MuG DMP API Documentation

Release 0

Mark McDowall

Aug 15, 2018

Table of Contents

CHAPTER 1

Requirements and Installation

1.1 Requirements

1.1.1 Software

- Mongo DB 3.2
- Python 2.7.10+

1.1.2 Python Modules

- pymongo
- mongomock
- h5py
- numpy
- pyBigWig
- pysam

1.2 Installation

Directly from GitHub:

```
git clone https://github.com/Multiscale-Genomics/mg-dm-api.git
```

Using pip:

```
pip install git+https://github.com/Multiscale-Genomics/mg-dm-api.git
```

1.3 Documentation

To build the documentation:

```
pip install Sphinx
pip install sphinx-autobuild
cd docs
make html
```

CHAPTER 2

Data Management Plan API

2.1 Methods

class dmp.dmp.**dmp**(*cnf_loc=u”*, *test=False*)

API for management of files within the VRE

add_file_metadata(*user_id*, *file_id*, *key*, *value*)

Add a key value pair to the meta data for a file

This way a user is able to add extra information to the meta data to better describe the file.

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to “common” if the files can be shared between users
- **file_id** (*str*) – ID of the file. This is the value returned when a file is loaded into the DMP or is the *_id* for a given file when the files have been retrieved.
- **key** (*str*) – Unique key for the identification of the extra meta data. If the key matches a value already in the meta data then it over-writes the current value.
- **value** – Value to be stored for the given key. This can be a str, int, list or dict.

Returns This is an id for that file within the system and can be used for tracing this file and where it is used and where it has come from.

Return type str

get_file_by_file_path(*user_id*, *file_path*, *rest=False*)

Get a list of the file dictionary objects given a *user_id* and *file_path*

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to “common” if the files can be shared between users
- **file_path** (*str*) – File path (see validate_file)

Returns

file_path [str] Location of the file in the file system

file_type [str] File format (see validate_file)

data_type [str] The type of information in the file (RNA-seq, ChIP-seq, etc)

taxon_id [int] Taxon ID that the species that the file has been derived from

compressed [str] Type of compression (None, gzip, zip)

source_id [list] List of IDs of files that were processed to generate this file

meta_data [dict] Dictionary object containing the extra data related to the generation of the file or describing the way it was processed

creation_time [list] Time at which the file was loaded into the system

Return type dict

Example

```
from dmp import dmp
da = dmp()
da.get_files_by_file_path(<user_id>, <file_type>)
```

get_file_by_id(*user_id*, *file_id*, *rest=False*)

Returns files data based on the unique_id for a given file

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users
- **file_id** (*str*) – Location of the file in the file system

Returns

file_path [str] Location of the file in the file system

path_type [str] File or Folder

file_type [str] File format (see validate_file)

size [int] Size of the file

parent_dir [str] Location of the parent dir

data_type [str] The type of information in the file (RNA-seq, ChIP-seq, etc)

taxon_id [int] Taxon ID that the species that the file has been derived from

compressed [str] Type of compression (None, gzip, zip)

source_id [list] List of IDs of files that were processed to generate this file

meta_data [dict] Dictionary object containing the extra data related to the generation of the file or describing the way it was processed

creation_time [list] Time at which the file was loaded into the system

Return type dict

Example

```
from dmp import dmp
da = dmp()
da.get_file_by_id(<unique_file_id>)
```

get_file_history(*user_id*, *file_id*)
Returns the full path of file_ids from the current file to the original file(s)

Needs work to define the format for how declaring the history is best

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users
- **file_id** (*str*) – ID of the file. This is the value returned when a file is loaded into the DMP or is the *_id* for a given file when the files have been retrieved.

Returns List of lists representing the adjancency of child and parent files.

Return type list

Example

```
from dmp import dmp
da = dmp()
history = da.get_file_history("aLongString")
print history
```

Output: `[['aLongString', 'parentOfaLongString'], ['parentOfaLongString', 'parentOfParent']]`

These IDs can then be requested to ruturn the meta data and locations with the *get_file_by_id* method.

get_files_by_assembly(*user_id*, *assembly*, *rest=False*)
Get a list of the file dictionary objects given a *user_id* and *assembly*

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users
- **assembly** (*str*) – Assembly that the species that the file has been derived from

Returns

file_path [str] Location of the file in the file system

file_type [str] File format (see validate_file)

data_type [str] The type of information in the file (RNA-seq, ChIP-seq, etc)

taxon_id [int] Taxon ID that the species that the file has been derived from

compressed [str] Type of compression (None, gzip, zip)

source_id [list] List of IDs of files that were processed to generate this file

meta_data [dict] Dictionary object containing the extra data related to the generation of the file or describing the way it was processed

creation_time [list] Time at which the file was loaded into the system

Return type dict

Example

```
from dmp import dmp
da = dmp()
da.get_files_by_taxon_id(<user_id>, <taxon_id>)
```

get_files_by_data_type(*user_id*, *data_type*, *rest=False*)
Get a list of the file dictionary objects given a *user_id* and *data_type*

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users
- **data_type** (*str*) – The type of information in the file (RNA-seq, ChIP-seq, etc)

Returns

file_path [str] Location of the file in the file system

file_type [str] File format (see validate_file)

data_type [str] The type of information in the file (RNA-seq, ChIP-seq, etc)

taxon_id [int] Taxon ID that the species that the file has been derived from

compressed [str] Type of compression (None, gzip, zip)

source_id [list] List of IDs of files that were processed to generate this file

meta_data [dict] Dictionary object containing the extra data related to the generation of the file or describing the way it was processed

creation_time [list] Time at which the file was loaded into the system

Return type dict

Example

```
from dmp import dmp
da = dmp()
da.get_files_by_data_type(<user_id>, <data_type>)
```

get_files_by_file_type(*user_id*, *file_type*, *rest=False*)
Get a list of the file dictionary objects given a *user_id* and *file_type*

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users
- **file_type** (*str*) – File format (see validate_file)

Returns

file_path [str] Location of the file in the file system

file_type [str] File format (see validate_file)

data_type [str] The type of information in the file (RNA-seq, ChIP-seq, etc)

taxon_id [int] Taxon ID that the species that the file has been derived from

compressed [str] Type of compression (None, gzip, zip)

source_id [list] List of IDs of files that were processed to generate this file

meta_data [dict] Dictionary object containing the extra data related to the generation of the file or describing the way it was processed

creation_time [list] Time at which the file was loaded into the system

Return type dict

Example

```
from dmp import dmp
da = dmp()
da.get_files_by_file_type(<user_id>, <file_type>)
```

get_files_by_taxon_id(*user_id*, *taxon_id*, *rest=False*)

Get a list of the file dictionary objects given a *user_id* and *taxon_id*

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users
- **taxon_id** (*int*) – Taxon ID that the species that the file has been derived from

Returns

file_path [str] Location of the file in the file system

file_type [str] File format (see validate_file)

data_type [str] The type of information in the file (RNA-seq, ChIP-seq, etc)

taxon_id [int] Taxon ID that the species that the file has been derived from

compressed [str] Type of compression (None, gzip, zip)

source_id [list] List of IDs of files that were processed to generate this file

meta_data [dict] Dictionary object containing the extra data related to the generation of the file or describing the way it was processed

creation_time [list] Time at which the file was loaded into the system

Return type dict

Example

```
from dmp import dmp
da = dmp()
da.get_files_by_taxon_id(<user_id>, <taxon_id>)
```

get_files_by_user(*user_id*, *rest=False*)

Get a list of the file dictionary objects given a *user_id*

Parameters **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users

Returns List of dict objects for each file that has been loaded by a user.

Return type list

Example

```
from dmp import dmp
da = dmp()
da.get_files_by_user(<user_id>)
```

modify_column (*user_id*, *file_id*, *key*, *value*)
Update a key value pair for the record

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users
- **file_id** (*str*) – ID of the file. This is the value returned when a file is loaded into the DMP or is the *_id* for a given file when the files have been retrieved.
- **key** (*str*) – Unique key for the identification of the extra meta data. If the key matches a value already in the meta data then it over-writes the current value.
- **value** – Value to be stored for the given key. This can be a str, int, list or dict.

Returns This is an id for that file within the system and can be used for tracing this file and where it is used and where it has come from.

Return type str

remove_file (*user_id*, *file_id*)
Removes a single file from the directory. Returns the ID of the file that was removed

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users
- **file_id** (*str*) – ID of the file. This is the value returned when a file is loaded into the DMP or is the *_id* for a given file when the files have been retrieved.

Returns The file_id of the removed file.

Return type str

Example

```
from dmp import dmp
da = dmp()
da.remove_file(<file_id>)
```

remove_file_metadata (*user_id*, *file_id*, *key*)
Remove a key value pair from the meta data for a given file

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users

- **file_id** (*str*) – ID of the file. This is the value returned when a file is loaded into the DMP or is the *_id* for a given file when the files have been retrieved.
- **key** (*str*) – Unique key for the identification of the extra meta data to be removed

Returns This is an id for that file within the system and can be used for tracing this file and where it is used and where it has come from.

Return type str

set_file(*user_id*, *file_path*, *path_type*, *file_type=u"*, *size=0*, *parent_dir=u"*, *data_type=u"*, *taxon_id=u"*, *compressed=None*, *source_id=None*, *meta_data=None*, ***kwargs*)

Adds a file to the data management API.

Parameters

- **user_id** (*str*) – Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users
- **file_path** (*str*) – Location of the file in the file system
- **path_type** (*str*) –
- **parent_dir** (*str*) – _id of the parent directory
- **file_type** (*str*) – File format (see validate_file)
- **size** (*int*) – File size in bytes
- **data_type** (*str*) – The type of information in the file (RNA-seq, ChIP-seq, etc)
- **taxon_id** (*int*) – Taxon ID that the species that the file has been derived from
- **compressed** (*str*) – Type of compression (None, gzip, zip)
- **source_id** (*list*) – List of IDs of files that were processed to generate this file
- **meta_data** (*dict*) – Dictionary object containing the extra data related to the generation of the file or describing the way it was processed

  assembly [string] Dependent paramenter. If the sequence has been aligned at some point during the production of this file then the assembly must be recorded.

Returns This is an id for that file within the system and can be used for tracing this file and where it is used and where it has come from.

Return type str

Example

```
from dmp import dmp
da = dmp()
unique_file_id = da.set_file(
    'user1', '/tmp/example_file.fastq', 'fastq', 'RNA-seq', 9606, None)
```

If there is a processed result of 1 or more files then these can be specified using the file_id:

```
>>> da.set_file(
    'user1', '/tmp/example_file.fastq', 'fastq', 'RNA-seq', 9606, None,
    source_id=[1, 2])
```

Meta data about the file can also be included to provide extra information about the file, origins or how it was generated:

```
>>> da.set_file('user1', '/tmp/example_file.fastq', 'fastq', 'RNA-seq',
    9606, None, meta_data={'assembly' : 'GCA_0000nnnn',
    'downloaded_from' : 'http://www.', })
```

static validate_file(*entry*)

Validate that the required meta data for a given entry is present. If there is missing data then a ValueError excepetion is raised. This function checks that all required paths are defined and that when various selections are made then the correct matching data is also present

Parameters **entry** (*dict*) –

user_id [str] Identifier to uniquely locate the users files. Can be set to "common" if the files can be shared between users

file_path [str] Location of the file in the file system

path_type [str] File or folder

file_type [str] File format ("amb", "ann", "bam", "bb", "bed", "bt2", "bw", "bwt", "cpt", "csv", "dcd", "fa", "fasta", "fastq", "gem", "gff3", "gz", "hdf5", "json", 'lif', "pac", "pdb", "pdf", "png", "prmtop", "sa", "tbi", "tif", "tpr", "trj", "tsv", "txt", "wig")

size [int] Size of the file in bytes

data_type [str] The type of information in the file (RNA-seq, ChIP-seq, etc)

taxon_id [int] Taxon ID that the species that the file has been derived from

compressed [str] Type of compression (None, gzip, zip)

source_id [list] List of IDs of files that were processed to generate this file

meta_data [dict] Dictionary object containing the extra data related to the generation of the file or describing the way it was processed assembly : string

Returns

- *bool* – Returns True if there are no errors with the entry
- *If there are issues with the entry then a ValueError is raised.*

CHAPTER 3

Data Management RESTful API

3.1 Methods

class dmp.rest.**rest**(*cnf_loc=''*, *test=False*)
API for management of files within the VRE

add_service(*name*, *url*, *description*, *status=None*)
Add a service to the registry

Parameters

- **name** (*str*) – Unique name for the service
- **description** (*str*) – Description defined by the service
- **url** (*str*) – Base URL for the REST service.
- **status** (*str*) – Service HTTP status code - *up* or *down*

Returns Entry ID

Return type str

get_available_services()
List all services

Returns List of dict objects for each service

Return type list

get_down_services()
List services that are NOT returning HTTP code 200

Returns List of dict objects for each service

Return type list

get_service(*name*)
Retreive the full details about a service

Parameters **name** (*str*) – Unique name for the service

Returns

name: str Unique name for the service

description: str Description defined by the service

url: str Base URL for the RESET service.

status: str Service HTTP status code - *up* or *down*

Return type dict

get_up_services()

List services that are returning HTTP code 200

Returns List of dict objects for each service

Return type list

is_service(*name*)

Identify if a service is already present in the registry

Parameters **name** (*str*) – Unique name for the service

set_service_status(*name*, *status*)

Update the status of the service if it is already present in the db.

Parameters

- **name** (*str*) – Unique name for the service
- **status** (*str*) – Service HTTP status code - *up* or *down*

Returns *True* when done

Return type bool

update_service_url(*name*, *url*)

Update the url of the service if it is already present in the db.

Parameters

- **name** (*str*) – Unique name for the service
- **url** (*str*) – Base URL for the REST service.

Returns *True* when done

Return type bool

CHAPTER 4

Custom Reader APIs

4.1 HDF5 Files

4.1.1 Hi-C Adjacency Files

class reader.hdf5_adjacency.**adjacency**(*user_id*, *file_id*, *resolution=None*, *cnf_loc=''*)

Class related to handling the functions for interacting directly with the HDF5 files. All required information should be passed to this class.

close()

Close the HDF5 data file handle

get_chromosome_from_array_index(*index*)

Identify the chromosome based on either the x or y coordinate in the array.

Parameters **index** (*int*) – Location within the array

Returns **chr_id** – Identity of the chromosome

Return type str

Example

```
from reader import adjacency
r = adjacency('test', '', 10000)
cid = r.get_chromosome_from_array_index(1234567890)
```

get_chromosome_parameters()

Return a list of the available resolutions in a given HDF5 file

Returns chromosomes : list chr_param : dict resolitions

Return type dict

Example

```
from reader import adjacency
r = adjacency('test', '', 10000)
value = r.get_chromosome_parameters()
```

get_chromosomes()
List of chromosomes that have models at a given resolution

Returns **chromosomes** – List of chromosomes at the set resolution

Return type list

get_details()
Return a list of the available resolutions in a given HDF5 file

get_range(*chr_id*, *start*, *end*, *limit_chr=None*, *limit_start=None*, *limit_end=None*, *value_url='/api/getValue'*, *no_links=None*)
Get the interactions that happen within a defined region on a specific chromosome. Returns inter and intra interactions with the defined region.

Parameters

- **chr_id** (*str*) – Chromosomal name
- **start** (*int*) – Start position within the chromosome
- **end** (*int*) – End position within the chromosome
- **limit_chr** (*str (Optional)*) – Limit the results to a particular chromosome
- **limit_start** (*int (Optional)*) – Limit the range start position on the limit_chr paramter
- **limit_end** (*int (Optional)*) – Limit the range end position on the limit_chr parameter
- **value_url** (*str (Optional)*) – Define a custom URL snippet for the location of the file if different from the defaul
- **no_links** (*bool (Optional)*) – Will return the URL links to the individual points within the adjacency matrix. In cases where this generates a large number of points it is possible to turn off generating these links. Set this value to 1.

Returns

log [list] List of messages about the state for debugging

results [list] List of values for given positions within the adjacency matrix

Return type dict

Example

```
from reader import adjacency
r = adjacency('test', '', 10000)
value = r.get_range(2000000, 1000000)
```

get_resolution()
List the current level of rseolution

Returns **resolution** – Current level of resolution

Return type int

get_resolutions()
List resolutions that models have been generated for

Returns **list** – Available levels of resolution that can be set

Return type str

get_value(*bin_i*, *bin_j*)
Get a specific value for a given dataset, resolution

Parameters

- **bin_i** (*int*) – Array position in the first dimension
- **bin_j** (*int*) – Array position in the second dimension

Returns **value** – Value for a given cell in the adjacency array

Return type int

Example

```
from reader import adjacency
r = adjacency('test', '', 10000)
value = r.get_value(2000000, 1000000)
```

set_resolution(*resolution*)
Set, or change, the resolution level

Parameters **resolution** (*int*) – Level of resolution

4.1.2 Hi-C Coordinate Files

class reader.hdf5_coord.**coord**(*user_id*, *file_id*, *resolution=None*, *cnf_loc=''*)
Class related to handling the functions for interacting directly with the HDF5 files. All required information should be passed to this class.

close()
Tidy function to close file handles

get_centroids(*region_id*)
List the centroid models for each cluster

Returns **centroids** – List of the centroid models for each cluster

Return type list

get_chromosomes()
List of chromosomes that have models at a given resolution

Returns **chromosomes** – List of chromosomes at the set resolution

Return type list

get_clusters(*region_id*)
List all clusters of models

Returns **clusters** – List of models in each cluster

Return type list

get_model(*region_id*, *model_ids=None*, *page=0*, *mpp=10*)

Get the coordinates within a defined region on a specific chromosome. If the model_id is not returned the the consensus models for that region are returned

Parameters

- **region_id** (*str*) – Region ID
- **model_ids** (*list*) – List of model IDs for the models that are required
- **page** (*int*) – Page number
- **mpp** (*int*) – Number of models per page (default: 10; max: 100)

Returns

array –

model [dict]

metadata [dict] Relevant extra meta data added by TADbit

object [dict] Key value pair of information about the region

models [list] List of dictionaries for each model

clusters [list] List of models for each cluster

centroids [list] List of all centroid models

restraints [list] List of retraints for each position

hic_data [dict] Hi-C model data

metadata [dict]

model_count [int] Count of the number of models for the defined region ID

page_count [int] Number of pages

Return type list

get_models(*region_id*)

List all models for a given region

Returns model_id : int cluster_id : int

Return type List

get_object_data(*region_id*)

Prepare the object header data structure ready for printing

Parameters **region_id** (*int*) – Region that is getting downloaded

Returns **objectdata** – All headers and values required for the JSON output

Return type dict

get_region_order(*chr_id=None*, *region=None*)

List the regions on a given chromosome ID or region ID in the order that they are located on the chromosome

Parameters

- **chr_id** (*str*) – Chromosome ID
- **region** (*str*) – Region ID

Returns

region_id [str] List of the region IDs

Return type list

get_regions(*chr_id*, *start*, *end*)
List regions that are within a given range on a chromosome

Parameters

- **chr_id** (*str*) – Chromosome ID
- **start** (*int*) – Start position
- **end** (*int*) – Stop position

Returns **regions** – List of region IDs whose parameters match those provided

Return type list

get_resolution()
List the current level of rseolution

Returns **resolution** – Current level of resolution

Return type int

get_resolutions()
List resolutions that models have been generated for

Returns **list** – Available levels of resolution that can be set

Return type str

set_resolution(*resolution*)
Set, or change, the resolution level

Parameters **resolution** (*int*) – Level of resolution

4.1.3 Text File Index

Lists all files that are available for a user in bed and wig formats and lists the files than have data in a given region so that only the required files are requested by the client

class reader.hdf5_reader.**hdf5_reader**(*user_id*, *file_id*, *cnf_loc=''*)
Class related to handling the functions for interacting directly with the HDF5 files. All required information should be passed to this class.

close()
Tidy function to close file handles

Example

```
from hdf5_reader import hdf5_reader
h5r = hdf5_reader('test')
h5r.close()
```

get_assemblies()
List all assemblies for which there are files that have been indexed

Returns **assembly** – List of assemblies in the index

Return type list

Example

```
from hdf5_reader import hdf5_reader
h5r = hdf5_reader('test')
h5r.assemblies()
```

get_chromosomes (*assembly*)
List all chromosomes that are covered by the index

Parameters **assembly** (*str*) – Genome assembly ID

Returns **chromosomes** – List of the chromosomes for a given assembly in the index

Return type list

Example

```
from hdf5_reader import hdf5_reader
h5r = hdf5_reader('test')
asm = h5r.assemblies()
chr_list = h5r.get_chromosomes(asm[0])
```

get_files (*assembly*)
List all files for an assembly. If files are missing they can either get loaded or the search can be performed directly on the bigBed files

Parameters **assembly** (*str*) – Genome assembly ID

Returns **file_ids** – List of file ids for a given assembly in the index

Return type list

Example

```
from hdf5_reader import hdf5_reader
h5r = hdf5_reader('test')
asm = h5r.assemblies()
file_list = h5r.get_files(asm[0])
```

get_regions (*assembly*, *chromosome_id*, *start*, *end*)
List files that have data in a given region.

Parameters

- **assembly** (*str*) – Genome assembly ID
- **chromosome_id** (*str*) – Chromosome names as listed by the get_files function
- **start** (*int*) – Start position for the region of interest
- **end** (*int*) – End position for the region of interest

Returns **file_ids** – List of the file_ids that have sequence features within the region of interest

Return type list

Example

```
from hdf5_reader import hdf5_reader
h5r = hdf5_reader('test')
asm = h5r.assemblies()
file_list = h5r.get_chromosomes(asm[0], 1, 1000000, 1100000)
```

4.2 BigBed Files

class reader.bigbed.**bigbed_reader**(*user_id*, *file_id*, *cnf_loc=''*)

Class related to handling the functions for interacting directly with the BigBed files. All required information should be passed to this class.

close()

Tidy function to close file handles

Example

```
from reader.bigbed import bigbed_reader
bbr = bigbed_reader('test')
bbr.close()
```

get_chromosomes()

List the chromosome names and lengths

Returns **chromosomes** – Key value pair of chromosome name and the value is the length of the chromosome.

Return type dict

get_header()

Get the bigBed header

Returns **header**

Return type dict

get_range(*chr_id*, *start*, *end*, *file_type='bed'*)

Get entries in a given range

Parameters

- **chr_id** (*string*) – Chromosome name
- **start** (*int*) – Start of the region to query
- **end** (*int*) – End of the region to query
- **file_type** (*string (OPTIONAL)*) – *bed* format returning the whole file as a string is the default option. *list* will return the bed rows but as a list of lists.

Returns

- **bed** (*str (DEFAULT)*) – List of strings for the rows in a bed file
- **bed_array** (*list*) – List of lists of each row for the bed file format

4.3 BigWig Files

class reader.bigwig.**bigwig_reader**(*user_id*, *file_id*, *cnf_loc=''*)

Class related to handling the functions for interacting directly with the BigBed files. All required information should be passed to this class.

get_chromosomes()

List the chromosome names and lengths

Returns **chromosomes** – Key value pair of chromosome name and the value is the length of the chromosome.

Return type dict

get_header()

Get the bigWig header

Returns **header**

Return type dict

get_range(*chr_id*, *start*, *end*, *file_type='wig'*)

Get entries in a given range

Parameters

- **chr_id** (*string*) – Chromosome name
- **start** (*int*) – Start of the region to query
- **end** (*int*) – End of the region to query
- **format** (*string (OPTIONAL)*) – *wig* format returning the whole file as a string is the default option. *list* will return the wig rows but as a list of lists.

Returns

- **wig** (*str (DEFAULT)*) – List of strings for the rows in a wig file
- **wig_array** (*list*) – List of lists of each row for the wig file format

4.4 Tabix Files

class reader.tabix.**tabix**(*user_id*, *file_id*, *cnf_loc=''*)

Class related to handling the functions for interacting directly with the BigBed files. All required information should be passed to this class.

get_range(*chr_id*, *start*, *end*, *file_type='gff3'*)

Get entries in a given range

Parameters

- **chr_id** (*string*) – Chromosome name
- **start** (*int*) – Start of the region to query
- **end** (*int*) – End of the region to query
- **format** (*string (OPTIONAL)*) – *gff3* format returning the whole file as a string is the default option. *list* will return the gff3 rows but as a list of lists.

Returns

- **gff3** (*str (DEFAULT)*) – List of strings for the rows in a gff3 file
- **wig_array** (*list*) – List of each row for the gff3 file format

CHAPTER 5

License

Apache License Version 2.0, January 2004 http://www.apache.org/licenses/

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

> To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "{}" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

r

Index