 'name': the new library's name (required) 'description': the new library's description (optional) 'synopsis': the new library's synopsis (optional)

Returns detailed library information

Return type dict

Raises ItemAccessibilityException, RequestParameterMissingException

delete(*self*, *trans*, *id*, ***kwd*)

•**DELETE /api/libraries/{id}** marks the library with the given `id` as *deleted* (or removes the *deleted* mark if the *undelete* param is true)

Note: Currently, only admin users can un/delete libraries.

Parameters

- **id** (*an encoded id string*) – the encoded id of the library to un/delete
- **undelete** (bool) – (optional) flag specifying whether the item should be deleted or undeleted, defaults to false:

Returns detailed library information

Return type dictionary

See also:

galaxy.model.Library.dict_element_visible_keys

Raises ItemAccessibilityException, MalformedId, ObjectNotFound

get_permissions(*trans*, **args*, ***kwargs*)

•GET /api/libraries/{id}/permissions

Load all permissions for the given library id and return it.

Parameters

- **encoded_library_id** (*an encoded id string*) – the encoded id of the library
- **scope** (*string*) – either 'current' or 'available'
- **is_library_access** (bool) – indicates whether the roles available for the library access are requested

Returns dictionary with all applicable permissions' values

Return type dictionary

Raises ObjectNotFound, InsufficientPermissionsException

index(*self*, *trans*, ***kwd*)

•**GET /api/libraries:** Returns a list of summary data for all libraries.

Parameters **deleted** (*boolean (optional)*) – if True, show only `deleted` libraries, if False show only `non-deleted`

Returns list of dictionaries containing library information

Return type *list*

See also:

galaxy.model.Library.dict_collection_visible_keys

set_permissions(*trans*, **args*, ***kwargs*)

def set_permissions(self, trans, encoded_dataset_id, **kwd): *POST /api/libraries/{encoded_library_id}/permissions

Parameters

- **encoded_library_id** (*an encoded id string*) – the encoded id of the library to set the permissions of
- **action** (*string*) – (required) describes what action should be performed available actions: remove_restrictions, set_permissions
- **access_ids[]** (*string or list*) – list of Role.id defining roles that should have access permission on the library
- **add_ids[]** (*string or list*) – list of Role.id defining roles that should have add item permission on the library
- **manage_ids[]** (*string or list*) – list of Role.id defining roles that should have manage permission on the library

- **modify_ids[]** (*string or list*) – list of Role.id defining roles that should have modify permission on the library

Return type dictionary

Returns dict of current roles for all available permission types

Raises RequestParameterInvalidException, ObjectNotFound, InsufficientPermissionsException, InternalServerError RequestParameterMissingException

set_permissions_old(*trans*, *library*, *payload*, ***kwd*)

*** old implementation for backward compatibility ***

POST /api/libraries/{encoded_library_id}/permissions Updates the library permissions.

show(*self*, *trans*, *id*, *deleted='False'*, ***kwd*)

•**GET /api/libraries/{encoded_id}:** returns detailed information about a library

•**GET /api/libraries/deleted/{encoded_id}:** returns detailed information about a `deleted` library

Parameters

- **id** (*an encoded id string*) – the encoded id of the library
- **deleted** (*boolean*) – if True, allow information on a `deleted` library

Returns detailed library information

Return type dictionary

See also:

galaxy.model.Library.dict_element_visible_keys

Raises MalformedId, ObjectNotFound

update(*trans*, **args*, ***kwargs*)

•**PATCH /api/libraries/{encoded_id}** Updates the library defined by an `encoded_id` with the data in the payload.

Note: Currently, only admin users can update libraries. Also the library must not be *deleted*.

param id the encoded id of the library

type id an encoded id string

param payload (required) dictionary structure containing:: 'name': new library's name, cannot be empty 'description': new library's description 'synopsis': new library's synopsis

type payload dict

returns detailed library information

rtype dict

raises ItemAccessibilityException, MalformedId, ObjectNotFound, RequestParameterInvalidException, RequestParameterMissingException

---

library_contents Module API operations on the contents of a data library.

class galaxy.webapps.galaxy.api.library_contents.**LibraryContentsController**(*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesLibra*
galaxy.web.base.controller.UsesLibraryMixinItems

create(*self*, *trans*, *library_id*, *payload*, ***kwd*)

•**POST /api/libraries/{library_id}/contents:** create a new library file or folder

To copy an HDA into a library send `create_type` of 'file' and the HDA's encoded id in `from_hda_id` (and optionally `ldda_message`).

Parameters

- **library_id** (*str*) – the encoded id of the library where to create the new item
- **payload** (*dict*) – dictionary structure containing:
  - folder_id: the encoded id of the parent folder of the new item
  - create_type: the type of item to create ('file', 'folder' or 'collection')
  - **from_hda_id: (optional, only if create_type is 'file') the** encoded id of an accessible HDA to copy into the library
  - ldda_message: (optional) the new message attribute of the LDDA created
  - **extended_metadata: (optional) dub-dictionary containing any extended** metadata to associate with the item
  - upload_option: (optional) one of 'upload_file' (default), 'upload_directory' or 'upload_paths'
  - **server_dir: (optional, only if upload_option is** 'upload_directory') relative path of the subdirectory of Galaxy `library_import_dir` to upload. All and only the files (i.e. no subdirectories) contained in the specified directory will be uploaded.
  - **filesystem_paths: (optional, only if upload_option is** 'upload_paths' and the user is an admin) file paths on the Galaxy server to upload to the library, one file per line
  - **link_data_only: (optional, only when upload_option is** 'upload_directory' or 'upload_paths') either 'copy_files' (default) or 'link_to_files'. Setting to 'link_to_files' symlinks instead of copying the files
  - **name: (optional, only if create_type is 'folder') name of the** folder to create
  - **description: (optional, only if create_type is 'folder')** description of the folder to create

Return type dict

Returns a dictionary containing the id, name, and 'show' url of the new item

delete(*self*, *trans*, *library_id*, *id*, ***kwd*)

•**DELETE /api/libraries/{library_id}/contents/{id}** delete the LibraryDataset with the given `id`

Parameters

- **id** (*str*) – the encoded id of the library dataset to delete
- **kwd** (*dict*) – (optional) dictionary structure containing:

– **payload: a dictionary itself containing:**

* purge: if True, purge the LD

Return type dict

Returns an error object if an error occurred or a dictionary containing: * id: the encoded id of the library dataset, * deleted: if the library dataset was marked as deleted, * purged: if the library dataset was purged

index (*self*, *trans*, *library_id*, ***kwd*)

•**GET /api/libraries/{library_id}/contents:** Returns a list of library files and folders.

Note: May be slow! Returns all content traversing recursively through all folders.

See also:

`galaxy.webapps.galaxy.api.FolderContentsController.index` for a non-recursive solution

Parameters **library_id** (*str*) – the encoded id of the library

Returns

list of dictionaries of the form: * id: the encoded id of the library item * name: the 'library path'

or relationship of the library item to the root

- type: 'file' or 'folder'
- url: the url to get detailed information on the library item

Return type *list*

Raises MalformedId, InconsistentDatabase, RequestParameterInvalidException, InternalServerError

show (*self*, *trans*, *id*, *library_id*, ***kwd*)

•**GET /api/libraries/{library_id}/contents/{id}** Returns information about library file or folder.

Parameters

- **id** (*str*) – the encoded id of the library item to return
- **library_id** (*str*) – the encoded id of the library that contains this item

Returns detailed library item information

Return type dict

See also:

galaxy.model.LibraryDataset.to_dict() and *galaxy.model.LibraryFolder.dict_element_vi*

update (*self*, *trans*, *id*, *library_id*, *payload*, ***kwd*)

•**PUT /api/libraries/{library_id}/contents/{id}** create a ImplicitlyConvertedDatasetAssociation

See also:

galaxy.model.ImplicitlyConvertedDatasetAssociation

Parameters

- **id** (*str*) – the encoded id of the library item to return
- **library_id** (*str*) – the encoded id of the library that contains this item
- **payload** (*dict*) – dictionary structure containing:: 'converted_dataset_id':

Return type None

Returns None

metrics Module API operations for for querying and recording user metrics from some client (typically a user's browser).

class galaxy.webapps.galaxy.api.metrics.**MetricsController**(*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*

create(*trans*, *payload*)

•**POST /api/metrics:** record any metrics sent and return some status object

Note: Anonymous users can post metrics

Parameters **payload** (*dict*) – (optional) dictionary structure containing: * metrics: a list containing dictionaries of the form:

** namespace: label indicating the source of the metric ** time: isoformat datetime when the metric was recorded ** level: an integer representing the metric's log level ** args: a json string containing an array of extra data

Return type dict

Returns status object

debugging = None

set to true to send additional debugging info to the log

page_revisions Module API for updating Galaxy Pages

class galaxy.webapps.galaxy.api.page_revisions.**PageRevisionsController**(*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.SharableIt*
galaxy.model.item_attrs.UsesAnnotations, *galaxy.web.base.controller.SharableMixin*

create(*self*, *trans*, *page_id*, *payload* ***kwd*)

•**POST /api/pages/{page_id}/revisions** Create a new revision for a page

Parameters

- **page_id** – Add revision to Page with ID=page_id
- **payload** – A dictionary containing:: 'title' = New title of the page 'content' = New content of the page

Return type dictionary

Returns Dictionary with 'success' or 'error' element to indicate the result of the request

index(*self*, *trans*, *page_id*, ***kwd*)

•**GET /api/pages/{page_id}/revisions** return a list of Page revisions

Parameters **page_id** – Display the revisions of Page with ID=page_id

Return type *list*

Returns dictionaries containing different revisions of the page

pages Module API for updating Galaxy Pages

class galaxy.webapps.galaxy.api.pages.**PagesController**(*app*)

Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.SharableIt*
galaxy.model.item_attrs.UsesAnnotations, *galaxy.web.base.controller.SharableMixin*

create(*self*, *trans*, *payload*, ***kwd*)

•**POST /api/pages** Create a page and return dictionary containing Page summary

Parameters **payload** – dictionary structure containing:: 'slug' = The title slug for the page URL, must be unique 'title' = Title of the page 'content' = HTML contents of the page 'annotation' = Annotation that will be attached to the page

Return type dict

Returns Dictionary return of the Page.to_dict call

delete(*self*, *trans*, *id*, ***kwd*)

•**DELETE /api/pages/{id}** Create a page and return dictionary containing Page summary

Parameters **id** – ID of page to be deleted

Return type dict

Returns Dictionary with 'success' or 'error' element to indicate the result of the request

index(*self*, *trans*, *deleted=False*, ***kwd*)

•**GET /api/pages** return a list of Pages viewable by the user

Parameters **deleted** – Display deleted pages

Return type *list*

Returns dictionaries containing summary or detailed Page information

show(*self*, *trans*, *id*, ***kwd*)

•**GET /api/pages/{id}** View a page summary and the content of the latest revision

Parameters **id** – ID of page to be displayed

Return type dict

Returns Dictionary return of the Page.to_dict call with the 'content' field populated by the most recent revision

provenance Module API operations provenance

class galaxy.webapps.galaxy.api.provenance.**BaseProvenanceController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

create(*trans*, **args*, ***kwargs*)

delete(*trans*, **args*, ***kwargs*)

index(*trans*, **args*, ***kwargs*)

show(*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.api.provenance.**HDAProvenanceController**(*app*)
Bases: galaxy.webapps.galaxy.api.provenance.BaseProvenanceController

controller_name = 'history_content_provenance'

provenance_item_class = 'HistoryDatasetAssociation'

provenance_item_id = 'history_content_id'

class galaxy.webapps.galaxy.api.provenance.**LDDAProvenanceController**(*app*)
Bases: galaxy.webapps.galaxy.api.provenance.BaseProvenanceController

controller_name = 'ldda_provenance'

provenance_item_class = 'LibraryDatasetDatasetAssociation'

provenance_item_id = 'library_content_id'

quotas Module API operations on Quota objects.

class galaxy.webapps.galaxy.api.quotas.**QuotaAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controllers.admin.Adm
galaxy.actions.admin.AdminActions, galaxy.web.base.controller.UsesQuotaMixin,
galaxy.web.params.QuotaParamParser

create(*trans*, **args*, ***kwargs*)
POST /api/quotas Creates a new quota.

delete(*trans*, **args*, ***kwargs*)
DELETE /api/quotas/{encoded_quota_id} Deletes a quota

index(*trans*, **args*, ***kwargs*)
GET /api/quotas GET /api/quotas/deleted Displays a collection (list) of quotas.

show(*trans*, **args*, ***kwargs*)
GET /api/quotas/{encoded_quota_id} GET /api/quotas/deleted/{encoded_quota_id} Displays information about a quota.

undelete(*trans*, **args*, ***kwargs*)
POST /api/quotas/deleted/{encoded_quota_id}/undelete Undeletes a quota

update(*trans*, **args*, ***kwargs*)
PUT /api/quotas/{encoded_quota_id} Modifies a quota.

request_types Module API operations on RequestType objects.

class galaxy.webapps.galaxy.api.request_types.**RequestTypeAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

create(*trans*, **args*, ***kwargs*)
POST /api/request_types Creates a new request type (external_service configuration).

index (*trans*, **args*, ***kwargs*)
GET /api/request_types Displays a collection (list) of request_types.

show (*trans*, **args*, ***kwargs*)
GET /api/request_types/{encoded_request_type_id} Displays information about a request_type.

requests Module API operations on a sample tracking system.

class galaxy.webapps.galaxy.api.requests.**RequestsAPIController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

index (*trans*, **args*, ***kwargs*)
GET /api/requests Displays a collection (list) of sequencing requests.

show (*trans*, **args*, ***kwargs*)
GET /api/requests/{encoded_request_id} Displays details of a sequencing request.

update (*trans*, **args*, ***kwargs*)
PUT /api/requests/{encoded_request_id} Updates a request state, sample state or sample dataset transfer status depending on the update_type

v = ('REQUEST', 'request_state')

roles Module API operations on Role objects.

class galaxy.webapps.galaxy.api.roles.**RoleAPIController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

create (*trans*, **args*, ***kwargs*)
POST /api/roles Creates a new role.

index (*trans*, **args*, ***kwargs*)
GET /api/roles Displays a collection (list) of roles.

show (*trans*, **args*, ***kwargs*)
GET /api/roles/{encoded_role_id} Displays information about a role.

samples Module API operations for samples in the Galaxy sample tracking system.

class galaxy.webapps.galaxy.api.samples.**SamplesAPIController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*

index (*trans*, **args*, ***kwargs*)
GET /api/requests/{encoded_request_id}/samples Displays a collection (list) of sample of a sequencing request.

k = 'SAMPLE_DATASET'

update (*trans*, **args*, ***kwargs*)
PUT /api/samples/{encoded_sample_id} Updates a sample or objects related (mapped) to a sample.

update_type_values = ['sample_state', 'run_details', 'sample_dataset_transfer_status']

update_types = <galaxy.util.bunch.Bunch object>

v = ['sample_dataset_transfer_status']

search Module API for searching Galaxy Datasets

class galaxy.webapps.galaxy.api.search.**SearchController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.SharableIt

create(*trans*, **args*, ***kwargs*)
POST /api/search Do a search of the various elements of Galaxy.

tool_data Module

class galaxy.webapps.galaxy.api.tool_data.**ToolData**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

RESTful controller for interactions with tool data

delete(*trans*, **args*, ***kwargs*)
DELETE /api/tool_data/{id} Removes an item from a data table

Parameters

- **id** (*str*) – the id of the data table containing the item to delete
- **kwd** (*dict*) – (required) dictionary structure containing:
  - **payload: a dictionary itself containing:**
    - values: <TAB> separated list of column contents, there must be a value for all the columns of the data table

download_field_file(*trans*, **args*, ***kwargs*)

index(*trans*, **args*, ***kwargs*)
GET /api/tool_data: returns a list tool_data tables:

reload(*trans*, **args*, ***kwargs*)
GET /api/tool_data/{id}/reload

Reloads a tool_data table.

show(*trans*, **args*, ***kwargs*)

show_field(*trans*, **args*, ***kwargs*)
GET /api/tool_data/<id>/fields/<value>

Get information about a partiular field in a tool_data table

tool_shed_repositories Module

class galaxy.webapps.galaxy.api.tool_shed_repositories.**ToolShedRepositoriesController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController

RESTful controller for interactions with tool shed repositories.

exported_workflows(*trans*, **args*, ***kwargs*)
GET /api/tool_shed_repositories/{encoded_tool_shed_repository_id}/exported_workflows

Display a list of dictionaries containing information about this tool shed repository's exported workflows.

Parameters **id** – the encoded id of the ToolShedRepository object

get_latest_installable_revision(*trans*, **args*, ***kwargs*)
POST /api/tool_shed_repositories/get_latest_installable_revision Get the latest installable revision of a specified repository from a specified Tool Shed.

Parameters **key** – the current Galaxy admin user's API key

The following parameters are included in the payload. :param tool_shed_url (required): the base URL of the Tool Shed from which to retrieve the Repository revision. :param name (required): the name of the Repository :param owner (required): the owner of the Repository

import_workflow(*trans*, **args*, ***kwargs*)
POST /api/tool_shed_repositories/import_workflow

Import the specified exported workflow contained in the specified installed tool shed repository into Galaxy.

Parameters

- **key** – the API key of the Galaxy user with which the imported workflow will be associated.
- **id** – the encoded id of the ToolShedRepository object

The following parameters are included in the payload. :param index: the index location of the workflow tuple in the list of exported workflows stored in the metadata for the specified repository

import_workflows(*trans*, **args*, ***kwargs*)
POST /api/tool_shed_repositories/import_workflows

Import all of the exported workflows contained in the specified installed tool shed repository into Galaxy.

Parameters

- **key** – the API key of the Galaxy user with which the imported workflows will be associated.
- **id** – the encoded id of the ToolShedRepository object

index(*trans*, **args*, ***kwargs*)
GET /api/tool_shed_repositories Display a list of dictionaries containing information about installed tool shed repositories.

install_repository_revision(*trans*, **args*, ***kwargs*)
POST /api/tool_shed_repositories/install_repository_revision Install a specified repository revision from a specified tool shed into Galaxy.

Parameters **key** – the current Galaxy admin user's API key

The following parameters are included in the payload. :param tool_shed_url (required): the base URL of the Tool Shed from which to install the Repository :param name (required): the name of the Repository :param owner (required): the owner of the Repository :param changeset_revision (required): the changeset_revision of the RepositoryMetadata object associated with the Repository :param new_tool_panel_section_label (optional): label of a new section to be added to the Galaxy tool panel in which to load

tools contained in the Repository. Either this parameter must be an empty string or the tool_panel_section_id parameter must be an empty string or both must be an empty string (both cannot be used simultaneously).

Parameters

- **(optional)** (*shed_tool_conf*) – id of the Galaxy tool panel section in which to load tools contained in the Repository. If this parameter is an empty string and the above new_tool_panel_section_label parameter is an empty string, tools will be loaded outside of any sections in the tool panel. Either this parameter must be an empty string or the tool_panel_section_id parameter must be an empty string of both must be an empty string (both cannot be used simultaneously).

- **(optional)** – Set to True if you want to install repository dependencies defined for the specified repository being installed. The default setting is False.
- **(optional)** – Set to True if you want to install tool dependencies defined for the specified repository being installed. The default setting is False.
- **(optional)** – The shed-related tool panel configuration file configured in the "tool_config_file" setting in the Galaxy config file (e.g., galaxy.ini). At least one shed-related tool panel config file is required to be configured. Setting this parameter to a specific file enables you to choose where the specified repository will be installed because the tool_path attribute of the <toolbox> from the specified file is used as the installation location (e.g., <toolbox tool_path="../shed_tools">). If this parameter is not set, a shed-related tool panel configuration file will be selected automatically.

install_repository_revisions(*trans*, **args*, ***kwargs*)

POST /api/tool_shed_repositories/install_repository_revisions Install one or more specified repository revisions from one or more specified tool sheds into Galaxy. The received parameters must be ordered lists so that positional values in tool_shed_urls, names, owners and changeset_revisions are associated.

It's questionable whether this method is needed as the above method for installing a single repository can probably cover all desired scenarios. We'll keep this one around just in case...

Parameters **key** – the current Galaxy admin user's API key

The following parameters are included in the payload. :param tool_shed_urls: the base URLs of the Tool Sheds from which to install a specified Repository :param names: the names of the Repositories to be installed :param owners: the owners of the Repositories to be installed :param changeset_revisions: the changeset_revisions of each RepositoryMetadata object associated with each Repository to be installed :param new_tool_panel_section_label: optional label of a new section to be added to the Galaxy tool panel in which to load

tools contained in the Repository. Either this parameter must be an empty string or the tool_panel_section_id parameter must be an empty string, as both cannot be used.

Parameters

- **tool_panel_section_id** – optional id of the Galaxy tool panel section in which to load tools contained in the Repository. If not set, tools will be loaded outside of any sections in the tool panel. Either this parameter must be an empty string or the tool_panel_section_id parameter must be an empty string, as both cannot be used.
- **(optional)** (*shed_tool_conf*) – Set to True if you want to install repository dependencies defined for the specified repository being installed. The default setting is False.
- **(optional)** – Set to True if you want to install tool dependencies defined for the specified repository being installed. The default setting is False.
- **(optional)** – The shed-related tool panel configuration file configured in the "tool_config_file" setting in the Galaxy config file (e.g., galaxy.ini). At least one shed-related tool panel config file is required to be configured. Setting this parameter to a specific file enables you to choose where the specified repository will be installed because the tool_path attribute of the <toolbox> from the specified file is used as the installation location (e.g., <toolbox tool_path="../shed_tools">). If this parameter is not set, a shed-related tool panel configuration file will be selected automatically.

repair_repository_revision (*trans*, **args*, ***kwargs*)
POST /api/tool_shed_repositories/repair_repository_revision Repair a specified repository revision previously installed into Galaxy.

Parameters **key** – the current Galaxy admin user's API key

The following parameters are included in the payload. :param tool_shed_url (required): the base URL of the Tool Shed from which the Repository was installed :param name (required): the name of the Repository :param owner (required): the owner of the Repository :param changeset_revision (required): the changeset_revision of the RepositoryMetadata object associated with the Repository

reset_metadata_on_installed_repositories (*trans*, **args*, ***kwargs*)
PUT /api/tool_shed_repositories/reset_metadata_on_installed_repositories

Resets all metadata on all repositories installed into Galaxy in an "orderly fashion".

Parameters **key** – the API key of the Galaxy admin user.

show (*trans*, **args*, ***kwargs*)
GET /api/tool_shed_repositories/{encoded_tool_shed_repsository_id} Display a dictionary containing information about a specified tool_shed_repository.

Parameters **id** – the encoded id of the ToolShedRepository object

galaxy.webapps.galaxy.api.tool_shed_repositories.**get_message_for_no_shed_tool_config**()

tools Module

users Module API operations on User objects.

class galaxy.webapps.galaxy.api.users.**UserAPIController** (*app*)
Bases: *galaxy.web.base.controller.BaseAPIController*, *galaxy.web.base.controller.UsesTagsMi*
galaxy.web.base.controller.CreatesUsersMixin, *galaxy.web.base.controller.CreatesApiKeysI*

anon_user_api_value (*trans*)
Returns data for an anonymous user, truncated to only usage and quota_percent

api_key (*trans*, **args*, ***kwargs*)
POST /api/users/{encoded_user_id}/api_key Creates a new API key for specified user.

create (*trans*, **args*, ***kwargs*)
POST /api/users Creates a new Galaxy user.

delete (*trans*, **args*, ***kwargs*)

index (*trans*, **args*, ***kwargs*)
GET /api/users GET /api/users/deleted Displays a collection (list) of users.

show (*trans*, **args*, ***kwargs*)
GET /api/users/{encoded_user_id} GET /api/users/deleted/{encoded_user_id} GET /api/users/current Displays information about a user.

undelete (*trans*, **args*, ***kwargs*)

update (*trans*, **args*, ***kwargs*)

visualizations Module Visualizations resource control over the API.

NOTE!: this is a work in progress and functionality and data structures may change often.

class galaxy.webapps.galaxy.api.visualizations.**VisualizationsController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesVisual
galaxy.web.base.controller.SharableMixin, galaxy.model.item_attrs.UsesAnnotations

RESTful controller for interactions with visualizations.

create(*trans*, **args*, ***kwargs*)
POST /api/visualizations creates a new visualization using the given payload

POST /api/visualizations?import_id={encoded_visualization_id} imports a copy of an existing visualization into the user's workspace

index(*trans*, **args*, ***kwargs*)
GET /api/visualizations:

show(*trans*, **args*, ***kwargs*)
GET /api/visualizations/{viz_id}

update(*trans*, **args*, ***kwargs*)
PUT /api/visualizations/{encoded_visualization_id}

workflows Module API operations for Workflows

class galaxy.webapps.galaxy.api.workflows.**WorkflowsAPIController**(*app*)
Bases: galaxy.web.base.controller.BaseAPIController, galaxy.web.base.controller.UsesStored
galaxy.model.item_attrs.UsesAnnotations, galaxy.web.base.controller.SharableMixin

build_module(*trans*, **args*, ***kwargs*)
POST /api/workflows/build_module Builds module details including a tool model for the workflow editor.

cancel_invocation(*trans*, **args*, ***kwargs*)
DELETE /api/workflows/{workflow_id}/invocation/{invocation_id} Cancel the specified workflow invocation.

Parameters

- **workflow_id** (*str*) – the workflow id (required)
- **invocation_id** (*str*) – the usage id (required)

Raises exceptions.MessageException, exceptions.ObjectNotFound

create(*trans*, **args*, ***kwargs*)
POST /api/workflows

Run or create workflows from the api.

If installed_repository_file or from_history_id is specified a new workflow will be created for this user. Otherwise, workflow_id must be specified and this API method will cause a workflow to execute.

:param installed_repository_file The path of a workflow to import. Either workflow_id, installed_repository_file or from_history_id must be specified :type installed_repository_file str

Parameters

- **workflow_id** (*str*) – An existing workflow id. Either workflow_id, installed_repository_file or from_history_id must be specified
- **parameters** (*dict*) – If workflow_id is set - see _update_step_parameters()
- **ds_map** (*dict*) – If workflow_id is set - a dictionary mapping each input step id to a dictionary with 2 keys: 'src' (which can be 'ldda', 'ld' or 'hda') and 'id'

(which should be the id of a LibraryDatasetDatasetAssociation, LibraryDataset or HistoryDatasetAssociation respectively)

- **no_add_to_history** (*str*) – If workflow_id is set - if present in the payload with any value, the input datasets will not be added to the selected history
- **history** (*str*) – If workflow_id is set - optional history where to run the workflow, either the name of a new history or "hist_id=HIST_ID" where HIST_ID is the id of an existing history. If not specified, the workflow will be run a new unnamed history
- **replacement_params** (*dict*) – If workflow_id is set - an optional dictionary used when renaming datasets
- **from_history_id** (*str*) – Id of history to extract a workflow from. Either workflow_id, installed_repository_file or from_history_id must be specified
- **job_ids** (*str*) – If from_history_id is set - optional list of jobs to include when extracting a workflow from history
- **dataset_ids** (*str*) – If from_history_id is set - optional list of HDA 'hid's corresponding to workflow inputs when extracting a workflow from history
- **dataset_collection_ids** (*str*) – If from_history_id is set - optional list of HDCA 'hid's corresponding to workflow inputs when extracting a workflow from history
- **workflow_name** (*str*) – If from_history_id is set - name of the workflow to create when extracting a workflow from history

delete (*trans*, **args*, ***kwargs*)

DELETE /api/workflows/{encoded_workflow_id} Deletes a specified workflow Author: rpark

copied from galaxy.web.controllers.workflows.py (delete)

import_new_workflow_deprecated (*trans*, **args*, ***kwargs*)

POST /api/workflows/upload Importing dynamic workflows from the api. Return newly generated workflow id. Author: rpark

currently assumes payload['workflow'] is a json representation of a workflow to be inserted into the database

Deprecated in favor to POST /api/workflows with encoded 'workflow' in payload the same way.

import_shared_workflow_deprecated (*trans*, **args*, ***kwargs*)

POST /api/workflows/import Import a workflow shared by other users.

Parameters **workflow_id** (*str*) – the workflow id (required)

Raises exceptions.MessageException, exceptions.ObjectNotFound

index (*trans*, **args*, ***kwargs*)

GET /api/workflows

Displays a collection of workflows.

Parameters **show_published** (*boolean*) – if True, show also published workflows

index_invocations (*trans*, **args*, ***kwargs*)

GET /api/workflows/{workflow_id}/invocations

Get the list of the workflow invocations

Parameters **workflow_id** (*str*) – the workflow id (required)

Raises exceptions.MessageException, exceptions.ObjectNotFound

invocation_step(*trans*, **args*, ***kwargs*)
GET /api/workflows/{workflow_id}/invocation/{invocation_id}/steps/{step_id}

Parameters

- **workflow_id** (*str*) – the workflow id (required)
- **invocation_id** (*str*) – the invocation id (required)
- **step_id** (*str*) – encoded id of the WorkflowInvocationStep (required)
- **payload** – payload containing update action information for running workflow.

Raises exceptions.MessageException, exceptions.ObjectNotFound

invoke(*trans*, **args*, ***kwargs*)
POST /api/workflows/{encoded_workflow_id}/invocations

Schedule the workflow specified by *workflow_id* to run.

show(*trans*, **args*, ***kwargs*)
GET /api/workflows/{encoded_workflow_id}

Displays information needed to run a workflow from the command line.

show_invocation(*trans*, **args*, ***kwargs*)
GET /api/workflows/{workflow_id}/invocation/{invocation_id} Get detailed description of workflow invocation

Parameters

- **workflow_id** (*str*) – the workflow id (required)
- **invocation_id** (*str*) – the invocation id (required)

Raises exceptions.MessageException, exceptions.ObjectNotFound

update(*trans*, **args*, ***kwargs*)

•**PUT /api/workflows/{id}** updates the workflow stored with `id`

Parameters

- **id** (*str*) – the encoded id of the workflow to update
- **payload** (*dict*) – a dictionary containing any or all the * workflow the json description of the workflow as would be

  produced by GET workflows/<id>/download or given to *POST workflows*

  The workflow contents will be updated to target this.

Return type dict

Returns serialized version of the workflow

update_invocation_step(*trans*, **args*, ***kwargs*)
PUT /api/workflows/{workflow_id}/invocation/{invocation_id}/steps/{step_id} Update state of running workflow step invocation - still very nebulous but this would be for stuff like confirming paused steps can proceed etc....

Parameters

- **workflow_id** (*str*) – the workflow id (required)

- **invocation_id** (*str*) – the usage id (required)
- **step_id** (*str*) – encoded id of the WorkflowInvocationStep (required)

Raises exceptions.MessageException, exceptions.ObjectNotFound

workflow_dict (*trans*, **args*, ***kwargs*)
GET /api/workflows/{encoded_workflow_id}/download Returns a selected workflow as a json dictionary.

controllers Package

controllers Package Galaxy web controllers.

admin Module

class galaxy.webapps.galaxy.controllers.admin.**AdminGalaxy** (*app*)
Bases: galaxy.web.base.controller.BaseUIController, galaxy.web.base.controllers.admin.Admi galaxy.actions.admin.AdminActions, galaxy.web.base.controller.UsesQuotaMixin, galaxy.web.params.QuotaParamParser

check_for_tool_dependencies (*trans*, *migration_stage*)

create_quota (*trans*, **args*, ***kwargs*)

delete_operation = <galaxy.web.framework.helpers.grids.GridOperation object>

display_applications (*trans*, **args*, ***kwargs*)

edit_quota (*trans*, **args*, ***kwargs*)

group_list_grid = <galaxy.webapps.galaxy.controllers.admin.GroupListGrid object>

impersonate (*trans*, **args*, ***kwargs*)

manage_users_and_groups_for_quota (*trans*, **args*, ***kwargs*)

mark_quota_deleted (*trans*, **args*, ***kwargs*)

purge_operation = <galaxy.web.framework.helpers.grids.GridOperation object>

purge_quota (*trans*, **args*, ***kwargs*)

quota_list_grid = <galaxy.webapps.galaxy.controllers.admin.QuotaListGrid object>

quotas (*trans*, **args*, ***kwargs*)

recalculate_user_disk_usage (*trans*, **args*, ***kwargs*)

reload_display_application (*trans*, **args*, ***kwargs*)

rename_quota (*trans*, **args*, ***kwargs*)

review_tool_migration_stages (*trans*, **args*, ***kwargs*)

role_list_grid = <galaxy.webapps.galaxy.controllers.admin.RoleListGrid object>

set_quota_default (*trans*, **args*, ***kwargs*)

tool_version_list_grid = <galaxy.webapps.galaxy.controllers.admin.ToolVersionListGrid object>

undelete_operation = <galaxy.web.framework.helpers.grids.GridOperation object>

undelete_quota (*trans*, **args*, ***kwargs*)

unset_quota_default (*trans*, **args*, ***kwargs*)

user_list_grid = <galaxy.webapps.galaxy.controllers.admin.UserListGrid object>

view_datatypes_registry(*trans*, **args*, ***kwargs*)

view_tool_data_tables(*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.controllers.admin.**GroupListGrid**

Bases: galaxy.web.framework.helpers.grids.Grid

class NameColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *group*)

class GroupListGrid.**RolesColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *group*)

class GroupListGrid.**StatusColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *group*)

class GroupListGrid.**UsersColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *group*)

GroupListGrid.**columns** = [<galaxy.webapps.galaxy.controllers.admin.NameColumn object at 0x7f496e23a450>, <ga

GroupListGrid.**default_sort_key** = 'name'

GroupListGrid.**global_actions** = [<galaxy.web.framework.helpers.grids.GridAction object at 0x7f496e23ae50>]

GroupListGrid.**model_class**

alias of Group

GroupListGrid.**num_rows_per_page** = 50

GroupListGrid.**operations** = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f496e23a5d0>, <g

GroupListGrid.**preserve_state** = False

GroupListGrid.**standard_filters** = [<galaxy.web.framework.helpers.grids.GridColumnFilter object at 0x7f496b

GroupListGrid.**template** = '/admin/dataset_security/group/grid.mako'

GroupListGrid.**title** = 'Groups'

GroupListGrid.**use_paging** = True

class galaxy.webapps.galaxy.controllers.admin.**QuotaListGrid**

Bases: galaxy.web.framework.helpers.grids.Grid

class AmountColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *quota*)

class QuotaListGrid.**DescriptionColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *quota*)

class QuotaListGrid.**GroupsColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *quota*)

class QuotaListGrid.**NameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *quota*)

class QuotaListGrid.**StatusColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *quota*)

class QuotaListGrid.**UsersColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *quota*)

QuotaListGrid.**columns** = **[<galaxy.webapps.galaxy.controllers.admin.NameColumn object at 0x7f496b268110>, <ga**

QuotaListGrid.**default_sort_key** = **'name'**

QuotaListGrid.**global_actions** = **[<galaxy.web.framework.helpers.grids.GridAction object at 0x7f496b268ed0>]**

QuotaListGrid.**model_class**

alias of Quota

QuotaListGrid.**num_rows_per_page** = **50**

QuotaListGrid.**operations** = **[<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f496b268cd0>, <g**

QuotaListGrid.**preserve_state** = **False**

QuotaListGrid.**standard_filters** = **[<galaxy.web.framework.helpers.grids.GridColumnFilter object at 0x7f496b**

QuotaListGrid.**template** = '/admin/quota/grid.mako'

QuotaListGrid.**title** = 'Quotas'

QuotaListGrid.**use_paging** = True

class galaxy.webapps.galaxy.controllers.admin.**RoleListGrid**

Bases: *galaxy.web.framework.helpers.grids.Grid*

class **DescriptionColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *role*)

class RoleListGrid.**GroupsColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.GridColumn*

get_value(*trans*, *grid*, *role*)

class RoleListGrid.**NameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *role*)

class RoleListGrid.**StatusColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.GridColumn*

get_value(*trans*, *grid*, *role*)

class RoleListGrid.**TypeColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *role*)

class RoleListGrid.**UsersColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.GridColumn*

get_value(*trans*, *grid*, *role*)

RoleListGrid.**apply_query_filter**(*trans*, *query*, ***kwargs*)

RoleListGrid.**columns** = [<galaxy.webapps.galaxy.controllers.admin.NameColumn object at 0x7f496e23a410>, <gala

RoleListGrid.**default_sort_key** = 'name'

RoleListGrid.**global_actions** = [<galaxy.web.framework.helpers.grids.GridAction object at 0x7f496e23ac50>]

RoleListGrid.**model_class**
alias of Role

RoleListGrid.**num_rows_per_page = 50**

RoleListGrid.**operations = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f496e23a590>, <gal**

RoleListGrid.**preserve_state = False**

RoleListGrid.**standard_filters = [<galaxy.web.framework.helpers.grids.GridColumnFilter object at 0x7f496e23**

RoleListGrid.**template = '/admin/dataset_security/role/grid.mako'**

RoleListGrid.**title = 'Roles'**

RoleListGrid.**use_paging = True**

class galaxy.webapps.galaxy.controllers.admin.**ToolVersionListGrid**
Bases: *galaxy.web.framework.helpers.grids.Grid*

class ToolIdColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)
Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *tool_version*)

class ToolVersionListGrid.**ToolVersionsColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)
Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *tool_version*)

ToolVersionListGrid.**build_initial_query**(*trans*, ***kwd*)

ToolVersionListGrid.**columns = [<galaxy.webapps.galaxy.controllers.admin.ToolIdColumn object at 0x7f496b268**

ToolVersionListGrid.**default_filter = {}**

ToolVersionListGrid.**default_sort_key = 'tool_id'**

ToolVersionListGrid.**global_actions = []**

ToolVersionListGrid.**model_class**
alias of ToolVersion

ToolVersionListGrid.**num_rows_per_page = 50**

ToolVersionListGrid.**operations = []**

ToolVersionListGrid.**preserve_state = False**

ToolVersionListGrid.**standard_filters = []**

ToolVersionListGrid.**template = '/admin/tool_version/grid.mako'**

ToolVersionListGrid.**title = 'Tool versions'**

ToolVersionListGrid.**use_paging = True**

class galaxy.webapps.galaxy.controllers.admin.**UserListGrid**
Bases: *galaxy.web.framework.helpers.grids.Grid*

class ActivatedColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *user*)

class UserListGrid.**EmailColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *user*)

class UserListGrid.**ExternalColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *user*)

class UserListGrid.**GroupsColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *user*)

class UserListGrid.**LastLoginColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *user*)

class UserListGrid.**RolesColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *user*)

class UserListGrid.**StatusColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *user*)

class UserListGrid.**TimeCreatedColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *user*)

class UserListGrid.**UserNameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *user*)

UserListGrid.**columns = [<galaxy.webapps.galaxy.controllers.admin.EmailColumn object at 0x7f496a27f750>, <gala**

UserListGrid.**default_sort_key = 'email'**

UserListGrid.**get_current_item**(*trans*, ***kwargs*)

UserListGrid.**global_actions = [<galaxy.web.framework.helpers.grids.GridAction object at 0x7f496e23a990>]**

UserListGrid.**model_class**

alias of User

UserListGrid.**num_rows_per_page = 50**

UserListGrid.**operations = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f496e23a890>, <gal**

UserListGrid.**preserve_state = False**

UserListGrid.**standard_filters = [<galaxy.web.framework.helpers.grids.GridColumnFilter object at 0x7f496e23**

UserListGrid.**template = '/admin/user/grid.mako'**

UserListGrid.**title = 'Users'**

UserListGrid.**use_paging = True**

admin_toolshed Module

class galaxy.webapps.galaxy.controllers.admin_toolshed.**AdminToolshed**(*app*)

Bases: galaxy.webapps.galaxy.controllers.admin.AdminGalaxy

activate_repository(*trans*, **args*, ***kwargs*)

Activate a repository that was deactivated but not uninstalled.

browse_repositories(*trans*, **args*, ***kwargs*)

browse_repository(*trans*, **args*, ***kwargs*)

browse_tool_dependency(*trans*, **args*, ***kwargs*)

browse_tool_shed(*trans*, **args*, ***kwargs*)

browse_tool_sheds(*trans*, **args*, ***kwargs*)

check_for_updates(*trans*, **args*, ***kwargs*)

Send a request to the relevant tool shed to see if there are any updates.

deactivate_or_uninstall_repository(*trans*, **args*, ***kwargs*)

Handle all changes when a tool shed repository is being deactivated or uninstalled. Notice that if the repository contents include a file named tool_data_table_conf.xml.sample, its entries are not removed from the defined config.shed_tool_data_table_config. This is because it becomes a bit complex to determine if other installed repositories include tools that require the same entry. For now we'll never delete entries from config.shed_tool_data_table_config, but we may choose to do so in the future if it becomes necessary.

display_image_in_repository(*trans*, ***kwd*)

Open an image file that is contained in an installed tool shed repository or that is referenced by a URL for display. The image can be defined in either a README.rst file contained in the repository or the

help section of a Galaxy tool config that is contained in the repository. The following image definitions are all supported. The former $PATH_TO_IMAGES is no longer required, and is now ignored. .. image:: https://raw.github.com/galaxy/some_image.png .. image:: $PATH_TO_IMAGES/some_image.png .. image:: /static/images/some_image.gif .. image:: some_image.jpg .. image:: /deep/some_image.png

find_tools_in_tool_shed(*trans*, **args*, ***kwargs*)

find_workflows_in_tool_shed(*trans*, **args*, ***kwargs*)

generate_workflow_image(*trans*, **args*, ***kwargs*)
Return an svg image representation of a workflow dictionary created when the workflow was exported.

get_file_contents(*trans*, **args*, ***kwargs*)

get_tool_dependencies(*trans*, **args*, ***kwargs*)
Send a request to the appropriate tool shed to retrieve the dictionary of tool dependencies defined for the received repository name, owner and changeset revision. The received repository_id is the encoded id of the installed tool shed repository in Galaxy. We need it so that we can derive the tool shed from which it was installed.

get_updated_repository_information(*trans*, **args*, ***kwargs*)
Send a request to the appropriate tool shed to retrieve the dictionary of information required to reinstall an updated revision of an uninstalled tool shed repository.

import_workflow(*trans*, **args*, ***kwargs*)
Import a workflow contained in an installed tool shed repository into Galaxy.

initiate_tool_dependency_installation(*trans*, **args*, ***kwargs*)
Install specified dependencies for repository tools. The received list of tool_dependencies are the database records for those dependencies defined in the tool_dependencies.xml file (contained in the repository) that should be installed. This allows for filtering out dependencies that have not been checked for installation on the 'Manage tool dependencies' page for an installed tool shed repository.

install_latest_repository_revision(*trans*, **args*, ***kwargs*)
Install the latest installable revision of a repository that has been previously installed.

install_tool_dependencies_with_update(*trans*, **args*, ***kwargs*)
Updating an installed tool shed repository where new tool dependencies but no new repository dependencies are included in the updated revision.

installed_repository_grid = <tool_shed.galaxy_install.grids.admin_toolshed_grids.InstalledRepositoryGrid obje

manage_repositories(*trans*, **args*, ***kwargs*)

manage_repository(*trans*, **args*, ***kwargs*)

manage_repository_tool_dependencies(*trans*, **args*, ***kwargs*)

manage_tool_dependencies(*trans*, **args*, ***kwargs*)

monitor_repository_installation(*trans*, **args*, ***kwargs*)

open_folder(*trans*, **args*, ***kwargs*)

prepare_for_install(*trans*, **args*, ***kwargs*)

purge_repository(*trans*, **args*, ***kwargs*)
Purge a "white ghost" repository from the database.

reinstall_repository(*trans*, **args*, ***kwargs*)
Reinstall a tool shed repository that has been previously uninstalled, making sure to handle all repository and tool dependencies of the repository.

repair_repository(*trans*, **args*, ***kwargs*)
Inspect the repository dependency hierarchy for a specified repository and attempt to make sure they are all properly installed as well as each repository's tool dependencies.

repair_tool_shed_repositories(*trans*, **args*, ***kwargs*)
Repair specified tool shed repositories.

repository_installation_grid = <tool_shed.galaxy_install.grids.admin_toolshed_grids.RepositoryInstallationGr

repository_installation_status_updates(*trans*, **args*, ***kwargs*)

reselect_tool_panel_section(*trans*, **args*, ***kwargs*)
Select or change the tool panel section to contain the tools included in the tool shed repository being reinstalled. If there are updates available for the repository in the tool shed, the tool_dependencies and repository_dependencies associated with the updated changeset revision will have been retrieved from the tool shed and passed in the received kwd. In this case, the stored tool shed repository metadata from the Galaxy database will not be used since it is outdated.

reset_metadata_on_selected_installed_repositories(*trans*, **args*, ***kwargs*)

reset_repository_metadata(*trans*, **args*, ***kwargs*)
Reset all metadata on a single installed tool shed repository.

reset_to_install(*trans*, **args*, ***kwargs*)
An error occurred while cloning the repository, so reset everything necessary to enable another attempt.

set_tool_versions(*trans*, **args*, ***kwargs*)
Get the tool_versions from the tool shed for each tool in the installed revision of a selected tool shed repository and update the metadata for the repository's revision in the Galaxy database.

tool_dependency_grid = <tool_shed.galaxy_install.grids.admin_toolshed_grids.ToolDependencyGrid object>

tool_dependency_status_updates(*trans*, **args*, ***kwargs*)

uninstall_tool_dependencies(*trans*, **args*, ***kwargs*)

update_to_changeset_revision(*trans*, **args*, ***kwargs*)
Update a cloned repository to the latest revision possible.

update_tool_shed_status_for_installed_repository(*trans*, **args*, ***kwargs*)

view_tool_metadata(*trans*, **args*, ***kwargs*)

view_workflow(*trans*, **args*, ***kwargs*)
Retrieve necessary information about a workflow from the database so that it can be displayed in an svg image.

async Module Upload class

class galaxy.webapps.galaxy.controllers.async.**ASync**(*app*)
Bases: *galaxy.web.base.controller.BaseUIController*

default(*trans*, *tool_id=None*, *data_id=None*, *data_secret=None*, ***kwd*)
Catches the tool id and redirects as needed

index(*trans*, *tool_id=None*, *data_secret=None*, ***kwd*)
Manages ascynchronous connections

cloudlaunch Module

data_admin Module

dataset Module

class galaxy.webapps.galaxy.controllers.dataset.**DatasetInterface**(*app*)

Bases: galaxy.web.base.controller.BaseUIController, galaxy.model.item_attrs.UsesAnnotation
galaxy.model.item_attrs.UsesItemRatings, galaxy.web.base.controller.UsesExtendedMetadata

annotate_async(*trans*, *id*, *new_annotation=None*, ***kwargs*)

copy_datasets(*trans*, *source_history=None*, *source_content_ids=''*, *target_history_id=None*, *target_history_ids=''*, *new_history_name=''*, *do_copy=False*, ***kwd*)

default(*trans*, *dataset_id=None*, ***kwd*)

delete(*trans*, *dataset_id*, *filename*, *show_deleted_on_refresh=False*)

delete_async(*trans*, *dataset_id*, *filename*)

display(*trans*, *dataset_id=None*, *preview=False*, *filename=None*, *to_ext=None*, *chunk=None*, ***kwd*)

display_application(*trans*, *dataset_id=None*, *user_id=None*, *app_name=None*, *link_name=None*, *app_action=None*, *action_param=None*, ***kwds*)

Access to external display applications

display_at(*trans*, *dataset_id*, *filename=None*, ***kwd*)

Sets up a dataset permissions so it is viewable at an external site

display_by_username_and_slug(*trans*, *username*, *slug*, *filename=None*, *preview=True*)

Display dataset by username and slug; because datasets do not yet have slugs, the slug is the dataset's id.

edit(*trans*, *dataset_id=None*, *filename=None*, *hid=None*, ***kwd*)

Allows user to modify parameters of an HDA.

errors(*trans*, *id*)

exit_code(*trans*, *dataset_id=None*, ***kwargs*)

get_annotation_async(*trans*, *id*)

get_embed_html_async(*trans*, *id*)

Returns HTML for embedding a dataset in a page.

get_item_content_async(*trans*, *id*)

Returns item content in HTML format.

get_metadata_file(*trans*, *hda_id*, *metadata_name*)

Allows the downloading of metadata files associated with datasets (eg. bai index for bam files)

get_name_and_link_async(*trans*, **args*, ***kwargs*)

Returns dataset's name and link.

imp(*trans*, *dataset_id=None*, ***kwd*)

Import another user's dataset via a shared URL; dataset is added to user's current history.

list(*trans*, **args*, ***kwargs*)

List all available datasets

purge(*trans*, *dataset_id*, *filename*, *show_deleted_on_refresh=False*)

purge_async(*trans*, *dataset_id*, *filename*)

rate_async(*trans*, **args*, ***kwargs*)

Rate a dataset asynchronously and return updated community data.

report_error(*trans*, *id*, *email=''*, *message=''*, ***kwd*)

set_accessible_async (*trans*, **args*, ***kwargs*)
Does nothing because datasets do not have an importable/accessible attribute. This method could potentially set another attribute.

show_params (*trans*, *dataset_id=None*, *from_noframe=None*, ***kwd*)
Show the parameters used for the job associated with an HDA

stderr (*trans*, *dataset_id=None*, ***kwargs*)

stdout (*trans*, *dataset_id=None*, ***kwargs*)

stored_list_grid = <galaxy.webapps.galaxy.controllers.dataset.HistoryDatasetAssociationListGrid object>

transfer_status (*trans*, **args*, ***kwargs*)
Primarily used for the S3ObjectStore - get the status of data transfer if the file is not in cache

undelete (*trans*, *dataset_id*, *filename*)

undelete_async (*trans*, *dataset_id*, *filename*)

unhide (*trans*, *dataset_id*, *filename*)

class galaxy.webapps.galaxy.controllers.dataset.**HistoryDatasetAssociationListGrid**
Bases: *galaxy.web.framework.helpers.grids.Grid*

class HistoryColumn (*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)
Bases: *galaxy.web.framework.helpers.grids.GridColumn*

get_value (*trans*, *grid*, *hda*)

class HistoryDatasetAssociationListGrid.**StatusColumn** (*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)
Bases: *galaxy.web.framework.helpers.grids.GridColumn*

get_accepted_filters ()
Returns a list of accepted filters for this column.

get_value (*trans*, *grid*, *hda*)

HistoryDatasetAssociationListGrid.**build_initial_query** (*trans*, ***kwargs*)

HistoryDatasetAssociationListGrid.**columns = [<galaxy.web.framework.helpers.grids.TextColumn object at**

HistoryDatasetAssociationListGrid.**default_filter = {'deleted': 'False', 'name': 'All', 'tags': 'All'}**

HistoryDatasetAssociationListGrid.**default_sort_key = '-update_time'**

HistoryDatasetAssociationListGrid.**model_class**
alias of HistoryDatasetAssociation

HistoryDatasetAssociationListGrid.**num_rows_per_page = 50**

HistoryDatasetAssociationListGrid.**operations = [<galaxy.web.framework.helpers.grids.GridOperation ob**

HistoryDatasetAssociationListGrid.**preserve_state = False**

HistoryDatasetAssociationListGrid.**standard_filters** = []

HistoryDatasetAssociationListGrid.**template** = '/dataset/grid.mako'

HistoryDatasetAssociationListGrid.**title** = 'Saved Datasets'

HistoryDatasetAssociationListGrid.**use_async** = True

HistoryDatasetAssociationListGrid.**use_paging** = True

error Module

class galaxy.webapps.galaxy.controllers.error.**Error**(*app*)

Bases: galaxy.web.base.controller.BaseUIController

index(*trans*)

external_service Module

class galaxy.webapps.galaxy.controllers.external_service.**ExternalService**(*app*)

Bases: galaxy.web.base.controller.BaseUIController, galaxy.web.base.controller.UsesFormDe

browse_external_services(*trans*, **args*, ***kwargs*)

create_external_service(*trans*, **args*, ***kwargs*)

delete_external_service(*trans*, **args*, ***kwargs*)

edit_external_service(*trans*, **args*, ***kwargs*)

edit_external_service_form_definition(*trans*, **args*, ***kwargs*)

external_service_grid = <galaxy.webapps.galaxy.controllers.external_service.ExternalServiceGrid object>

get_external_service_type(*trans*, *external_service_type_id*, *action='browse_external_services'*)

reload_external_service_types(*trans*, **args*, ***kwargs*)

undelete_external_service(*trans*, **args*, ***kwargs*)

update_external_service_form_definition(*trans*, **args*, ***kwargs*)

view_external_service(*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.controllers.external_service.**ExternalServiceGrid**

Bases: galaxy.web.framework.helpers.grids.Grid

class ExternalServiceTypeColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *external_service*)

class ExternalServiceGrid.**NameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *external_service*)

ExternalServiceGrid.**columns** = [<galaxy.webapps.galaxy.controllers.external_service.NameColumn object at 0x7

ExternalServiceGrid.**default_filter** = {'deleted': 'False'}

ExternalServiceGrid.**default_sort_key** = '-create_time'

ExternalServiceGrid.**global_actions** = [<galaxy.web.framework.helpers.grids.GridAction object at 0x7f49684e

ExternalServiceGrid.**model_class**
alias of *ExternalService*

ExternalServiceGrid.**num_rows_per_page** = 50

ExternalServiceGrid.**operations** = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f49684d1

ExternalServiceGrid.**preserve_state** = True

ExternalServiceGrid.**template** = 'admin/external_service/grid.mako'

ExternalServiceGrid.**title** = 'External Services'

ExternalServiceGrid.**use_paging** = True

external_services Module

class galaxy.webapps.galaxy.controllers.external_services.**ExternalServiceController**(*app*)

Bases: *galaxy.web.base.controller.BaseUIController*

access_action(*trans*, **args*, ***kwargs*)

forms Module

class galaxy.webapps.galaxy.controllers.forms.**Forms**(*app*)

Bases: *galaxy.web.base.controller.BaseUIController*

browse_form_definitions(*trans*, **args*, ***kwargs*)

build_form_definition_field_widgets(*trans*, *layout_grids*, *field_index*, *field*, *form_type*)
This method returns a list of widgets which describes a form definition field. This includes the field label, helptext, type, selectfield options, required/optional & layout

create_form_definition(*trans*, **args*, ***kwargs*)

delete_form_definition(*trans*, **args*, ***kwargs*)

edit_form_definition(*trans*, **args*, ***kwargs*)
This callback method is for handling form editing. The value of response_redirect should be an URL that is defined by the caller. This allows for redirecting as desired when the form changes have been saved. For an example of how this works, see the edit_template() method in the base controller.

empty_field = {'visible': True, 'helptext': '', 'name': '', 'default': '', 'layout': 'none', 'selectlist': [], 'required': False,

forms_grid = <galaxy.webapps.galaxy.controllers.forms.FormsGrid object>

get_current_form(*trans*, ***kwd*)
This method gets all the unsaved user-entered form details and returns a dictionary containing the name, desc, type, layout & fields of the form

get_saved_form(*form_definition*)
This retrieves the saved form and returns a dictionary containing the name, desc, type, layout & fields of the form

save_form_definition(*trans*, *form_definition_current_id=None*, ***kwd*)
This method saves the current form

show_editable_form_definition(*trans*, *form_definition*, *current_form*, *message=''*, *status='done'*, *response_redirect=None*, ***kwd*)

Displays the form and any of the changes made to it in edit mode. In this method all the widgets are build for all name, description and all the fields of a form definition.

undelete_form_definition(*trans*, **args*, ***kwargs*)

view_latest_form_definition(*trans*, **args*, ***kwargs*)

Displays the layout of the latest version of the form definition

class galaxy.webapps.galaxy.controllers.forms.**FormsGrid**

Bases: *galaxy.web.framework.helpers.grids.Grid*

class DescriptionColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *form*)

class FormsGrid.**NameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *form*)

class FormsGrid.**TypeColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *form*)

FormsGrid.**build_initial_query**(*trans*, ***kwargs*)

FormsGrid.**columns = [<galaxy.webapps.galaxy.controllers.forms.NameColumn object at 0x7f4968610e90>, <galaxy.w**

FormsGrid.**default_filter = {'deleted': 'False'}**

FormsGrid.**default_sort_key = '-create_time'**

FormsGrid.**global_actions = [<galaxy.web.framework.helpers.grids.GridAction object at 0x7f49683a0890>]**

FormsGrid.**model_class**

alias of FormDefinitionCurrent

FormsGrid.**num_rows_per_page = 50**

FormsGrid.**operations = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f49683a07d0>, <galaxy.**

FormsGrid.**preserve_state = True**

FormsGrid.**template = 'admin/forms/grid.mako'**

FormsGrid.**title = 'Forms'**

FormsGrid.**use_paging = True**

history Module

class galaxy.webapps.galaxy.controllers.history.**HistoryAllPublishedGrid**

Bases: *galaxy.web.framework.helpers.grids.Grid*

class NameURLColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.PublicURLColumn*, *galaxy.webapps.galaxy.controllers.history.NameColumn*

HistoryAllPublishedGrid.**apply_query_filter**(*trans*, *query*, ***kwargs*)

HistoryAllPublishedGrid.**build_initial_query**(*trans*, ***kwargs*)

HistoryAllPublishedGrid.**columns = [<galaxy.webapps.galaxy.controllers.history.NameURLColumn object at 0x**

HistoryAllPublishedGrid.**default_filter = {'username': 'All', 'public_url': 'All', 'tags': 'All'}**

HistoryAllPublishedGrid.**default_sort_key = 'update_time'**

HistoryAllPublishedGrid.**model_class**

alias of History

HistoryAllPublishedGrid.**num_rows_per_page = 50**

HistoryAllPublishedGrid.**operations = []**

HistoryAllPublishedGrid.**title = 'Published Histories'**

HistoryAllPublishedGrid.**use_async = True**

HistoryAllPublishedGrid.**use_paging = True**

class galaxy.webapps.galaxy.controllers.history.**HistoryController**(*app*)

Bases: *galaxy.web.base.controller.BaseUIController*, *galaxy.web.base.controller.SharableMix*
galaxy.model.item_attrs.UsesAnnotations, *galaxy.model.item_attrs.UsesItemRatings*,
galaxy.web.base.controller.ExportsHistoryMixin, *galaxy.web.base.controller.ImportsHisto*

citations(*trans*)

copy(*trans*, **args*, ***kwargs*)

Copy one or more histories

create_new_current(*trans*, **args*, ***kwargs*)

current_history_json(*trans*, **args*, ***kwargs*)

delete_current(*trans*, *purge=False*)

Delete just the active history – this does not require a logged in user.

delete_hidden_datasets(*trans*)

This method deletes all hidden datasets in the current history.

display_by_username_and_slug(*trans*, *username*, *slug*)

Display history based on a username and slug.

display_structured(*trans*, *id=None*)

Display a history as a nested structure showing the jobs and workflow invocations that created each dataset (if any).

export_archive(*trans*, *id=None*, *gzip=True*, *include_hidden=False*, *include_deleted=False*, *preview=False*)

Export a history to an archive.

get_item(*trans*, *id*)

get_name_and_link_async(*trans*, **args*, ***kwargs*)

Returns history's name and link.

history_data(*trans*, *history*)

imp (*trans*, *id=None*, *confirm=False*, ***kwd*)
Import another user's history via a shared URL

import_archive (*trans*, **args*, ***kwargs*)
Import a history from a file archive.

index (*trans*)

list (*trans*, **args*, ***kwargs*)
List all available histories

list_as_xml (*trans*)
XML history list for functional tests

list_published (*trans*, ***kwargs*)

list_shared (*trans*, **args*, ***kwargs*)
List histories shared with current user by others

name_autocomplete_data (*trans*, *q=None*, *limit=None*, *timestamp=None*)
Return autocomplete data for history names

published_list_grid = <galaxy.webapps.galaxy.controllers.history.HistoryAllPublishedGrid object>

purge_deleted_datasets (*trans*)

rate_async (*trans*, **args*, ***kwargs*)
Rate a history asynchronously and return updated community data.

rename (*trans*, **args*, ***kwargs*)

resume_paused_jobs (*trans*, *current=False*, *ids=None*)
Resume paused jobs the active history – this does not require a logged in user.

set_accessible_async (*trans*, **args*, ***kwargs*)
Set history's importable attribute and slug.

set_as_current (*trans*, **args*, ***kwargs*)

share (*trans*, **args*, ***kwargs*)

share_restricted (*trans*, **args*, ***kwargs*)

shared_list_grid = <galaxy.webapps.galaxy.controllers.history.SharedHistoryListGrid object>

sharing (*trans*, **args*, ***kwargs*)
Handle history sharing.

stored_list_grid = <galaxy.webapps.galaxy.controllers.history.HistoryListGrid object>

structure (*trans*, *id=None*, ***kwargs*)

switch_to_history (*trans*, **args*, ***kwargs*)

unhide_datasets (*trans*, *current=False*, *ids=None*)
Unhide the datasets in the active history – this does not require a logged in user.

view (*trans*, *id=None*, *show_deleted=False*, *show_hidden=False*, *use_panels=True*)
View a history. If a history is importable, then it is viewable by any user.

view_multiple (*trans*, *include_deleted_histories=False*, *order='update'*)

class galaxy.webapps.galaxy.controllers.history.**HistoryListGrid**
Bases: *galaxy.web.framework.helpers.grids.Grid*

class **DatasetsByStateColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *history*)

class HistoryListGrid.**DeletedColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.DeletedColumn

get_value(*trans*, *grid*, *history*)

sort(*trans*, *query*, *ascending*, *column_name=None*)

class HistoryListGrid.**HistoryListNameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.webapps.galaxy.controllers.history.NameColumn

get_link(*trans*, *grid*, *history*)

HistoryListGrid.**apply_query_filter**(*trans*, *query*, ***kwargs*)

HistoryListGrid.**columns = [<galaxy.webapps.galaxy.controllers.history.HistoryListNameColumn object at 0x7f49**

HistoryListGrid.**default_filter = {'deleted': 'False', 'sharing': 'All', 'name': 'All', 'tags': 'All'}**

HistoryListGrid.**default_sort_key = '-update_time'**

HistoryListGrid.**get_current_item**(*trans*, ***kwargs*)

HistoryListGrid.**info_text = 'Histories that have been deleted for more than a time period specified by the Galax**

HistoryListGrid.**model_class**

alias of History

HistoryListGrid.**num_rows_per_page = 50**

HistoryListGrid.**operations = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f4967456710>,**

HistoryListGrid.**preserve_state = False**

HistoryListGrid.**standard_filters = [<galaxy.web.framework.helpers.grids.GridColumnFilter object at 0x7f49**

HistoryListGrid.**template = '/history/grid.mako'**

HistoryListGrid.**title = 'Saved Histories'**

HistoryListGrid.**use_async = True**

HistoryListGrid.**use_paging = True**

class galaxy.webapps.galaxy.controllers.history.**NameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *history*)

class galaxy.webapps.galaxy.controllers.history.**SharedHistoryListGrid**

Bases: galaxy.web.framework.helpers.grids.Grid

class **DatasetsByStateColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *history*)

class SharedHistoryListGrid.**SharedByColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *history*)

SharedHistoryListGrid.**apply_query_filter**(*trans*, *query*, ***kwargs*)

SharedHistoryListGrid.**build_initial_query**(*trans*, ***kwargs*)

SharedHistoryListGrid.**columns = [<galaxy.web.framework.helpers.grids.GridColumn object at 0x7f49674569d0**

SharedHistoryListGrid.**default_filter = {}**

SharedHistoryListGrid.**default_sort_key = '-update_time'**

SharedHistoryListGrid.**model_class**

alias of History

SharedHistoryListGrid.**operations = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f49674**

SharedHistoryListGrid.**standard_filters = []**

SharedHistoryListGrid.**title = 'Histories shared with you by others'**

library Module

class galaxy.webapps.galaxy.controllers.library.**Library**(*app*)

Bases: galaxy.web.base.controller.BaseUIController

browse_libraries(*trans*, ***kwd*)

index(*trans*, ***kwd*)

library_list_grid = <galaxy.webapps.galaxy.controllers.library.LibraryListGrid object>

list(*trans*, ***kwd*)

class galaxy.webapps.galaxy.controllers.library.**LibraryListGrid**

Bases: galaxy.web.framework.helpers.grids.Grid

class DescriptionColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *library*)

class LibraryListGrid.**NameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *library*)

LibraryListGrid.**apply_query_filter**(*trans*, *query*, ***kwd*)

LibraryListGrid.**build_initial_query**(*trans*, ***kwargs*)

LibraryListGrid.**columns** = [<galaxy.webapps.galaxy.controllers.library.NameColumn object at 0x7f4967586710>,

LibraryListGrid.**default_filter** = {'deleted': 'False', 'description': 'All', 'purged': 'False', 'name': 'All'}

LibraryListGrid.**default_sort_key** = 'name'

LibraryListGrid.**model_class**

alias of Library

LibraryListGrid.**num_rows_per_page** = 50

LibraryListGrid.**preserve_state** = False

LibraryListGrid.**standard_filters** = []

LibraryListGrid.**template** = '/library/grid.mako'

LibraryListGrid.**title** = 'Data Libraries'

LibraryListGrid.**use_paging** = True

library_admin Module

class galaxy.webapps.galaxy.controllers.library_admin.**LibraryAdmin**(*app*)

Bases: galaxy.web.base.controller.BaseUIController

browse_libraries(*trans*, **args*, ***kwargs*)

create_library(*trans*, **args*, ***kwargs*)

delete_library(*trans*, **args*, ***kwargs*)

library_list_grid = <galaxy.webapps.galaxy.controllers.library_admin.LibraryListGrid object>

purge_library(*trans*, **args*, ***kwargs*)

undelete_library(*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.controllers.library_admin.**LibraryListGrid**

Bases: galaxy.web.framework.helpers.grids.Grid

class DescriptionColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *library*)

class LibraryListGrid.**NameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *library*)

class LibraryListGrid.**StatusColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *library*)

LibraryListGrid.**columns** = **[<galaxy.webapps.galaxy.controllers.library_admin.NameColumn object at 0x7f496857**

LibraryListGrid.**default_filter** = **{'deleted': 'False', 'description': 'All', 'purged': 'False', 'name': 'All'}**

LibraryListGrid.**default_sort_key** = **'name'**

LibraryListGrid.**global_actions** = **[<galaxy.web.framework.helpers.grids.GridAction object at 0x7f4968578410**

LibraryListGrid.**model_class**

alias of Library

LibraryListGrid.**num_rows_per_page** = **50**

LibraryListGrid.**preserve_state** = **False**

LibraryListGrid.**standard_filters** = **[<galaxy.web.framework.helpers.grids.GridColumnFilter object at 0x7f49**

LibraryListGrid.**template** = **'/admin/library/grid.mako'**

LibraryListGrid.**title** = **'Data Libraries'**

LibraryListGrid.**use_paging** = **True**

library_common Module

class galaxy.webapps.galaxy.controllers.library_common.**LibraryCommon**(*app*)

Bases: galaxy.web.base.controller.BaseUIController, galaxy.web.base.controller.UsesFormDe galaxy.web.base.controller.UsesExtendedMetadataMixin, galaxy.web.base.controller.UsesLibraryMixinItems

act_on_multiple_datasets(*trans*, *cntrller*, *library_id=None*, *ldda_ids=''*, ***kwd*)

add_history_datasets_to_library(*trans*, *cntrller*, *library_id*, *folder_id*, *hda_ids=''*, ***kwd*)

browse_library(*trans*, *cntrller='library'*, ***kwd*)

create_folder(*trans*, *cntrller*, *parent_id*, *library_id*, ***kwd*)

delete_library_item(*trans*, *cntrller*, *library_id*, *item_id*, *item_type*, ***kwd*)

download_dataset_from_folder(*trans*, *cntrller*, *id*, *library_id=None*, ***kwd*)

Catches the dataset id and displays file contents as directed

folder_info(*trans*, *cntrller*, *id*, *library_id*, ***kwd*)

folder_permissions(*trans*, *cntrller*, *id*, *library_id*, ***kwd*)

get_path_paste_uploaded_datasets(*trans*, *cntrller*, *params*, *library_bunch*, *response_code*, *message*)

get_server_dir_uploaded_datasets(*trans*, *cntrller*, *params*, *full_dir*, *import_dir_desc*, *library_bunch*, *response_code*, *message*)

import_datasets_to_histories(*trans*, *cntrller*, *library_id=''*, *folder_id=''*, *ldda_ids=''*, *target_history_id=''*, *target_history_ids=''*, *new_history_name=''*, ***kwd*)

ldda_edit_info(*trans*, *cntrller*, *library_id*, *folder_id*, *id*, ***kwd*)

ldda_info(*trans*, *cntrller*, *library_id*, *folder_id*, *id*, ***kwd*)

ldda_permissions(*trans*, *cntrller*, *library_id*, *folder_id*, *id*, ***kwd*)

library_dataset_info(*trans*, *cntrller*, *id*, *library_id*, ***kwd*)

library_dataset_permissions(*trans*, *cntrller*, *id*, *library_id*, ***kwd*)

library_info(*trans*, *cntrller*, ***kwd*)

library_item_updates(*trans*, **args*, ***kwargs*)

library_permissions(*trans*, *cntrller*, ***kwd*)

make_library_item_public(*trans*, *cntrller*, *library_id*, *item_type*, *id*, ***kwd*)

make_library_uploaded_dataset(*trans*, *cntrller*, *params*, *name*, *path*, *type*, *library_bunch*, *in_folder=None*)

manage_template_inheritance(*trans*, *cntrller*, *item_type*, *library_id*, *folder_id=None*, *ldda_id=None*, ***kwd*)

move_library_item(*trans*, *cntrller*, *item_type*, *item_id*, *source_library_id=''*, *make_target_current=True*, ***kwd*)

undelete_library_item(*trans*, *cntrller*, *library_id*, *item_id*, *item_type*, ***kwd*)

upload_dataset(*trans*, *cntrller*, *library_id*, *folder_id*, *replace_dataset=None*, ***kwd*)

upload_library_dataset(*trans*, *cntrller*, *library_id*, *folder_id*, ***kwd*)

galaxy.webapps.galaxy.controllers.library_common.**activatable_folders**(*trans*, *folder*)

galaxy.webapps.galaxy.controllers.library_common.**activatable_folders_and_library_datasets**(*t*
f

galaxy.webapps.galaxy.controllers.library_common.**active_folders**(*trans*, *folder*)

galaxy.webapps.galaxy.controllers.library_common.**active_folders_and_library_datasets**(*trans*, *folder*)

galaxy.webapps.galaxy.controllers.library_common.**branch_deleted**(*folder*)

galaxy.webapps.galaxy.controllers.library_common.**datasets_for_lddas**(*trans*, *lddas*)

Given a list of LDDAs, return a list of Datasets for them.

galaxy.webapps.galaxy.controllers.library_common.**get_comptypes**(*trans*)

galaxy.webapps.galaxy.controllers.library_common.**get_containing_library_from_library_datase**

Given a library_dataset, get the containing library

galaxy.webapps.galaxy.controllers.library_common.**get_sorted_accessible_library_items**(*trans*, *cntrller*, *items*, *sort_att*

galaxy.webapps.galaxy.controllers.library_common.**lucene_search**(*trans*, *cntrller*, *search_term*, *search_url*, ***kwd*)

Return display of results from a full-text lucene search of data libraries.

galaxy.webapps.galaxy.controllers.library_common.**map_library_datasets_to_lddas**(*trans*, *lib_datasets*)

Given a list of LibraryDatasets, return a map from the LibraryDatasets to their LDDAs. If an LDDA does not exist for a LibraryDataset, then there will be no entry in the return hash.

galaxy.webapps.galaxy.controllers.library_common.**sort_by_attr**(*seq*, *attr*)

Sort the sequence of objects by object's attribute Arguments: seq - the list or any sequence (including immutable one) of objects to sort. attr - the name of attribute to sort by

galaxy.webapps.galaxy.controllers.library_common.**whoosh_search**(*trans*, *cntrller*, *search_term*, ***kwd*)

Return display of results from a full-text whoosh search of data libraries.

mobile Module

class galaxy.webapps.galaxy.controllers.mobile.**Mobile**(*app*)

Bases: *galaxy.web.base.controller.BaseUIController*

dataset_detail(*trans*, *id*)

dataset_peek(*trans*, *id*)

history_detail(*trans*, *id*)

history_list(*trans*)

index(*trans*, ***kwargs*)

settings(*trans*, *email=None*, *password=None*)

page Module

class galaxy.webapps.galaxy.controllers.page.**HistoryDatasetAssociationSelectionGrid**

Bases: *galaxy.webapps.galaxy.controllers.page.ItemSelectionGrid*

Grid for selecting HDAs.

apply_query_filter(*trans*, *query*, ***kwargs*)

columns = [<galaxy.webapps.galaxy.controllers.page.NameColumn object at 0x7f4966fba9d0>, <galaxy.web.framework.

model_class

alias of HistoryDatasetAssociation

title = 'Saved Datasets'

class galaxy.webapps.galaxy.controllers.page.**HistorySelectionGrid**

Bases: *galaxy.webapps.galaxy.controllers.page.ItemSelectionGrid*

Grid for selecting histories.

apply_query_filter (*trans*, *query*, ***kwargs*)

columns = [<galaxy.webapps.galaxy.controllers.page.NameColumn object at 0x7f4966f9a1d0>, <galaxy.web.framework.

model_class
alias of `History`

title = 'Saved Histories'

class galaxy.webapps.galaxy.controllers.page.**ItemSelectionGrid**
Bases: *galaxy.web.framework.helpers.grids.Grid*

Base class for pages' item selection grids.

class NameColumn (*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)
Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value (*trans*, *grid*, *item*)

ItemSelectionGrid.**apply_query_filter** (*trans*, *query*, ***kwargs*)

ItemSelectionGrid.**default_filter = {'deleted': 'False', 'sharing': 'All'}**

ItemSelectionGrid.**default_sort_key = '-update_time'**

ItemSelectionGrid.**num_rows_per_page = 10**

ItemSelectionGrid.**show_item_checkboxes = True**

ItemSelectionGrid.**template = '/page/select_items_grid.mako'**

ItemSelectionGrid.**use_async = True**

ItemSelectionGrid.**use_paging = True**

class galaxy.webapps.galaxy.controllers.page.**PageAllPublishedGrid**
Bases: *galaxy.web.framework.helpers.grids.Grid*

apply_query_filter (*trans*, *query*, ***kwargs*)

build_initial_query (*trans*, ***kwargs*)

columns = [<galaxy.web.framework.helpers.grids.PublicURLColumn object at 0x7f4966fbab10>, <galaxy.web.framewor

default_filter = {'username': 'All', 'title': 'All'}

default_sort_key = 'update_time'

model_class
alias of `Page`

title = 'Published Pages'

use_async = True

use_panels = True

class galaxy.webapps.galaxy.controllers.page.**PageController** (*app*)
Bases: *galaxy.web.base.controller.BaseUIController*, *galaxy.web.base.controller.SharableMix*
galaxy.web.base.controller.UsesStoredWorkflowMixin, *galaxy.web.base.controller.UsesVisua*
galaxy.model.item_attrs.UsesItemRatings

create (*trans*, **args*, ***kwargs*)
Create a new page

display (*trans*, **args*, ***kwargs*)

display_by_username_and_slug(*trans*, *username*, *slug*)
Display page based on a username and slug.

edit(*trans*, **args*, ***kwargs*)
Edit a page's attributes.

edit_content(*trans*, **args*, ***kwargs*)
Render the main page editor interface.

get_editor_iframe(*trans*)
Returns the document for the page editor's iframe.

get_embed_html_async(*trans*, *id*)
Returns HTML for embedding a workflow in a page.

get_item(*trans*, *id*)

get_name_and_link_async(*trans*, **args*, ***kwargs*)
Returns page's name and link.

get_page(*trans*, *id*, *check_ownership=True*, *check_accessible=False*)
Get a page from the database by id.

list(*trans*, **args*, ***kwargs*)
List user's pages.

list_datasets_for_selection(*trans*, **args*, ***kwargs*)
Returns HTML that enables a user to select one or more datasets.

list_histories_for_selection(*trans*, **args*, ***kwargs*)
Returns HTML that enables a user to select one or more histories.

list_pages_for_selection(*trans*, **args*, ***kwargs*)
Returns HTML that enables a user to select one or more pages.

list_published(*trans*, **args*, ***kwargs*)

list_visualizations_for_selection(*trans*, **args*, ***kwargs*)
Returns HTML that enables a user to select one or more visualizations.

list_workflows_for_selection(*trans*, **args*, ***kwargs*)
Returns HTML that enables a user to select one or more workflows.

rate_async(*trans*, **args*, ***kwargs*)
Rate a page asynchronously and return updated community data.

save(*trans*, **args*, ***kwargs*)

set_accessible_async(*trans*, **args*, ***kwargs*)
Set page's importable attribute and slug.

share(*trans*, **args*, ***kwargs*)
Handle sharing with an individual user.

sharing(*trans*, **args*, ***kwargs*)
Handle page sharing.

class galaxy.webapps.galaxy.controllers.page.**PageListGrid**
Bases: *galaxy.web.framework.helpers.grids.Grid*

class URLColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)
Bases: *galaxy.web.framework.helpers.grids.PublicURLColumn*

get_value(*trans*, *grid*, *item*)

PageListGrid.**apply_query_filter**(*trans*, *query*, ***kwargs*)

PageListGrid.**columns = [<galaxy.web.framework.helpers.grids.TextColumn object at 0x7f4966fbafd0>, <galaxy.wel**

PageListGrid.**default_filter = {'title': 'All', 'sharing': 'All', 'tags': 'All', 'published': 'All'}**

PageListGrid.**default_sort_key = '-update_time'**

PageListGrid.**global_actions = [<galaxy.web.framework.helpers.grids.GridAction object at 0x7f4966fba110>]**

PageListGrid.**model_class**
alias of Page

PageListGrid.**operations = [<galaxy.web.framework.helpers.grids.DisplayByUsernameAndSlugGridOperation obj**

PageListGrid.**title = 'Pages'**

PageListGrid.**use_panels = True**

class galaxy.webapps.galaxy.controllers.page.**PageSelectionGrid**
Bases: *galaxy.webapps.galaxy.controllers.page.ItemSelectionGrid*

Grid for selecting pages.

columns = [<galaxy.web.framework.helpers.grids.TextColumn object at 0x7f4966f9a6d0>, <galaxy.web.framework.helpe

model_class
alias of Page

title = 'Saved Pages'

class galaxy.webapps.galaxy.controllers.page.**VisualizationSelectionGrid**
Bases: *galaxy.webapps.galaxy.controllers.page.ItemSelectionGrid*

Grid for selecting visualizations.

columns = [<galaxy.web.framework.helpers.grids.TextColumn object at 0x7f4966ecdb90>, <galaxy.web.framework.helpe

model_class
alias of Visualization

title = 'Saved Visualizations'

class galaxy.webapps.galaxy.controllers.page.**WorkflowSelectionGrid**
Bases: *galaxy.webapps.galaxy.controllers.page.ItemSelectionGrid*

Grid for selecting workflows.

columns = [<galaxy.webapps.galaxy.controllers.page.NameColumn object at 0x7f4966f9a590>, <galaxy.web.framework.h

model_class
alias of StoredWorkflow

title = 'Saved Workflows'

galaxy.webapps.galaxy.controllers.page.**format_bool**(*b*)

request_type Module

class galaxy.webapps.galaxy.controllers.request_type.**RequestType**(*app*)
Bases: *galaxy.web.base.controller.BaseUIController*, *galaxy.web.base.controller.UsesFormDef*

browse_request_types(*trans*, **args*, ***kwargs*)

create_request_type(*trans*, **args*, ***kwargs*)

delete_request_type(*trans*, **args*, ***kwargs*)

edit_request_type(*trans*, **args*, ***kwargs*)

request_type_grid = <galaxy.webapps.galaxy.controllers.request_type.RequestTypeGrid object>

request_type_permissions(*trans*, **args*, ***kwargs*)

undelete_request_type(*trans*, **args*, ***kwargs*)

view_editable_request_type(*trans*, **args*, ***kwargs*)

view_form_definition(*trans*, **args*, ***kwargs*)

view_request_type(*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.controllers.request_type.**RequestTypeGrid**

Bases: *galaxy.web.framework.helpers.grids.Grid*

class DescriptionColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *request_type*)

class RequestTypeGrid.**ExternalServiceColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.IntegerColumn*

get_value(*trans*, *grid*, *request_type*)

class RequestTypeGrid.**NameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *request_type*)

class RequestTypeGrid.**RequestFormColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *request_type*)

class RequestTypeGrid.**SampleFormColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *request_type*)

RequestTypeGrid.**columns = [<galaxy.webapps.galaxy.controllers.request_type.NameColumn object at 0x7f4966fba**

RequestTypeGrid.**default_filter = {'deleted': 'False'}**

RequestTypeGrid.**default_sort_key** = '-create_time'

RequestTypeGrid.**global_actions** = [<galaxy.web.framework.helpers.grids.GridAction object at 0x7f4965c0ee10>

RequestTypeGrid.**model_class**
alias of *RequestType*

RequestTypeGrid.**num_rows_per_page** = 50

RequestTypeGrid.**operations** = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f4965c0ecd0>,

RequestTypeGrid.**preserve_state** = True

RequestTypeGrid.**template** = 'admin/request_type/grid.mako'

RequestTypeGrid.**title** = 'Request Types'

RequestTypeGrid.**use_paging** = True

requests Module

class galaxy.webapps.galaxy.controllers.requests.**Requests**(*app*)
Bases: *galaxy.web.base.controller.BaseUIController*

browse_requests(*trans*, ***kwd*)

find_samples_index(*trans*, **args*, ***kwargs*)

index(*trans*, **args*, ***kwargs*)

request_grid = <galaxy.webapps.galaxy.controllers.requests.UserRequestsGrid object>

class galaxy.webapps.galaxy.controllers.requests.**UserRequestsGrid**
Bases: *galaxy.webapps.galaxy.controllers.requests_common.RequestsGrid*

apply_query_filter(*trans*, *query*, ***kwd*)

operation = <galaxy.web.framework.helpers.grids.GridOperation object>

operations = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f49684d1a90>, <galaxy.web.framewor

requests_admin Module

class galaxy.webapps.galaxy.controllers.requests_admin.**AdminRequestsGrid**
Bases: *galaxy.webapps.galaxy.controllers.requests_common.RequestsGrid*

class UserColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)
Bases: *galaxy.web.framework.helpers.grids.TextColumn*

get_value(*trans*, *grid*, *request*)

AdminRequestsGrid.**col** = <galaxy.web.framework.helpers.grids.MulticolFilterColumn object>

AdminRequestsGrid.**columns** = [<galaxy.webapps.galaxy.controllers.requests_common.NameColumn object at 0x7f

AdminRequestsGrid.**global_actions** = [<galaxy.web.framework.helpers.grids.GridAction object at 0x7f49775852

AdminRequestsGrid.**operation** = <galaxy.web.framework.helpers.grids.GridOperation object>

AdminRequestsGrid.**operations** = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f49684d1a90

class galaxy.webapps.galaxy.controllers.requests_admin.**DataTransferGrid**
Bases: *galaxy.web.framework.helpers.grids.Grid*

class **ExternalServiceColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *sample_dataset*)

class DataTransferGrid.**NameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *sample_dataset*)

class DataTransferGrid.**SizeColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *sample_dataset*)

class DataTransferGrid.**StatusColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *sample_dataset*)

DataTransferGrid.**apply_query_filter**(*trans*, *query*, ***kwd*)

DataTransferGrid.**columns = [<galaxy.webapps.galaxy.controllers.requests_admin.NameColumn object at 0x7f497**

DataTransferGrid.**default_sort_key = '-create_time'**

DataTransferGrid.**model_class**

alias of SampleDataset

DataTransferGrid.**num_rows_per_page = 50**

DataTransferGrid.**operations = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f4977585910>**

DataTransferGrid.**preserve_state = True**

DataTransferGrid.**template = 'admin/requests/sample_datasets_grid.mako'**

DataTransferGrid.**title = 'Sample Datasets'**

DataTransferGrid.**use_paging = False**

class galaxy.webapps.galaxy.controllers.requests_admin.**RequestsAdmin**(*app*)

Bases: galaxy.web.base.controller.BaseUIController, galaxy.web.base.controller.UsesFormDe

browse_requests(*trans*, **args*, ***kwargs*)

datatx_grid = <galaxy.webapps.galaxy.controllers.requests_admin.DataTransferGrid object>

get_file_details(*trans*, **args*, ***kwargs*)

index(*trans*, **args*, ***kwargs*)

initiate_data_transfer(*trans*, **args*, ***kwargs*)

manage_datasets(*trans*, **args*, ***kwargs*)

open_folder(*trans*, **args*, ***kwargs*)

reject_request(*trans*, **args*, ***kwargs*)

rename_datasets(*trans*, **args*, ***kwargs*)

request_grid = <galaxy.webapps.galaxy.controllers.requests_admin.AdminRequestsGrid object>

select_datasets_to_transfer(*trans*, **args*, ***kwargs*)

update_sample_dataset_status(*trans*, *cntrller*, *sample_dataset_ids*, *new_status*, *error_msg=None*)

galaxy.webapps.galaxy.controllers.requests_admin.**build_rename_datasets_for_sample_select_f**

requests_common Module

class galaxy.webapps.galaxy.controllers.requests_common.**RequestsCommon**(*app*)

Bases: *galaxy.web.base.controller.BaseUIController*, *galaxy.web.base.controller.UsesFormDe*

add_sample(*trans*, **args*, ***kwargs*)

add_samples(*trans*, **args*, ***kwargs*)

create_request(*trans*, **args*, ***kwargs*)

dataset_transfer_status_updates(*trans*, **args*, ***kwargs*)

delete_request(*trans*, **args*, ***kwargs*)

delete_sample(*trans*, **args*, ***kwargs*)

edit_basic_request_info(*trans*, **args*, ***kwargs*)

edit_email_settings(*trans*, **args*, ***kwargs*)

Allow for changing the email notification settings where email is sent to a list of users whenever the request state changes to one selected for notification.

edit_samples(*trans*, **args*, ***kwargs*)

find_samples(*trans*, **args*, ***kwargs*)

sample_datasets_updates(*trans*, **args*, ***kwargs*)

sample_state_updates(*trans*, **args*, ***kwargs*)

submit_request(*trans*, **args*, ***kwargs*)

undelete_request(*trans*, **args*, ***kwargs*)

update_request_state(*trans*, **args*, ***kwargs*)

update_sample_state(*trans*, *cntrller*, *sample_ids*, *new_state*, *comment=None*)

view_request(*trans*, **args*, ***kwargs*)

view_request_history(*trans*, **args*, ***kwargs*)

view_sample(*trans*, **args*, ***kwargs*)

view_sample_datasets(*trans*, **args*, ***kwargs*)

view_sample_history(*trans*, **args*, ***kwargs*)

class galaxy.webapps.galaxy.controllers.requests_common.**RequestsGrid**

Bases: galaxy.web.framework.helpers.grids.Grid

class DescriptionColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *request*)

class RequestsGrid.**NameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *request*)

class RequestsGrid.**SamplesColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *request*)

class RequestsGrid.**StateColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.StateColumn

filter(*trans*, *user*, *query*, *column_filter*)

Modify query to filter request by state.

get_value(*trans*, *grid*, *request*)

class RequestsGrid.**TypeColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *request*)

RequestsGrid.**columns = [<galaxy.webapps.galaxy.controllers.requests_common.NameColumn object at 0x7f496bb2**

RequestsGrid.**default_filter = {'deleted': 'False', 'state': 'All'}**

RequestsGrid.**default_sort_key = '-update_time'**

RequestsGrid.**model_class**

alias of Request

RequestsGrid.**num_rows_per_page = 50**

RequestsGrid.**operations = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f49684d1a90>]**

RequestsGrid.**template = 'requests/grid.mako'**

RequestsGrid.**title = 'Sequencing Requests'**

RequestsGrid.**use_paging** = **True**

galaxy.webapps.galaxy.controllers.requests_common.**invalid_id_redirect**(*trans*, *controller*, *obj_id*, *item='sequencing request'*, *action='browse_requests'*)

root Module Contains the main interface in the Universe class

class galaxy.webapps.galaxy.controllers.root.**RootController**(*app*)

Bases: galaxy.web.base.controller.BaseUIController, galaxy.model.item_attrs.UsesAnnotation

Controller class that maps to the url root of Galaxy (i.e. '/').

bucket_proxy(*trans*, *bucket=None*, ***kwd*)

clear_history(*trans*)

Clears the history for a user.

dataset_make_primary(*trans*, *id=None*)

Copies a dataset and makes primary.

default(*trans*, *target1=None*, *target2=None*, ***kwd*)

Called on any url that does not match a controller method.

display(*trans*, *id=None*, *hid=None*, *tofile=None*, *toext='.txt'*, *encoded_id=None*, ***kwd*)

Returns data directly into the browser.

Sets the mime-type according to the extension.

Used by the twill tool test driver - used anywhere else? Would like to drop hid argument and path if unneeded now. Likewise, would like to drop encoded_id=XXX and use assume id is encoded (likely id wouldn't be coming in encoded if this is used anywhere else though.)

display_as(*trans*, *id=None*, *display_app=None*, ***kwd*)

Returns a file in a format that can successfully be displayed in display_app.

display_child(*trans*, *parent_id=None*, *designation=None*, *tofile=None*, *toext='.txt'*)

Returns child data directly into the browser, based upon parent_id and designation.

echo(*trans*, ***kwd*)

Echos parameters (debugging).

echo_json(*trans*, **args*, ***kwargs*)

Echos parameters as JSON (debugging).

Attempts to parse values passed as boolean, float, then int. Defaults to string. Non-recursive (will not parse lists).

generate_error(*trans*, *code=500*)

Raises an exception (debugging).

generate_json_error(*trans*, **args*, ***kwargs*)

Raises an exception (debugging).

history(*trans*, *as_xml=False*, *show_deleted=None*, *show_hidden=None*, ***kwd*)

Display the current history in its own page or as xml.

history_add_to(*trans*, *history_id=None*, *file_data=None*, *name='Data Added to History'*, *info=None*, *ext='txt'*, *dbkey='?'*, *copy_access_from=None*, ***kwd*)
Adds a POSTed file to a History.

history_as_xml(*trans*, *show_deleted=None*, *show_hidden=None*)

history_delete(*trans*, *id*)
Backward compatibility with check_galaxy script.

history_import(*trans*, *id=None*, *confirm=False*, ***kwd*)

history_new(*trans*, *name=None*)
Create a new history with the given name and refresh the history panel.

history_options(*trans*)
Displays a list of history related actions.

history_set_default_permissions(*trans*, *id=None*, ***kwd*)
Sets the permissions on a history.

index(*trans*, *id=None*, *tool_id=None*, *mode=None*, *workflow_id=None*, *m_c=None*, *m_a=None*, ***kwd*)
Called on the root url to display the main Galaxy page.

peek(*trans*, *id=None*)
Returns a 'peek' at the data.

tool_help(*trans*, *id*)
Return help page for tool identified by 'id' if available

tool_search(*trans*, **args*, ***kwargs*)
Searches the tool database and returns data for any tool whose text matches the query.

Data are returned in JSON format.

welcome(*trans*)

tag Module Tags Controller: handles tagging/untagging of entities and provides autocomplete support.

class galaxy.webapps.galaxy.controllers.tag.**TagsController**(*app*)
Bases: galaxy.web.base.controller.BaseUIController, galaxy.web.base.controller.UsesTagsMix

add_tag_async(*trans*, **args*, ***kwargs*)
Add tag to an item.

get_tagging_elt_async(*trans*, **args*, ***kwargs*)
Returns HTML for editing an item's tags.

remove_tag_async(*trans*, **args*, ***kwargs*)
Remove tag from an item.

retag_async(*trans*, **args*, ***kwargs*)
Apply a new set of tags to an item; previous tags are deleted.

tag_autocomplete_data(*trans*, **args*, ***kwargs*)
Get autocomplete data for an item's tags.

tool_runner Module Upload class

class galaxy.webapps.galaxy.controllers.tool_runner.**AddFrameData**

class galaxy.webapps.galaxy.controllers.tool_runner.**ToolRunner**(*app*)
Bases: galaxy.web.base.controller.BaseUIController

biomart(*trans*, *tool_id='biomart'*, ***kwd*)
Catches the tool id and redirects as needed

data_source_redirect(*trans*, *tool_id=None*)
Redirects a user accessing a Data Source tool to its target action link. This method will subvert mix-mode content blocking in several browsers when accessing non-https data_source tools from an https galaxy server.

Tested as working on Safari 7.0 and FireFox 26 Subverting did not work on Chrome 31

default(*trans*, *tool_id=None*, ***kwd*)
Catches the tool id and redirects as needed

hapmapmart(*trans*, *tool_id='hapmapmart'*, ***kwd*)
Catches the tool id and redirects as needed

index(*trans*, *tool_id=None*, *from_noframe=None*, ***kwd*)

redirect(*trans*, *redirect_url=None*, ***kwd*)

rerun(*trans*, *id=None*, *from_noframe=None*, *job_id=None*, ***kwd*)
Given a HistoryDatasetAssociation id, find the job and that created the dataset, extract the parameters, and display the appropriate tool form with parameters already filled in.

upload_async_create(*trans*, **args*, ***kwargs*)
Precreate datasets for asynchronous uploading.

upload_async_message(*trans*, ***kwd*)

ucsc_proxy Module Contains the UCSC proxy

class galaxy.webapps.galaxy.controllers.ucsc_proxy.**UCSCProxy**(*app*)
Bases: galaxy.web.base.controller.BaseUIController

create_display(*store*)
Creates a more meaningulf display name

index(*trans*, *init=False*, ***kwd*)

user Module Contains the user interface in the Universe class

class galaxy.webapps.galaxy.controllers.user.**User**(*app*)
Bases: galaxy.web.base.controller.BaseUIController, galaxy.web.base.controller.UsesFormDe
galaxy.web.base.controller.CreatesUsersMixin, galaxy.web.base.controller.CreatesApiKeysM

activate(*trans*, ***kwd*)
Check whether token fits the user and then activate the user's account.

api_keys(*trans*, **args*, ***kwargs*)

change_password(*trans*, *token=None*, ***kwd*)
Provides a form with which one can change their password. If token is provided, don't require current password.

create(*trans*, *cntrller='user'*, *redirect_url=''*, *refresh_frames=[]*, ***kwd*)

dbkeys(*trans*, **args*, ***kwargs*)
Handle custom builds.

delete_address (*trans*, **args*, ***kwargs*)

edit_address (*trans*, **args*, ***kwargs*)

edit_info (*trans*, *cntrller*, ***kwd*)
Edit user information = username, email or password.

edit_toolbox_filters (*trans*, **args*, ***kwargs*)

edit_username (*trans*, **args*, ***kwargs*)

get_activation_token (*trans*, *email*)
Check for the activation token. Create new activation token and store it in the database if no token found.

get_most_recently_used_tool_async (*trans*, **args*, ***kwargs*)
Returns information about the most recently used tool.

index (*trans*, *cntrller*, ***kwd*)

installed_len_files = None

is_outside_grace_period (*trans*, *create_time*)
Function checks whether the user is outside the config-defined grace period for inactive accounts.

log_user_action_async (*trans*, *action*, *context*, *params*)
Log a user action asynchronously. If user is not logged in, do nothing.

login (*trans*, *refresh_frames=[]*, ***kwd*)
Handle Galaxy Log in

logout (*trans*, *logout_all=False*)

manage_addresses (*trans*, ***kwd*)

manage_user_info (*trans*, *cntrller*, ***kwd*)
Manage a user's login, password, public username, type, addresses, etc.

new_address (*trans*, **args*, ***kwargs*)

openid_associate (*trans*, *cntrller='user'*, ***kwd*)
Associates a user with an OpenID log in

openid_auth (*trans*, ***kwd*)
Handles user request to access an OpenID provider

openid_disassociate (*trans*, **args*, ***kwargs*)
Disassociates a user with an OpenID

openid_manage (*trans*, **args*, ***kwargs*)
Manage OpenIDs for user

openid_process (*trans*, ***kwd*)
Handle's response from OpenID Providers

prepare_activation_link (*trans*, *email*)
Prepare the account activation link for the user.

proceed_login (*trans*, *user*, *redirect*)
Function processes user login. It is called in case all the login requirements are valid.

resend_verification (*trans*)
Exposed function for use outside of the class. E.g. when user click on the resend link in the masthead.

resend_verification_email (*trans*, *email*, *username*)
Function resends the verification email in case user wants to log in with an inactive account or he clicks the resend link.

reset_password(*trans*, *email=None*, ***kwd*)
Reset the user's password. Send an email with token that allows a password change.

send_verification_email(*trans*, *email*, *username*)
Send the verification email containing the activation link to the user's email.

set_default_permissions(*trans*, *cntrller*, ***kwd*)
Set the user's default permissions for the new histories

set_user_pref_async(*trans*, *pref_name*, *pref_value*)
Set a user preference asynchronously. If user is not logged in, do nothing.

toolbox_filters(*trans*, **args*, ***kwargs*)
Sets the user's default filters for the toolbox. Toolbox filters are specified in galaxy.ini. The user can activate them and the choice is stored in user_preferences.

undelete_address(*trans*, **args*, ***kwargs*)

user_openid_grid = <galaxy.webapps.galaxy.controllers.user.UserOpenIDGrid object>

class galaxy.webapps.galaxy.controllers.user.**UserOpenIDGrid**
Bases: *galaxy.web.framework.helpers.grids.Grid*

build_initial_query(*trans*, ***kwd*)

columns = [<galaxy.web.framework.helpers.grids.TextColumn object at 0x7f49666b0b10>, <galaxy.web.framework.help

default_filter = {'openid': 'All'}

default_sort_key = '-create_time'

model_class
alias of UserOpenID

operations = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f4975bb2190>]

template = '/user/openid_manage.mako'

title = 'OpenIDs linked to your account'

use_panels = False

visualization Module

workflow Module

class galaxy.webapps.galaxy.controllers.workflow.**SingleTagContentsParser**(*target_tag*)
Bases: sgmllib.SGMLParser

handle_data(*text*)
Called for each block of plain text.

unknown_starttag(*tag*, *attrs*)
Called for each start tag.

class galaxy.webapps.galaxy.controllers.workflow.**StoredWorkflowAllPublishedGrid**
Bases: *galaxy.web.framework.helpers.grids.Grid*

apply_query_filter(*trans*, *query*, ***kwargs*)

build_initial_query(*trans*, ***kwargs*)

columns = [<galaxy.web.framework.helpers.grids.PublicURLColumn object at 0x7f496b28d7d0>, <galaxy.web.framewo

default_filter = {'username': 'All', 'public_url': 'All', 'tags': 'All'}

default_sort_key = 'update_time'

model_class
alias of StoredWorkflow

operations = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f49780e7d50>, <galaxy.web.framewor

title = 'Published Workflows'

use_async = True

class galaxy.webapps.galaxy.controllers.workflow.**StoredWorkflowListGrid**
Bases: galaxy.web.framework.helpers.grids.Grid

class StepsColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)
Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *workflow*)

StoredWorkflowListGrid.**apply_query_filter**(*trans*, *query*, ***kwargs*)

StoredWorkflowListGrid.**columns** = [<galaxy.web.framework.helpers.grids.TextColumn object at 0x7f496b28d3d

StoredWorkflowListGrid.**default_filter** = {'name': 'All', 'tags': 'All'}

StoredWorkflowListGrid.**default_sort_key** = '-update_time'

StoredWorkflowListGrid.**model_class**
alias of StoredWorkflow

StoredWorkflowListGrid.**operations** = [<galaxy.web.framework.helpers.grids.GridOperation object at 0x7f496b

StoredWorkflowListGrid.**title** = 'Saved Workflows'

StoredWorkflowListGrid.**use_panels** = True

class galaxy.webapps.galaxy.controllers.workflow.**WorkflowController**(*app*)
Bases: galaxy.web.base.controller.BaseUIController, galaxy.web.base.controller.SharableMix
galaxy.web.base.controller.UsesStoredWorkflowMixin, galaxy.model.item_attrs.UsesItemRat

annotate_async(*trans*, **args*, ***kwargs*)

build_from_current_history(*trans*, *job_ids=None*, *dataset_ids=None*, *dataset_collection_ids=None*, *workflow_name=None*)

configure_menu(*trans*, *workflow_ids=None*)

copy(*trans*, **args*, ***kwargs*)

create(*trans*, **args*, ***kwargs*)
Create a new stored workflow with name *workflow_name*.

delete(*trans*, *id=None*)
Mark a workflow as deleted

display_by_id(*trans*, *id*)
Display workflow based on id.

display_by_username_and_slug(*trans*, *username*, *slug*, *format='html'*)
Display workflow based on a username and slug. Format can be html, json, or json-download.

editor(*trans*, **args*, ***kwargs*)
Render the main workflow editor interface. The canvas is embedded as an iframe (necessary for scrolling to work properly), which is rendered by *editor_canvas*.

editor_form_post (*trans*, **args*, ***kwargs*)
Accepts a tool state and incoming values, and generates a new tool form and some additional information, packed into a json dictionary. This is used for the form shown in the right pane when a node is selected.

export (*trans*, **args*, ***kwargs*)
Handles download/export workflow command.

export_to_file (*trans*, **args*, ***kwargs*)
Get the latest Workflow for the StoredWorkflow identified by *id* and encode it as a json string that can be imported back into Galaxy

This has slightly different information than the above. In particular, it does not attempt to decode forms and build UIs, it just stores the raw state.

export_to_myexp (*trans*, **args*, ***kwargs*)
Exports a workflow to myExperiment website.

for_direct_import (*trans*, **args*, ***kwargs*)
Get the latest Workflow for the StoredWorkflow identified by *id* and encode it as a json string that can be imported back into Galaxy

This has slightly different information than the above. In particular, it does not attempt to decode forms and build UIs, it just stores the raw state.

gen_image (*trans*, **args*, ***kwargs*)

get_embed_html_async (*trans*, *id*)
Returns HTML for embedding a workflow in a page.

get_item (*trans*, *id*)

get_item_content_async (*trans*, *id*)
Returns item content in HTML format.

get_name_and_link_async (*trans*, **args*, ***kwargs*)
Returns workflow's name and link.

get_new_module_info (*trans*, **args*, ***kwargs*)
Get the info for a new instance of a module initialized with default parameters (any keyword arguments will be passed along to the module). Result includes data inputs and outputs, html representation of the initial form, and the initial tool state (with default values). This is called asynchronously whenever a new node is added.

imp (*trans*, **args*, ***kwargs*)
Imports a workflow shared by other users.

import_workflow (*trans*, *cntrller='workflow'*, ***kwd*)
Import a workflow by reading an url, uploading a file, opening and reading the contents of a local file, or receiving the textual representation of a workflow via http.

index (*trans*)

list (*trans*, **args*, ***kwargs*)
Render workflow main page (management of existing workflows)

list_for_run (*trans*, **args*, ***kwargs*)
Render workflow list for analysis view (just allows running workflow or switching to management view)

list_grid (*trans*, **args*, ***kwargs*)
List user's stored workflows.

list_published (*trans*, ***kwargs*)

load_workflow(*trans*, **args*, ***kwargs*)
Get the latest Workflow for the StoredWorkflow identified by *id* and encode it as a json string that can be read by the workflow editor web interface.

published_list_grid = <galaxy.webapps.galaxy.controllers.workflow.StoredWorkflowAllPublishedGrid object>

rate_async(*trans*, **args*, ***kwargs*)
Rate a workflow asynchronously and return updated community data.

rename(*trans*, **args*, ***kwargs*)

rename_async(*trans*, **args*, ***kwargs*)

run(*trans*, *id*, *history_id=None*, *hide_fixed_params=False*, ***kwargs*)

save_workflow(*trans*, **args*, ***kwargs*)
Save the workflow described by *workflow_data* with id *id*.

set_accessible_async(*trans*, **args*, ***kwargs*)
Set workflow's importable attribute and slug.

share(*trans*, **args*, ***kwargs*)

sharing(*trans*, **args*, ***kwargs*)
Handle workflow sharing.

stored_list_grid = <galaxy.webapps.galaxy.controllers.workflow.StoredWorkflowListGrid object>

tag_outputs(*trans*, *id*, ***kwargs*)

reports Package

reports Package The Galaxy Reports application.

app Module

class galaxy.webapps.reports.app.**UniverseApplication**(***kwargs*)
Bases: object

Encapsulates the state of a Universe application

shutdown()

buildapp Module Provides factory methods to assemble the Galaxy web application

class galaxy.webapps.reports.buildapp.**ReportsWebApplication**(*galaxy_app*, *session_cookie='galaxysession'*, *name=None*)
Bases: galaxy.web.framework.webapp.WebApplication

galaxy.webapps.reports.buildapp.**add_ui_controllers**(*webapp*, *app*)
Search for controllers in the 'galaxy.webapps.controllers' module and add them to the webapp.

galaxy.webapps.reports.buildapp.**app_factory**(*global_conf*, ***kwargs*)
Return a wsgi application serving the root object

galaxy.webapps.reports.buildapp.**build_template_error_formatters**()
Build a list of template error formatters for WebError. When an error occurs, WebError pass the exception to each function in this list until one returns a value, which will be displayed on the error page.

galaxy.webapps.reports.buildapp.**wrap_in_middleware**(*app*, *global_conf*, ***local_conf*)
Based on the configuration wrap *app* in a set of common and useful middleware.

galaxy.webapps.reports.buildapp.**wrap_in_static**(*app*, *global_conf*, ***local_conf*)

config Module Universe configuration builder.

class galaxy.webapps.reports.config.**Configuration**(***kwargs*)
Bases: object

check()

get(*key*, *default*)

exception galaxy.webapps.reports.config.**ConfigurationError**
Bases: exceptions.Exception

galaxy.webapps.reports.config.**configure_logging**(*config*)
Allow some basic logging configuration to be read from the cherrpy config.

galaxy.webapps.reports.config.**get_database_engine_options**(*kwargs*)
Allow options for the SQLAlchemy database engine to be passed by using the prefix "database_engine_option".

galaxy.webapps.reports.config.**resolve_path**(*path*, *root*)
If 'path' is relative make absolute by prepending 'root'

Subpackages

controllers Package

controllers Package Galaxy reports controllers.

jobs Module

class galaxy.webapps.reports.controllers.jobs.**Jobs**(*app*)
Bases: galaxy.web.base.controller.BaseUIController, galaxy.webapps.reports.controllers.que

Class contains functions for querying data requested by user via the webapp. It exposes the functions and responds to requests with the filled .mako templates.

job_info(*trans*, ***kwd*)

per_month_all(*trans*, ***kwd*)
Queries the DB for all jobs. Avoids monitor jobs.

per_month_in_error(*trans*, ***kwd*)
Queries the DB for user jobs in error. Filters out monitor jobs.

per_tool(*trans*, ***kwd*)

per_user(*trans*, ***kwd*)

specified_date_handler(*trans*, ***kwd*)

specified_date_list_grid = <galaxy.webapps.reports.controllers.jobs.SpecifiedDateListGrid object>

specified_month_all(*trans*, ***kwd*)
Queries the DB for all jobs in given month, defaults to current month.

specified_month_in_error(*trans*, ***kwd*)
Queries the DB for the user jobs in error.

tool_per_month(*trans*, ***kwd*)

user_per_month(*trans*, ***kwd*)

class galaxy.webapps.reports.controllers.jobs.**SpecifiedDateListGrid**

Bases: galaxy.web.framework.helpers.grids.Grid

class CreateTimeColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.DateTimeColumn

get_value(*trans*, *grid*, *job*)

class SpecifiedDateListGrid.**EmailColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

filter(*trans*, *user*, *query*, *column_filter*)

class SpecifiedDateListGrid.**JobIdColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.IntegerColumn

get_value(*trans*, *grid*, *job*)

class SpecifiedDateListGrid.**SpecifiedDateColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

filter(*trans*, *user*, *query*, *column_filter*)

class SpecifiedDateListGrid.**StateColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

filter(*trans*, *user*, *query*, *column_filter*)

get_value(*trans*, *grid*, *job*)

class SpecifiedDateListGrid.**ToolColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *job*)

class SpecifiedDateListGrid.**UserColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

get_value(*trans*, *grid*, *job*)

SpecifiedDateListGrid.**build_initial_query**(*trans*, ***kwd*)

SpecifiedDateListGrid.**columns = [<galaxy.webapps.reports.controllers.jobs.JobIdColumn object at 0x7f4964ab3**

SpecifiedDateListGrid.**default_filter = {'specified_date': 'All'}**

SpecifiedDateListGrid.**default_sort_key = 'id'**

SpecifiedDateListGrid.**model_class**

alias of Job

SpecifiedDateListGrid.**num_rows_per_page = 50**

SpecifiedDateListGrid.**preserve_state = False**

SpecifiedDateListGrid.**standard_filters = []**

SpecifiedDateListGrid.**template = '/webapps/reports/grid.mako'**

SpecifiedDateListGrid.**title = 'Jobs'**

SpecifiedDateListGrid.**use_async = False**

SpecifiedDateListGrid.**use_paging = True**

galaxy.webapps.reports.controllers.jobs.**get_job**(*trans*, *id*)

galaxy.webapps.reports.controllers.jobs.**get_monitor_id**(*trans*, *monitor_email*)

A convenience method to obtain the monitor job id.

root Module

class galaxy.webapps.reports.controllers.root.**Report**(*app*)

Bases: galaxy.web.base.controller.BaseUIController

index(*trans*, ***kwd*)

sample_tracking Module

class galaxy.webapps.reports.controllers.sample_tracking.**SampleTracking**(*app*)

Bases: galaxy.web.base.controller.BaseUIController, galaxy.webapps.reports.controllers.que

per_month_all(*trans*, ***kwd*)

per_user(*trans*, ***kwd*)

specified_date_handler(*trans*, ***kwd*)

specified_date_list_grid = <galaxy.webapps.reports.controllers.sample_tracking.SpecifiedDateListGrid object>

user_per_month(*trans*, ***kwd*)

class galaxy.webapps.reports.controllers.sample_tracking.**SpecifiedDateListGrid**

Bases: galaxy.web.framework.helpers.grids.Grid

class **CreateTimeColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.DateTimeColumn

get_value(*trans*, *grid*, *request*)

class SpecifiedDateListGrid.**EmailColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

filter(*trans*, *user*, *query*, *column_filter*)

class SpecifiedDateListGrid.**RequestNameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *request*)

class SpecifiedDateListGrid.**SpecifiedDateColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

filter(*trans*, *user*, *query*, *column_filter*)

class SpecifiedDateListGrid.**UserColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *request*)

SpecifiedDateListGrid.**build_initial_query**(*trans*, ***kwd*)

SpecifiedDateListGrid.**columns = [<galaxy.webapps.reports.controllers.sample_tracking.RequestNameColumn o**

SpecifiedDateListGrid.**default_filter = {'specified_date': 'All'}**

SpecifiedDateListGrid.**default_sort_key = 'name'**

SpecifiedDateListGrid.**model_class**

alias of Request

SpecifiedDateListGrid.**num_rows_per_page = 50**

SpecifiedDateListGrid.**preserve_state = False**

SpecifiedDateListGrid.**standard_filters = []**

SpecifiedDateListGrid.**template** = '/webapps/reports/grid.mako'

SpecifiedDateListGrid.**title** = 'Sequencing Requests'

SpecifiedDateListGrid.**use_async** = False

SpecifiedDateListGrid.**use_paging** = True

galaxy.webapps.reports.controllers.sample_tracking.**get_request**(*trans*, *id*)

system Module

class galaxy.webapps.reports.controllers.system.**System**(*app*)

Bases: *galaxy.web.base.controller.BaseUIController*

dataset_info(*trans*, ***kwd*)

deleted_datasets(*trans*, ***kwd*)

The number of datasets that were deleted more than the specified number of days ago, but have not yet been purged.

deleted_histories(*trans*, ***kwd*)

The number of histories that were deleted more than the specified number of days ago, but have not yet been purged. Also included is the number of datasets associated with the histories.

disk_usage(*trans*, ***kwd*)

get_disk_usage(*file_path*)

index(*trans*, ***kwd*)

userless_histories(*trans*, ***kwd*)

The number of userless histories and associated datasets that have not been updated for the specified number of days.

galaxy.webapps.reports.controllers.system.**nice_size**(*size*, *include_bytes=False*)

Returns a readably formatted string with the size

users Module

class galaxy.webapps.reports.controllers.users.**Users**(*app*)

Bases: *galaxy.web.base.controller.BaseUIController*, galaxy.webapps.reports.controllers.que

last_access_date(*trans*, ***kwd*)

registered_users(*trans*, ***kwd*)

registered_users_per_month(*trans*, ***kwd*)

specified_date(*trans*, ***kwd*)

specified_month(*trans*, ***kwd*)

user_disk_usage(*trans*, ***kwd*)

workflows Module

class galaxy.webapps.reports.controllers.workflows.**SpecifiedDateListGrid**

Bases: *galaxy.web.framework.helpers.grids.Grid*

class CreateTimeColumn(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: *galaxy.web.framework.helpers.grids.DateTimeColumn*

get_value(*trans*, *grid*, *stored_workflow*)

class SpecifiedDateListGrid.**EmailColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

filter(*trans*, *user*, *query*, *column_filter*)

class SpecifiedDateListGrid.**SpecifiedDateColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.GridColumn

filter(*trans*, *user*, *query*, *column_filter*)

class SpecifiedDateListGrid.**UserColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *stored_workflow*)

class SpecifiedDateListGrid.**WorkflowNameColumn**(*label*, *key=None*, *model_class=None*, *method=None*, *format=None*, *link=None*, *attach_popup=False*, *visible=True*, *nowrap=False*, *filterable=None*, *sortable=True*, *label_id_prefix=None*, *inbound=False*)

Bases: galaxy.web.framework.helpers.grids.TextColumn

get_value(*trans*, *grid*, *stored_workflow*)

SpecifiedDateListGrid.**build_initial_query**(*trans*, ***kwd*)

SpecifiedDateListGrid.**columns = [<galaxy.webapps.reports.controllers.workflows.WorkflowNameColumn objec**

SpecifiedDateListGrid.**default_filter = {'specified_date': 'All'}**

SpecifiedDateListGrid.**default_sort_key = 'name'**

SpecifiedDateListGrid.**model_class**

alias of StoredWorkflow

SpecifiedDateListGrid.**num_rows_per_page = 50**

SpecifiedDateListGrid.**preserve_state = False**

SpecifiedDateListGrid.**standard_filters = []**

SpecifiedDateListGrid.**template = '/webapps/reports/grid.mako'**

SpecifiedDateListGrid.**title = 'Workflows'**

SpecifiedDateListGrid.**use_async** = **False**

SpecifiedDateListGrid.**use_paging** = **True**

class galaxy.webapps.reports.controllers.workflows.**Workflows**(*app*)

Bases: *galaxy.web.base.controller.BaseUIController*, galaxy.webapps.reports.controllers.que

per_month_all(*trans*, ***kwd*)

per_user(*trans*, ***kwd*)

specified_date_handler(*trans*, ***kwd*)

specified_date_list_grid = **<galaxy.webapps.reports.controllers.workflows.SpecifiedDateListGrid object>**

user_per_month(*trans*, ***kwd*)

galaxy.webapps.reports.controllers.workflows.**get_workflow**(*trans*, *id*)

workflow Package

modules Module Modules used in building workflows

exception galaxy.workflow.modules.**CancelWorkflowEvaluation**

Bases: exceptions.Exception

exception galaxy.workflow.modules.**DelayedWorkflowEvaluation**

Bases: exceptions.Exception

class galaxy.workflow.modules.**InputDataCollectionModule**(*trans*)

Bases: *galaxy.workflow.modules.InputModule*

collection_type = **'list'**

default_collection_type = **'list'**

default_name = **'Input Dataset Collection'**

classmethod default_state(*Class*)

get_data_outputs()

get_runtime_inputs(*filter_set=['data']*)

name = **'Input dataset collection'**

state_fields = **['name', 'collection_type']**

type = **'data_collection_input'**

class galaxy.workflow.modules.**InputDataModule**(*trans*)

Bases: *galaxy.workflow.modules.InputModule*

default_name = **'Input Dataset'**

classmethod default_state(*Class*)

get_data_outputs()

get_runtime_inputs(*filter_set=['data']*)

name = **'Input dataset'**

state_fields = **['name']**

type = **'data_input'**

class galaxy.workflow.modules.**InputModule**(*trans*)
Bases: galaxy.workflow.modules.SimpleWorkflowModule

execute(*trans*, *progress*, *invocation*, *step*)

get_data_inputs()

get_runtime_input_dicts(*step_annotation*)

get_runtime_state()

recover_mapping(*step*, *step_invocations*, *progress*)

exception galaxy.workflow.modules.**MissingToolException**
Bases: exceptions.Exception

WorkflowModuleInjector will raise this if the tool corresponding to the module is missing.

class galaxy.workflow.modules.**PauseModule**(*trans*)
Bases: galaxy.workflow.modules.SimpleWorkflowModule

Initially this module will unconditionally pause a workflow - will aim to allow conditional pausing later on.

default_name = 'Pause for Dataset Review'

classmethod default_state(*Class*)

do_invocation_step_action(*step*, *action*)
Update or set the workflow invocation state action - generic extension point meant to allows users to interact with interactive workflow modules. The action object returned from this method will be attached to the WorkflowInvocationStep and be available the next time the workflow scheduler visits the workflow.

execute(*trans*, *progress*, *invocation*, *step*)

get_data_inputs()

get_data_outputs()

get_runtime_input_dicts(*step_annotation*)

get_runtime_inputs(***kwds*)

get_runtime_state()

name = 'Pause for dataset review'

recover_mapping(*step*, *step_invocations*, *progress*)

state_fields = ['name']

type = 'pause'

class galaxy.workflow.modules.**SimpleWorkflowModule**(*trans*)
Bases: galaxy.workflow.modules.WorkflowModule

compute_runtime_state(*trans*, *step_updates=None*, *source='html'*)

decode_runtime_state(*trans*, *string*)

classmethod default_state(*Class*)
This method should return a dictionary describing each configuration property and its default value.

encode_runtime_state(*trans*, *state*)

classmethod from_dict(*Class*, *trans*, *d*, *secure=True*)

classmethod from_workflow_step(*Class*, *trans*, *step*)

get_config_form()

get_state(*secure=True*)

classmethod new(*Class*, *trans*, *tool_id=None*)

normalize_runtime_state(*runtime_state*)

recover_runtime_state(*runtime_state*)
Take secure runtime state from persisted invocation and convert it into a DefaultToolState object for use during workflow invocation.

recover_state(*state*, ***kwds*)
Recover state *dict* from simple dictionary describing configuration state (potentially from persisted step state).

Sub-classes should supply *default_state* method and *state_fields* attribute which are used to build up the state *dict*.

save_to_step(*step*)

update_runtime_state(*trans*, *state*, *values*)

update_state(*incoming*)

class galaxy.workflow.modules.**ToolModule**(*trans*, *tool_id*, *tool_version=None*)
Bases: *galaxy.workflow.modules.WorkflowModule*

add_dummy_datasets(*connections=None*)

check_and_update_state()

compute_runtime_state(*trans*, *step_updates=None*, *source='html'*)

encode_runtime_state(*trans*, *state*)

execute(*trans*, *progress*, *invocation*, *step*)

classmethod from_dict(*Class*, *trans*, *d*, *secure=True*)

classmethod from_workflow_step(*Class*, *trans*, *step*)

get_config_form()

get_data_inputs()

get_data_outputs()

get_errors()

get_name()

get_post_job_actions(*incoming=None*)

get_runtime_input_dicts(*step_annotation*)

get_state(*secure=True*)

get_tool_id()

get_tool_version()

get_tooltip(*static_path=''*)

classmethod new(*Class*, *trans*, *tool_id=None*)

normalize_runtime_state(*runtime_state*)

recover_mapping(*step*, *step_invocations*, *progress*)

recover_runtime_state(*runtime_state*)
Take secure runtime state from persisted invocation and convert it into a DefaultToolState object for use during workflow invocation.

recover_state(*state*, ***kwds*)
Recover module configuration state property (a *DefaultToolState* object) using the tool's *params_from_strings* method.

save_to_step(*step*)

type = 'tool'

update_state(*incoming*)

class galaxy.workflow.modules.**WorkflowModule**(*trans*)
Bases: object

add_dummy_datasets(*connections=None*)

check_and_update_state()
If the state is not in sync with the current implementation of the module, try to update. Returns a list of messages to be displayed

compute_runtime_state(*trans*, *step_updates=None*, *source='html'*)
Determine the runtime state (potentially different from self.state which describes configuration state). This (again unlike self.state) is currently always a *DefaultToolState* object.

If *step_updates* is *None*, this is likely for rendering the run form for instance and no runtime properties are available and state must be solely determined by the default runtime state described by the step.

If *step_updates* are available they describe the runtime properties supplied by the workflow runner (potentially including a *tool_state* parameter which is the serialized default encoding state created with encode_runtime_state above).

do_invocation_step_action(*step*, *action*)
Update or set the workflow invocation state action - generic extension point meant to allows users to interact with interactive workflow modules. The action object returned from this method will be attached to the WorkflowInvocationStep and be available the next time the workflow scheduler visits the workflow.

encode_runtime_state(*trans*, *state*)
Encode the default runtime state at return as a simple *str* for use in a hidden parameter on the workflow run submission form.

This default runtime state will be combined with user supplied parameters in *compute_runtime_state* below at workflow invocation time to actually describe how each step will be executed.

execute(*trans*, *progress*, *invocation*, *step*)
Execute the given workflow step in the given workflow invocation. Use the supplied workflow progress object to track outputs, find inputs, etc...

classmethod from_dict(*Class*, *trans*, *d*)
Create a new instance of the module initialized from values in the dictionary *d*.

classmethod from_workflow_step(*Class*, *trans*, *step*)

get_config_form()
Render form that is embedded in workflow editor for modifying the step state of a node.

get_data_inputs()
Get configure time data input descriptions.

get_data_outputs()

get_errors()
It seems like this is effectively just used as boolean - some places in the tool shed self.errors is set to boolean, other places 'unavailable', likewise in Galaxy it stores a list containing a string with an unrecognized tool id error message.

get_name()

get_runtime_input_dicts(*step_annotation*)
Get runtime inputs (inputs and parameters) as simple dictionary.

get_runtime_inputs()
Used internally by modules and when displaying inputs in workflow editor and run workflow templates.

Note: The ToolModule doesn't implement this and these templates contain specialized logic for dealing with the tool and state directly in the case of ToolModules.

get_state()
Return a serializable representation of the persistable state of the step - for tools it DefaultToolState.encode returns a string and for simpler module types a json description is dumped out.

get_tool_id()

get_tooltip(*static_path=''*)

get_type()

classmethod new(*Class*, *trans*, *tool_id=None*)
Create a new instance of the module with default state

recover_mapping(*step*, *step_invocations*, *progress*)
Re-populate progress object with information about connections from previously executed steps recorded via step_invocations.

save_to_step(*step*)

update_state(*incoming*)
Update the current state of the module against the user supplied parameters in the dict-like object *incoming*.

class galaxy.workflow.modules.**WorkflowModuleFactory**(*module_types*)
Bases: object

from_dict(*trans*, *d*, ***kwargs*)
Return module initialized from the data in dictionary *d*.

from_workflow_step(*trans*, *step*)
Return module initializd from the WorkflowStep object *step*.

new(*trans*, *type*, *tool_id=None*)
Return module for type and (optional) tool_id intialized with new / default state.

class galaxy.workflow.modules.**WorkflowModuleInjector**(*trans*)
Bases: object

Injects workflow step objects from the ORM with appropriate module and module generated/influenced state.

inject(*step*, *step_args=None*, *source='html'*)
Pre-condition: *step* is an ORM object coming from the database, if supplied *step_args* is the representation of the inputs for that step supplied via web form.

Post-condition: The supplied *step* has new non-persistent attributes useful during workflow invocation. These include 'upgrade_messages', 'state', 'input_connections_by_name', and 'module'.

If step_args is provided from a web form this is applied to generate 'state' else it is just obtained from the database.

galaxy.workflow.modules.**is_tool_module_type**(*module_type*)

galaxy.workflow.modules.**load_module_sections**(*trans*)
Get abstract description of the workflow modules this Galaxy instance is configured with.

galaxy.workflow.modules.**populate_module_and_state**(*trans*, *workflow*, *param_map*)
Used by API but not web controller, walks through a workflow's steps and populates transient module and state attributes on each.

1.2.3 galaxy_utils Package

Subpackages

sequence Package

fasta Module

class galaxy_utils.sequence.fasta.**fastaNamedReader**(*fh*)
Bases: object

close()

get(*sequence_id*)

has_data()

class galaxy_utils.sequence.fasta.**fastaReader**(*fh*)
Bases: object

close()

next()

class galaxy_utils.sequence.fasta.**fastaSequence**
Bases: object

class galaxy_utils.sequence.fasta.**fastaWriter**(*fh*)
Bases: object

close()

write(*fastq_read*)

fastq Module

class galaxy_utils.sequence.fastq.**ReadlineCountFile**(*f*)
Bases: object

readline(**args*, ***kwds*)

class galaxy_utils.sequence.fastq.**fastqAggregator**
Bases: object

VALID_FORMATS = ['solexa', 'sanger', 'cssanger', 'illumina']

consume_read(*fastq_read*)

get_ascii_range()

get_base_counts_for_column(*column*)

get_decimal_range()

get_length_counts()

get_max_read_length()

get_read_count()

get_read_count_for_column(*column*)

get_score_at_position_for_column(*column*, *position*)

get_score_list_for_column(*column*)

get_score_max_for_column(*column*)

get_score_min_for_column(*column*)

get_score_sum_for_column(*column*)

get_summary_statistics_for_column(*i*)

get_valid_formats(*check_list=None*)

class galaxy_utils.sequence.fastq.**fastqCSSangerRead**

Bases: *galaxy_utils.sequence.fastq.fastqSequencingRead*

apply_galaxy_conventions()

ascii_max = 126

ascii_min = 33

assert_sequence_quality_lengths()

change_adapter(*new_adapter*, *clone=True*)

complement(*clone=True*)

format = 'cssanger'

get_sequence()

has_adapter_base()

insufficient_quality_length()

quality_max = 93

quality_min = 0

reverse(*clone=True*)

score_system = 'phred'

sequence_space = 'color'

valid_sequence_list = ['0', '1', '2', '3', '4', '5', '6', '.']

class galaxy_utils.sequence.fastq.**fastqCombiner**(*format*)

Bases: object

combine(*fasta_seq*, *quality_seq*)

class galaxy_utils.sequence.fastq.**fastqFakeFastaScoreReader**(*format='sanger'*, *quality_encoding=None*)

Bases: object

close()

get(*sequence*)

has_data()

class galaxy_utils.sequence.fastq.**fastqIlluminaRead**
Bases: *galaxy_utils.sequence.fastq.fastqSequencingRead*

ascii_max = 126

ascii_min = 64

format = 'illumina'

quality_max = 62

quality_min = 0

score_system = 'phred'

sequence_space = 'base'

class galaxy_utils.sequence.fastq.**fastqJoiner**(*format*, *force_quality_encoding=None*)
Bases: object

get_paired_identifier(*fastq_read*)

is_first_mate(*sequence_id*)

join(*read1*, *read2*)

class galaxy_utils.sequence.fastq.**fastqNamedReader**(*fh*, *format='sanger'*, *apply_galaxy_conventions=False*)
Bases: object

close()

get(*sequence_identifier*)

has_data()

class galaxy_utils.sequence.fastq.**fastqReader**(*fh*, *format='sanger'*, *apply_galaxy_conventions=False*)
Bases: object

close()

next()

class galaxy_utils.sequence.fastq.**fastqSangerRead**
Bases: *galaxy_utils.sequence.fastq.fastqSequencingRead*

ascii_max = 126

ascii_min = 33

format = 'sanger'

quality_max = 93

quality_min = 0

score_system = 'phred'

sequence_space = 'base'

class galaxy_utils.sequence.fastq.**fastqSequencingRead**
Bases: *galaxy_utils.sequence.sequence.SequencingRead*

apply_galaxy_conventions()

ascii_max = 126

ascii_min = 33

assert_sequence_quality_lengths()

classmethod convert_base_to_color_space(*sequence*)

classmethod convert_color_to_base_space(*sequence*)

convert_read_to_format(*format*, *force_quality_encoding=None*)

classmethod convert_score_phred_to_solexa(*decimal_score_list*)

classmethod convert_score_solexa_to_phred(*decimal_score_list*)

format = 'sanger'

get_ascii_quality_scores()

get_ascii_quality_scores_len()
Compute ascii quality score length, without generating relatively expensive qualty score array.

classmethod get_class_by_format(*format*)

get_decimal_quality_scores()

get_sequence()

insufficient_quality_length()

is_ascii_encoded()

is_valid_format()

is_valid_sequence()

quality_max = 93

quality_min = 0

classmethod restrict_scores_to_valid_range(*decimal_score_list*)

reverse(*clone=True*)

score_system = 'phred'

sequence_space = 'base'

slice(*left_column_offset*, *right_column_offset*)

classmethod transform_scores_to_valid_range(*decimal_score_list*)

classmethod transform_scores_to_valid_range_ascii(*decimal_score_list*)

class galaxy_utils.sequence.fastq.**fastqSolexaRead**
Bases: *galaxy_utils.sequence.fastq.fastqSequencingRead*

ascii_max = 126

ascii_min = 59

format = 'solexa'

quality_max = 62

quality_min = -5

score_system = 'solexa'

sequence_space = 'base'

class galaxy_utils.sequence.fastq.**fastqSplitter**
Bases: object

split (*fastq_read*)

class galaxy_utils.sequence.fastq.**fastqVerboseErrorReader** (*fh*, ***kwds*)

Bases: *galaxy_utils.sequence.fastq.fastqReader*

MAX_PRINT_ERROR_BYTES = 1024

next ()

class galaxy_utils.sequence.fastq.**fastqWriter** (*fh*, *format=None*, *force_quality_encoding=None*)

Bases: object

close ()

write (*fastq_read*)

galaxy_utils.sequence.fastq.**format**

alias of *fastqCSSangerRead*

sequence Module

class galaxy_utils.sequence.sequence.**SequencingRead**

Bases: object

append_quality (*quality*)

append_sequence (*sequence*)

clone ()

color_space_converter = <galaxy_utils.sequence.transform.ColorSpaceConverter object>

complement (*clone=True*)

is_DNA ()

reverse (*clone=True*)

reverse_complement (*clone=True*)

sequence_as_DNA (*clone=True*)

sequence_as_RNA (*clone=True*)

valid_sequence_list = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'

transform Module

class galaxy_utils.sequence.transform.**ColorSpaceConverter** (*fake_adapter_base='G'*)

Bases: object

base = 'N'

base_to_color_dict = {'A': {'A': '0', 'C': '1', 'T': '3', 'G': '2', 'N': '4'}, 'C': {'A': '1', 'C': '0', 'T': '2', 'G': '3', 'N':

color_dict = {'1': 'N', '0': 'N', '3': 'N', '2': 'N', '5': 'N', '4': 'N', '6': 'N', '.': 'N'}

color_to_base_dict = {'A': {'1': 'C', '0': 'A', '3': 'T', '2': 'G', '5': 'N', '4': 'N', '6': 'N', '.': 'N'}, 'C': {'1': 'A', '0':

key = '.'

to_base_space (*sequence*)

to_color_space (*sequence*, *adapter_base=None*)

unknown_base = 'N'

unknown_color = '.'

value = 'N'

galaxy_utils.sequence.transform.**DNA_complement**(*sequence*)

galaxy_utils.sequence.transform.**DNA_reverse_complement**(*sequence*)

galaxy_utils.sequence.transform.**RNA_complement**(*sequence*)

galaxy_utils.sequence.transform.**RNA_reverse_complement**(*sequence*)

galaxy_utils.sequence.transform.**reverse**(*sequence*)

galaxy_utils.sequence.transform.**to_DNA**(*sequence*)

galaxy_utils.sequence.transform.**to_RNA**(*sequence*)

vcf Module

class galaxy_utils.sequence.vcf.**Reader**(*fh*)

Bases: object

next()

class galaxy_utils.sequence.vcf.**VariantCall**(*vcf_line*, *metadata*, *sample_names*)

Bases: object

classmethod get_class_by_format(*format*)

header_startswith = None

required_header_fields = None

required_header_length = None

version = None

class galaxy_utils.sequence.vcf.**VariantCall33**(*vcf_line*, *metadata*, *sample_names*)

Bases: *galaxy_utils.sequence.vcf.VariantCall*

header_startswith = '#CHROM\tPOS\tID\tREF\tALT\tQUAL\tFILTER\tINFO'

required_header_fields = ['#CHROM', 'POS', 'ID', 'REF', 'ALT', 'QUAL', 'FILTER', 'INFO']

required_header_length = 8

version = 'VCFv3.3'

class galaxy_utils.sequence.vcf.**VariantCall40**(*vcf_line*, *metadata*, *sample_names*)

Bases: *galaxy_utils.sequence.vcf.VariantCall33*

version = 'VCFv4.0'

class galaxy_utils.sequence.vcf.**VariantCall41**(*vcf_line*, *metadata*, *sample_names*)

Bases: *galaxy_utils.sequence.vcf.VariantCall40*

version = 'VCFv4.1'

galaxy_utils.sequence.vcf.**format**

alias of *VariantCall41*

1.2.4 log_tempfile Module

class log_tempfile.**TempFile**

Bases: object

NamedTemporaryFile(**args*, ***kwargs*)

mkstemp(**args*, ***kwargs*)

1.2.5 mimeparse Module

MIME-Type Parser

This module provides basic functions for handling mime-types. It can handle matching mime-types against a list of media-ranges. See section 14.1 of the HTTP specification [RFC 2616] for a complete explanation.

http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.1

Contents:

- parse_mime_type(): Parses a mime-type into its component parts.
- parse_media_range(): Media-ranges are mime-types with wild-cards and a 'q' quality parameter.
- quality(): Determines the quality ('q') of a mime-type when compared against a list of media-ranges.
- quality_parsed(): Just like quality() except the second parameter must be pre-parsed.
- best_match(): Choose the mime-type with the highest quality ('q') from a list of candidates.

mimeparse.**best_match**(*supported*, *header*)

Takes a list of supported mime-types and finds the best match for all the media-ranges listed in header. The value of header must be a string that conforms to the format of the HTTP Accept: header. The value of 'supported' is a list of mime-types.

```
>>> best_match(['application/xbel+xml', 'text/xml'], 'text/*;q=0.5,*/*; q=0.1')
'text/xml'
```

mimeparse.**fitness_and_quality_parsed**(*mime_type*, *parsed_ranges*)

Find the best match for a given mime-type against a list of media_ranges that have already been parsed by parse_media_range(). Returns a tuple of the fitness value and the value of the 'q' quality parameter of the best match, or (-1, 0) if no match was found. Just as for quality_parsed(), 'parsed_ranges' must be a list of parsed media ranges.

mimeparse.**parse_media_range**(*range*)

Carves up a media range and returns a tuple of the (type, subtype, params) where 'params' is a dictionary of all the parameters for the media range. For example, the media range 'application/*;q=0.5' would get parsed into:

In addition this function also guarantees that there is a value for 'q' in the params dictionary, filling it in with a proper default if necessary.

mimeparse.**parse_mime_type**(*mime_type*)

Carves up a mime-type and returns a tuple of the (type, subtype, params) where 'params' is a dictionary of all the parameters for the media range. For example, the media range 'application/xhtml;q=0.5' would get parsed into:

('application', 'xhtml', {'q', '0.5'})

mimeparse.**quality**(*mime_type*, *ranges*)

Returns the quality 'q' of a mime-type when compared against the media-ranges in ranges. For example:

```
>>> quality('text/html','text/*;q=0.3, text/html;q=0.7, text/html;level=1, text/html;level=2;q=0
0.7
```

mimeparse.**quality_parsed**(*mime_type*, *parsed_ranges*)
Find the best match for a given mime-type against a list of media_ranges that have already been parsed by parse_media_range(). Returns the 'q' quality parameter of the best match, 0 if no match was found. This function bahaves the same as quality() except that 'parsed_ranges' must be a list of parsed media ranges.

1.2.6 pkg_resources Module

Package resource API

A resource is a logical file contained within a package, or a logical subdirectory thereof. The package resource API expects resource names to have their path parts separated with `/`, *not* whatever the local path separator is. Do not use os.path operations to manipulate resource names being passed into the API.

The package resource API is designed to work with normal filesystem packages, .egg files, and unpacked .egg files. It can also work in a limited way with .zip files and with custom PEP 302 loaders that support the `get_data()` method.

pkg_resources.**require**(*req_str*)

pkg_resources.**get_provider**(*moduleOrReq*)
Return an IResourceProvider for the named module or requirement

pkg_resources.**get_distribution**(*dist*)
Return a current distribution object for a Requirement or string

pkg_resources.**load_entry_point**(*dist*, *group*, *name*)
Return *name* entry point of *group* for *dist* or raise ImportError

pkg_resources.**get_entry_map**(*dist*, *group=None*)
Return the entry point map for *group*, or the full entry map

pkg_resources.**get_entry_info**(*dist*, *group*, *name*)
Return the EntryPoint object for *group*'+'*name*, or `None`

pkg_resources.**declare_namespace**(*packageName*)
Declare that package 'packageName' is a namespace package

pkg_resources.**find_distributions**(*path_item*, *only=False*)
Yield distributions accessible via *path_item*

pkg_resources.**get_default_cache**()
Determine the default cache location

This returns the `PYTHON_EGG_CACHE` environment variable, if set. Otherwise, on Windows, it returns a "Python-Eggs" subdirectory of the "Application Data" directory. On all other systems, it's "~/.python-eggs".

class pkg_resources.**Environment**(*search_path=None*, *platform='linux-x86_64'*, *python='2.7'*)
Bases: `object`

Searchable snapshot of distributions on a search path

add(*dist*)
Add *dist* if we `can_add()` it and it isn't already added

best_match(*req*, *working_set*, *installer=None*)
Find distribution best matching *req* and usable on *working_set*

This calls the `find(req)` method of the *working_set* to see if a suitable distribution is already active. (This may raise `VersionConflict` if an unsuitable version of the project is already active in the specified *working_set*.) If a suitable distribution isn't active, this method returns the newest distribution in the environment that meets the `Requirement` in *req*. If no suitable distribution is found, and *installer*

is supplied, then the result of calling the environment's `obtain(req, installer)` method will be returned.

can_add(*dist*)
Is distribution *dist* acceptable for this environment?

The distribution must match the platform and python version requirements specified when this environment was created, or False is returned.

obtain(*requirement*, *installer=None*)
Obtain a distribution matching *requirement* (e.g. via download)

Obtain a distro that matches requirement (e.g. via download). In the base `Environment` class, this routine just returns `installer(requirement)`, unless *installer* is None, in which case None is returned instead. This method is a hook that allows subclasses to attempt other ways of obtaining a distribution before falling back to the *installer* argument.

remove(*dist*)
Remove *dist* from the environment

scan(*search_path=None*)
Scan *search_path* for distributions usable in this environment

Any distributions found are added to the environment. *search_path* should be a sequence of `sys.path` items. If not supplied, `sys.path` is used. Only distributions conforming to the platform/python version defined at initialization are added.

class `pkg_resources.`**WorkingSet**(*entries=None*)
Bases: `object`

A collection of active distributions on sys.path (or a similar list)

add(*dist*, *entry=None*, *insert=True*)
Add *dist* to working set, associated with *entry*

If *entry* is unspecified, it defaults to the `.location` of *dist*. On exit from this routine, *entry* is added to the end of the working set's `.entries` (if it wasn't already present).

dist is only added to the working set if it's for a project that doesn't already have a distribution in the set. If it's added, any callbacks registered with the `subscribe()` method will be called.

add_entry(*entry*)
Add a path item to `.entries`, finding any distributions on it

`find_distributions(entry, True)` is used to find distributions corresponding to the path entry, and they are added. *entry* is always appended to `.entries`, even if it is already present. (This is because `sys.path` can contain the same value more than once, and the `.entries` of the `sys.path` WorkingSet should always equal `sys.path`.)

find(*req*)
Find a distribution matching requirement *req*

If there is an active distribution for the requested project, this returns it as long as it meets the version requirement specified by *req*. But, if there is an active distribution for the project and it does *not* meet the *req* requirement, `VersionConflict` is raised. If there is no active distribution for the requested project, `None` is returned.

find_plugins(*plugin_env*, *full_env=None*, *installer=None*, *fallback=True*)
Find all activatable distributions in *plugin_env*

Example usage:

```
        distributions, errors = working_set.find_plugins(
            Environment(plugin_dirlist)
        )
        map(working_set.add, distributions)  # add plugins+libs to sys.path
        print 'Could not load', errors        # display errors
```

The *plugin_env* should be an `Environment` instance that contains only distributions that are in the project's "plugin directory" or directories. The *full_env*, if supplied, should be an `Environment` contains all currently-available distributions. If *full_env* is not supplied, one is created automatically from the `WorkingSet` this method is called on, which will typically mean that every directory on `sys.path` will be scanned for distributions.

installer is a standard installer callback as used by the `resolve()` method. The *fallback* flag indicates whether we should attempt to resolve older versions of a plugin if the newest version cannot be resolved.

This method returns a 2-tuple: (*distributions*, *error_info*), where *distributions* is a list of the distributions found in *plugin_env* that were loadable, along with any other distributions that are needed to resolve their dependencies. *error_info* is a dictionary mapping unloadable plugin distributions to an exception instance describing the error that occurred. Usually this will be a `DistributionNotFound` or `VersionConflict` instance.

iter_entry_points(*group*, *name=None*)

Yield entry point objects from *group* matching *name*

If *name* is None, yields all entry points in *group* from all distributions in the working set, otherwise only ones matching both *group* and *name* are yielded (in distribution order).

require(**requirements*)

Ensure that distributions matching *requirements* are activated

requirements must be a string or a (possibly-nested) sequence thereof, specifying the distributions and versions required. The return value is a sequence of the distributions that needed to be activated to fulfill the requirements; all relevant distributions are included, even if they were already activated in this working set.

resolve(*requirements*, *env=None*, *installer=None*)

List all distributions needed to (recursively) meet *requirements*

requirements must be a sequence of `Requirement` objects. *env*, if supplied, should be an `Environment` instance. If not supplied, it defaults to all distributions available within any entry or distribution in the working set. *installer*, if supplied, will be invoked with each requirement that cannot be met by an already-installed distribution; it should return a `Distribution` or `None`.

run_script(*requires*, *script_name*)

Locate distribution for *requires* and run *script_name* script

subscribe(*callback*)

Invoke *callback* for all distributions (including existing ones)

class pkg_resources.**ResourceManager**

Manage resource extraction and packages

cleanup_resources(*force=False*)

Delete all extracted resource files and directories, returning a list of the file and directory names that could not be successfully removed. This function does not have any concurrency protection, so it should generally only be called when the extraction path is a temporary directory exclusive to a single process. This method is not automatically called; you must call it explicitly or register it as an `atexit` function if you wish to ensure cleanup of a temporary directory used for extractions.

extraction_error()

Give an error message for problems extracting file(s)

extraction_path = None

get_cache_path (*archive_name*, *names=()*)

Return absolute location in cache for *archive_name* and *names*

The parent directory of the resulting path will be created if it does not already exist. *archive_name* should be the base filename of the enclosing egg (which may not be the name of the enclosing zipfile!), including its ".egg" extension. *names*, if provided, should be a sequence of path name parts "under" the egg's extraction location.

This method should only be called by resource providers that need to obtain an extraction location, and only for names they intend to extract, as it tracks the generated names for possible cleanup later.

postprocess (*tempname*, *filename*)

Perform any platform-specific postprocessing of *tempname*

This is where Mac header rewrites should be done; other platforms don't have anything special they should do.

Resource providers should call this method ONLY after successfully extracting a compressed resource. They must NOT call it on resources that are already in the filesystem.

tempname is the current (temporary) name of the file, and *filename* is the name it will be renamed to by the caller after this routine returns.

resource_exists (*package_or_requirement*, *resource_name*)

Does the named resource exist?

resource_filename (*package_or_requirement*, *resource_name*)

Return a true filesystem path for specified resource

resource_isdir (*package_or_requirement*, *resource_name*)

Is the named resource an existing directory?

resource_listdir (*package_or_requirement*, *resource_name*)

List the contents of the named resource directory

resource_stream (*package_or_requirement*, *resource_name*)

Return a readable file-like object for specified resource

resource_string (*package_or_requirement*, *resource_name*)

Return specified resource as a string

set_extraction_path (*path*)

Set the base path where resources will be extracted to, if needed.

If you do not call this routine before any extractions take place, the path defaults to the return value of `get_default_cache()`. (Which is based on the `PYTHON_EGG_CACHE` environment variable, with various platform-specific fallbacks. See that routine's documentation for more details.)

Resources are extracted to subdirectories of this path based upon information given by the `IResourceProvider`. You may set this to a temporary directory, but then you must call `cleanup_resources()` to delete the extracted files when done. There is no guarantee that `cleanup_resources()` will be able to remove all extracted files.

(Note: you may not change the extraction path for a given resource manager once resources have been extracted, unless you first call `cleanup_resources()`.)

class `pkg_resources.`**Distribution** (*location=None*, *metadata=None*, *project_name=None*, *version=None*, *py_version='2.7'*, *platform=None*, *precedence=3*)

Bases: `object`

Wrap an actual or potential sys.path entry w/metadata

PKG_INFO = **'PKG-INFO'**

activate(*path=None*)
Ensure distribution is importable on *path* (default=sys.path)

as_requirement()
Return a `Requirement` that matches this distribution exactly

check_version_conflict()

clone(***kw*)
Copy this distribution, substituting in any changed keyword args

egg_name()
Return what this distribution's standard .egg filename should be

extras

classmethod from_filename(*filename*, *metadata=None*, ***kw*)

classmethod from_location(*location*, *basename*, *metadata=None*, ***kw*)

get_entry_info(*group*, *name*)
Return the EntryPoint object for *group*'+'*name*, or `None`

get_entry_map(*group=None*)
Return the entry point map for *group*, or the full entry map

has_version()

hashcmp

insert_on(*path*, *loc=None*)
Insert self.location in path before its nearest parent directory

key

load_entry_point(*group*, *name*)
Return the *name* entry point of *group* or raise ImportError

parsed_version

requires(*extras=()*)
List of Requirements needed for this distro if *extras* are used

version

class pkg_resources.**Requirement**(*project_name*, *specs*, *extras*)

static parse(*s*)

class pkg_resources.**EntryPoint**(*name*, *module_name*, *attrs=()*, *extras=()*, *dist=None*)
Bases: `object`

Object representing an advertised importable object

load(*require=True*, *env=None*, *installer=None*)

classmethod parse(*src*, *dist=None*)
Parse a single entry point from string *src*

Entry point syntax follows the form:

```
name = some.module:some.attr [extra1,extra2]
```

The entry name and module name are required, but the `:attrs` and `[extras]` parts are optional

classmethod parse_group (*group*, *lines*, *dist=None*)
Parse an entry point group

classmethod parse_map (*data*, *dist=None*)
Parse a map of entry point groups

require (*env=None*, *installer=None*)

exception pkg_resources.**ResolutionError**
Bases: exceptions.Exception

Abstract base for dependency resolution errors

exception pkg_resources.**VersionConflict**
Bases: *pkg_resources.ResolutionError*

An already-installed version conflicts with the requested version

exception pkg_resources.**DistributionNotFound**
Bases: *pkg_resources.ResolutionError*

A requested distribution was not found

exception pkg_resources.**UnknownExtra**
Bases: *pkg_resources.ResolutionError*

Distribution doesn't have an "extra feature" of the given name

exception pkg_resources.**ExtractionError**
Bases: exceptions.RuntimeError

An error occurred extracting a resource

The following attributes are available from instances of this exception:
manager The resource manager that raised this exception
cache_path The base directory for resource extraction
original_error The exception instance that caused extraction to fail

pkg_resources.**parse_requirements** (*strs*)
Yield Requirement objects for each specification in *strs*

strs must be an instance of basestring, or a (possibly-nested) iterable thereof.

pkg_resources.**parse_version** (*s*)
Convert a version string to a chronologically-sortable key

This is a rough cross between distutils' StrictVersion and LooseVersion; if you give it versions that would work with StrictVersion, then it behaves the same; otherwise it acts like a slightly-smarter LooseVersion. It is *possible* to create pathological version coding schemes that will fool this parser, but they should be very rare in practice.

The returned value will be a tuple of strings. Numeric portions of the version are padded to 8 digits so they will compare numerically, but without relying on how numbers compare relative to strings. Dots are dropped, but dashes are retained. Trailing zeros between alpha segments or dashes are suppressed, so that e.g. "2.4.0" is considered the same as "2.4". Alphanumeric parts are lower-cased.

The algorithm assumes that strings like "-" and any alpha string that alphabetically follows "final" represents a "patch level". So, "2.4-1" is assumed to be a branch or patch of "2.4", and therefore "2.4.1" is considered newer than "2.4-1", which in turn is newer than "2.4".

Strings like "a", "b", "c", "alpha", "beta", "candidate" and so on (that come before "final" alphabetically) are assumed to be pre-release versions, so that the version "2.4" is considered newer than "2.4a1".

Finally, to handle miscellaneous cases, the strings "pre", "preview", and "rc" are treated as if they were "c", i.e. as though they were release candidates, and therefore are not as new as a version string that does not contain them, and "dev" is replaced with an '@' so that it sorts lower than than any other pre-release tag.

`pkg_resources.`**`safe_name`**(*name*)
Convert an arbitrary string to a standard distribution name

Any runs of non-alphanumeric/. characters are replaced with a single '-'.

`pkg_resources.`**`safe_version`**(*version*)
Convert an arbitrary string to a standard version string

Spaces become dots, and all other non-alphanumeric characters become dashes, with runs of multiple dashes condensed to a single dash.

`pkg_resources.`**`get_platform`**()

`pkg_resources.`**`compatible_platforms`**(*provided*, *required*)
Can code for the *provided* platform run on the *required* platform?

Returns true if either platform is `None`, or the platforms are equal.

XXX Needs compatibility checks for Linux and other unixy OSes.

`pkg_resources.`**`yield_lines`**(*strs*)
Yield non-empty/non-comment lines of a `basestring` or sequence

`pkg_resources.`**`split_sections`**(*s*)
Split a string or iterable thereof into (section,content) pairs

Each `section` is a stripped version of the section header ("[section]") and each `content` is a list of stripped lines excluding blank lines and comment-only lines. If there are any such lines before the first section header, they're returned in a first `section` of `None`.

`pkg_resources.`**`safe_extra`**(*extra*)
Convert an arbitrary string to a standard 'extra' name

Any runs of non-alphanumeric characters are replaced with a single '_', and the result is always lowercased.

`pkg_resources.`**`to_filename`**(*name*)
Convert a project or version name to its filename-escaped form

Any '-' characters are currently replaced with '_'.

`pkg_resources.`**`invalid_marker`**(*text*)
Validate text as a PEP 426 environment marker; return exception or False

`pkg_resources.`**`evaluate_marker`**(*text*, *extra=None*, *_ops={'not in': <function <lambda> at 0x7f4978c077d0>, '==': <built-in function eq>, 304: <function test at 0x7f4978c07668>, 305: <function test at 0x7f4978c07668>, 306: <function and_test at 0x7f4978c02ed8>, 308: <function comparison at 0x7f4978c07758>, 'in': <function <lambda> at 0x7f4978c07848>, '!=': <built-in function ne>, 318: <function atom at 0x7f4978c076e0>}*)
Evaluate a PEP 426 environment marker on CPython 2.4+. Return a boolean indicating the marker result in this environment. Raise SyntaxError if marker is invalid.

This implementation uses the 'parser' module, which is not implemented on Jython and has been superseded by the 'ast' module in Python 2.6 and later.

`pkg_resources.`**`ensure_directory`**(*path*)
Ensure that the parent directory of *path* exists

pkg_resources.**normalize_path**(*filename*)
Normalize a file/dir name for comparison purposes

class pkg_resources.**IMetadataProvider**

get_metadata(*name*)
The named metadata resource as a string

get_metadata_lines(*name*)
Yield named metadata resource as list of non-blank non-comment lines

Leading and trailing whitespace is stripped from each line, and lines with # as the first non-blank character are omitted.

has_metadata(*name*)
Does the package's distribution contain the named metadata?

metadata_isdir(*name*)
Is the named metadata a directory? (like `os.path.isdir()`)

metadata_listdir(*name*)
List of metadata names in the directory (like `os.listdir()`)

run_script(*script_name*, *namespace*)
Execute the named script in the supplied namespace dictionary

class pkg_resources.**IResourceProvider**
Bases: *pkg_resources.IMetadataProvider*

An object that provides access to package resources

get_resource_filename(*manager*, *resource_name*)
Return a true filesystem path for *resource_name*

manager must be an `IResourceManager`

get_resource_stream(*manager*, *resource_name*)
Return a readable file-like object for *resource_name*

manager must be an `IResourceManager`

get_resource_string(*manager*, *resource_name*)
Return a string containing the contents of *resource_name*

manager must be an `IResourceManager`

has_resource(*resource_name*)
Does the package contain the named resource?

resource_isdir(*resource_name*)
Is the named resource a directory? (like `os.path.isdir()`)

resource_listdir(*resource_name*)
List of resource names in the directory (like `os.listdir()`)

class pkg_resources.**FileMetadata**(*path*)
Bases: *pkg_resources.EmptyProvider*

Metadata handler for standalone PKG-INFO files

Usage:

```
metadata = FileMetadata("/path/to/PKG-INFO")
```

This provider rejects all data and metadata requests except for PKG-INFO, which is treated as existing, and will be the contents of the file at the provided location.

get_metadata(*name*)

get_metadata_lines(*name*)

has_metadata(*name*)

class pkg_resources.**PathMetadata**(*path*, *egg_info*)
Bases: *pkg_resources.DefaultProvider*

Metadata provider for egg directories

Usage:

```
# Development eggs:

egg_info = "/path/to/PackageName.egg-info"
base_dir = os.path.dirname(egg_info)
metadata = PathMetadata(base_dir, egg_info)
dist_name = os.path.splitext(os.path.basename(egg_info))[0]
dist = Distribution(basedir,project_name=dist_name,metadata=metadata)

# Unpacked egg directories:

egg_path = "/path/to/PackageName-ver-pyver-etc.egg"
metadata = PathMetadata(egg_path, os.path.join(egg_path,'EGG-INFO'))
dist = Distribution.from_filename(egg_path, metadata=metadata)
```

class pkg_resources.**EggMetadata**(*importer*)
Bases: *pkg_resources.ZipProvider*

Metadata provider for .egg files

class pkg_resources.**EmptyProvider**
Bases: *pkg_resources.NullProvider*

Provider that returns nothing for all requests

module_path = None

class pkg_resources.**NullProvider**(*module*)
Try to implement resources and metadata for arbitrary PEP 302 loaders

egg_info = None

egg_name = None

get_metadata(*name*)

get_metadata_lines(*name*)

get_resource_filename(*manager*, *resource_name*)

get_resource_stream(*manager*, *resource_name*)

get_resource_string(*manager*, *resource_name*)

has_metadata(*name*)

has_resource(*resource_name*)

loader = None

metadata_isdir(*name*)

metadata_listdir(*name*)

resource_isdir(*resource_name*)

resource_listdir(*resource_name*)

run_script(*script_name*, *namespace*)

class pkg_resources.**EggProvider**(*module*)
Bases: *pkg_resources.NullProvider*

Provider based on a virtual filesystem

class pkg_resources.**DefaultProvider**(*module*)
Bases: *pkg_resources.EggProvider*

Provides access to package resources in the filesystem

get_resource_stream(*manager*, *resource_name*)

class pkg_resources.**ZipProvider**(*module*)
Bases: *pkg_resources.EggProvider*

Resource support for zips and eggs

eagers = None

get_resource_filename(*manager*, *resource_name*)

pkg_resources.**register_finder**(*importer_type*, *distribution_finder*)
Register *distribution_finder* to find distributions in sys.path items

importer_type is the type or class of a PEP 302 "Importer" (sys.path item handler), and *distribution_finder* is a callable that, passed a path item and the importer instance, yields `Distribution` instances found on that path item. See `pkg_resources.find_on_path` for an example.

pkg_resources.**register_namespace_handler**(*importer_type*, *namespace_handler*)
Register *namespace_handler* to declare namespace packages

importer_type is the type or class of a PEP 302 "Importer" (sys.path item handler), and *namespace_handler* is a callable like this:

```
def namespace_handler(importer,path_entry,moduleName,module):
    # return a path_entry to use for child packages
```

Namespace handlers are only called if the importer object has already agreed that it can handle the relevant path item, and they should only return a subpath if the module __path__ does not already contain an equivalent subpath. For an example namespace handler, see `pkg_resources.file_ns_handler`.

pkg_resources.**register_loader_type**(*loader_type*, *provider_factory*)
Register *provider_factory* to make providers for *loader_type*

loader_type is the type or class of a PEP 302 `module.__loader__`, and *provider_factory* is a function that, passed a *module* object, returns an `IResourceProvider` for that module.

pkg_resources.**fixup_namespace_packages**(*path_item*, *parent=None*)
Ensure that previously-declared namespace packages include path_item

pkg_resources.**get_importer**(*path_item*)
Retrieve a PEP 302 importer for the given path item

The returned importer is cached in sys.path_importer_cache if it was newly created by a path hook.

If there is no importer, a wrapper around the basic import machinery is returned. This wrapper is never inserted into the importer cache (None is inserted instead).

The cache (or part of it) can be cleared manually if a rescan of sys.path_hooks is necessary.

`pkg_resources.AvailableDistributions`
alias of *Environment*

1.2.7 psyco_full Module

Attempt to call psyco.full, but ignore any errors.

1.3 Releases

1.3.1 May 2015 Galaxy Release (v 15.05)

Highlights

Authentication Plugins Galaxy now has native support for LDAP and Active Directory via a new community developed authentication plugin system.

Tool Sections Tool parameters may now be groupped into collapsable sections.

Collection Creators New widgets have been added that allow much more flexibility when creating simple dataset pair and list collections.

Github

New

```
% git clone -b master https://github.com/galaxyproject/galaxy.git
```

Update to latest stable release

```
% git checkout master && pull --ff-only origin master
```

Update to exact version

```
% git checkout v15.05
```

BitBucket

Upgrade

```
% hg pull
% hg update latest_15.05
```

See our wiki for additional details regarding the source code locations.

Release Notes

Enhancements

- Pluggable framework to custom authentication (including new LDAP/Active Directory integration). Thanks to many including Andrew Robinson, Nicola Soranzo, and David Trudgian. Pull Request 1, Pull Request 33, Pull Request 51, Pull Request 75, Pull Request 98, Pull Request 216
- Implement a new `section` tag for tool parameters. Pull Request 35, Trello
- New UI widgets allowing much more flexibility when creating simple dataset pair and list collections. Pull Request 134, Trello
- Improved JavaScript build system for client code and libraries (now using uglify and featuring Source Maps). 72c876c, 9a7f5fc, 648a623, 22f280f, Trello
- Add an External Display Application for viewing GFF/GTF files with IGV. Pull Request 70, Trello
- Use TravisCI and Tox for continuous integration testing. Pull Request 40, Pull Request 62, Pull Request 97, Pull Request 99, Pull Request 123, Pull Request 222, Pull Request 235,
- Infrastructure for improved toolbox and Tool Shed searching. Pull Request 9, Pull Request 116, Pull Request 142, Pull Request 226, c2eb74c, 2bf52fe, ec549db, Trello, Trello
- Enhance UI to allow renaming dataset collections. 21d1d6b
- Improve highlighting of current/active content history panel. Pull Request 126
- Improvements to UI and API for histories and collections. e36e51e, 1e55206, 0c79680
- Update history dataset API to account for job re-submission. b4cf49a
- Allow recalculating user disk usage from the admin interface. 964e081
- Collect significantly more metadata for BAM files. Pull Request 107, Pull Request 108
- Implement `detect_errors` attribute on command of tool XML. Pull Request 117
- Allow setting `auto_format="True"` on tool `output` tags. Pull Request 130
- Allow testing tool outputs based on MD5 hashes. Pull Request 125
- Improved Cheetah type casting for int/float values. Pull Request 121
- Add option to pass arbitrary parameters to gem install as part of the tool shed `setup_ruby_environment` Tool Shed install action - thanks to Björn Grüning. Pull Request 118
- Add `argument` attribute to tool parameters. Pull Request 8
- Improve link and message that appears after workflows are run. Pull Request 143
- Add NCBI SRA datatype - thanks to Matt Shirley. Pull Request 87
- Stronger toolbox filtering. Pull Request 119
- Allow updating Tool Shed repositories via the API - thanks to Eric Rasche. Pull Request 30
- Expose category list in show call for Tool Shed repositories - thanks to Eric Rasche. Pull Request 29
- Add API endpoint to create Tool Shed repositories. Pull Request 2
- Do not configure Galaxy to use the test Tool Shed by default. Pull Request 38
- Add fields and improve display of Tool Shed repositories. a24e206, d6d61bc, Trello
- Enhance multi-selection widgets to allow key combinations `Ctrl-A` and `Ctrl-X`. e8564d7, Trello

- New, consistent button for displaying citation BibTeX. Pull Request 19
- Improved `README` reflecting move to Github - thanks in part to Eric Rasche. PR #2 (old repo), 226e826, 2650d09, 7d5dde8
- Update application to use new logo. 2748f9d, Pull Request 187, Pull Request 206
- Update many documentation links to use https sites - thanks to Nicola Soranzo. 8254cab
- Sync report options config with `galaxy.ini` - thanks to Björn Grüning. Pull Request 12
- Eliminate need to use API key to list tools via API. cd7abe8
- Restore function necessary for splitting sequence datatypes - thanks to Roberto Alonso. Pull Request 5
- Suppress filenames in SAM merge using `egrep` - thanks to Peter Cock and Roberto Alonso. Pull Request 4
- Option to sort counts in `Count1` tool (`tools/filters/uniq.xml`) - thanks to Peter Cock. Pull Request 16
- Preserve spaces in `Count1` tool (`tools/filters/uniq.xml`) - thanks to Peter Cock. Pull Request 13
- Interactive Environments improvements and fixes from multiple developers including Eric Rasche and Björn Grüning. Pull Request 69, Pull Request 73, Pull Request 131, Pull Request 135, Pull Request 152, Pull Request 197
- Enable multi-part upload for exporting files with the GenomeSpace export tool. Pull Request 74, Trello
- Large refactoring, expansion, and increase in test coverage for "managers". Pull Request 76
- Improved display of headers in tool help. 157eba6, Biostar
- Uniform configuration of "From" field for sent emails - thanks to Nicola Soranzo. Pull Request 23
- Allow setting `job_conf.xml` params via environment variables & `galaxy.ini`. dde2fc9
- Allow a tool data table to declare that duplicate entries are not allowed. Pull Request 245
- Add verbose test error flag option in run_tests.sh. 62f0495
- Update `.gitignore` to include `run_api_tests.html`. b52cc98
- Add experimental options to run tests in Docker. e99adb5
- Improve `run_test.sh --help` documentation to detail running specific tests. Pull Request 86
- Remove older, redundant history tests. Pull Request 120, Trello
- Add test tool demonstrating citing a Github repository. 65def71
- Add option to track all automated changes to the integrated tool panel. 10bb492
- Make tool version explicit in all distribution tool - thanks to Peter Cock. Pull Request 14.
- Relocate the external metadata setting script. Pull Request 7
- Parameterize script used to pull new builds from the UCSC Browser. e4e5df0
- Enhance jobs and workflow logging to report timings. 06346a4
- Add debug message for dynamic options exceptions. Pull Request 91
- Remove demo sequencer app. 3af3bf5
- Tweaks to the Pulsar's handling of async messages. Pull Request 109
- Return more specific API authentication errors. 71a64ca
- Upgrade Python dependency sqlalchemy to 1.0.0. d725aab, Pull Request 129

- Upgrade Python dependency amqp to 1.4.6. Pull Request 128
- Upgrade Python dependency kombu to 3.0.24. Pull Request 128
- Upgrade JavaScript dependency raven.js to 1.1.17. bcd1701

Fixes

- During the 15.05 development cycle dozens of fixes were pushed to the `release_15.03` branch of Galaxy. These are all included in 15.05 and summarized here (with special thanks to Björn Grüning and Marius van den Beek).
- Fix race condition that would occasionally prevent Galaxy from starting properly. Pull Request 198, Trello
- Fix scatter plot API communications for certain proxied Galaxy instances - thanks to @yhoogstrate. Pull Request 89
- Fix bug in collectl job metrics plugin - thanks to Carrie Ganote. Pull Request 231
- Fix late validation of tool parameters. Pull Request 115
- Fix `fasta_to_tabular_converter.py` (for implicit conversion) - thanks to Peter Cock. Pull Request 11
- Fix to eliminate race condition by collecting extra files before declaring dataset's OK. Pull Request 48
- Fix setting current history for certain proxied Galaxy instances - thanks to @wezen. 6946e46.
- Fix typo in tool failure testing example - thanks to Peter Cock. Pull Request 18.
- Fix Galaxy to default to using SSL for communicating with Tool Sheds. 0b037a2
- Fix data source tools to open in `_top` window. Pull Request 17
- Fix to fallback to name for tool parameters without labels. Pull Request 189, Trello
- Fix to remove redundant version ids in tool version selector. Pull Request 244
- Fix for downloading metadata files. Pull Request 234
- Fix for history failing to render if it contains more exotic dataset collection types. Pull Request 196
- Fixes for BaseURLToolParameter. Pull Request 247
- Fix to suppress pysam binary incompatibility warning when using datatypes in `binary.py`. Pull Request 252
- Fix for library UI duplication bug. Pull Request 179
- Fix for Backbone.js loading as AMD. 4e5218f
- Other small Tool Shed fixes. 815f86f, 76e0915
- Fix file closing in `lped_to_pbed_converter`. 182b67f
- Fix undefined variables in Tool Shed `add_repository_entry` API script. 47e6f08
- Fix user registration to respect use_panels when in the Galaxy app. 7ac8631, Trello
- Fix bug in scramble exception, incorrect reference to source_path 79d50d8
- Fix error handling in `pbed_to_lped`. 7aecd7a
- Fix error handling in Tool Shed step handler for `chmod` action. 1454396
- Fix `__safe_string_wrapper` in tool evaluation object_wrapper. ab6f13e
- Fixes for data types and data providers. c1d2d1f, 8da70bb, 0b83b1e

- Fixes for Tool Shed commit and mercurial handling modules. 6102edf, b639bc0, debea9d
- Fix to clean working directory during job re-submission. Pull Request 236
- Fix bug when task splitting jobs fail. Pull Request 214
- Fix some minor typos in comment docs in `config/galaxy.ini.sample`. Pull Request 210
- Fix admin disk usage message. Pull Request 205, Trello
- Fix to sessionStorage Model to suppress QUOTA DOMExceptions when Safari users are in private browsing mode. 0c94f04

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.2 March 2015 Galaxy Release (v 15.03)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.3 January 2015 Galaxy Release (v 15.01)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.4 October 2014 Galaxy Release (v 14.10)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.5 August 2014 Galaxy Release (v 14.08)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.6 June 2014 Galaxy Release (v 14.06)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.7 April 2014 Galaxy Release (v 14.04)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.8 February 2014 Galaxy Release (v 14.02)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.9 November 2013 Galaxy Release (v 13.11)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.10 August 2013 Galaxy Release (v 13.08)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.11 June 2013 Galaxy Release (v 13.06)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.12 April 2013 Galaxy Release (v 13.04)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.13 February 2013 Galaxy Release (v 13.02)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.14 January 2013 Galaxy Release (v 13.01)

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

1.3.15 Galaxy Releases older than v 13.01

Please see the Galaxy wiki for announcement and release notes.

To stay up to date with Galaxy's progress watch our screencasts, read our wiki, and follow @galaxyproject on Twitter.

Thanks for using Galaxy!

The Galaxy Team

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

g

l

m

p

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z