epivizfileserver Documentation

Release unknown

Jayaram Kancherla

Feb 12, 2020

Contents

Epiviz file Server is a scalable data query and compute system for indexed genomic files. In addition to querying data, users can also compute transformations, summarization and aggregation using NumPy functions directly on data queried from files.

Since the genomic files are indexed, the library will only request and parse necessary bytes from these files to process the request (without loading the entire file into memory). We implemented a cache system to efficiently manage already accessed bytes of a file. We also use dask to parallelize computing requests for query and transformation. This allows us to process and scale our system to large data repositories.

This blog post (Jupyter notebook) describes various features of the file server library using genomic files hosted from the NIH Roadmap Epigenomics project.

The library provides various modules to

- Parser: Read various genomic file formats,
- Query: Access only necessary bytes of file for a given genomic location,
- Compute: Apply transformations on data,
- Server: Instantly convert the datasets into a REST API
- Visualization: Interactive Exploration of data using Epiviz (uses the Server module above).

Note:

- The Epiviz file Server is an open source project on GitHub
- Let us know what you think and any feedback or feature requests to improve the library!

CHAPTER 1

Contents

1.1 Installation

1.1.1 Using PyPI

To install the package from PyPi,

```
pip install epivizfileserver
```

1.1.2 Development Version

To install the development version from GitHub: Install using pip

```
pip install git@github.com:epiviz/epivizFileParser.git
```

you can also clone the repository and install from local directory using *pip*

Note: If you don't have sudo rights to install the package, you can install it to the user directory using

```
pip install --user epivizfileserver
```

1.2 Tutorial

This blog post (Jupyter notebook) describes various features of the file server library using genomic files hosted from the NIH Roadmap Epigenomics project.

Note: This post describes a general walkthrough of the features of the file server. More usecases will be posted soon!

1.2.1 Import Measurements from File

Since large data repositories contains hundreds of files, manually adding files would be cumbersome. In order to make this process easier, we create a configuration file that lists all files with their locations. An example configuration file is described below -

1.2.2 Configuration file

The following is a configuration file for data hosted on the roadmap FTP server. This contains data for ChIP-seq experiments for the H3k27me3 marker in *Esophagus* and *Sigmoid Colon* tissues. Most fields in the configuration file are self explanatory.

```
[
    {
        url: "https://egg2.wustl.edu/roadmap/data/byFileType/signal/consolidated/
↪macs2signal/foldChange/E079-H3K27me3.fc.signal.bigwig",
        file_type: "bigwig",
        datatype: "bp",
        name: "E079-H3K27me3",
        id: "E079-H3K27me3",
        annotation: {
            group: "digestive",
            tissue: "Esophagus",
            marker: "H3K27me3"
        }
    }, {
        url: "https://egg2.wustl.edu/roadmap/data/byFileType/signal/consolidated/
↪macs2signal/foldChange/E106-H3K27me3.fc.signal.bigwig",
        file_type: "bigwig",
        datatype: "bp",
        name: "E106-H3K27me3",
        id: "E106-H3K27me3",
        annotation: {
            group: "digestive",
            tissue: "Sigmoid Colon",
            marker: "H3K27me3"
        }
    }
]
```

Once the configuration file is generated, we can import these measurements into the file server. We first create a *MeasurementManager* object which handles measurements from files and databases. We can then use the helper function *import_files* to import all measurements from this configuration file.

```
mMgr = MeasurementManager()
fmeasurements = mMgr.import_files(os.getcwd() + "/roadmap.json", mHandler)
fmeasurements
```

1.2.3 Query for a genomic location

After loading the measurements, we can query the object for data in a particular genomic region using the *get_data* function.

```
result, err = await fmeasurements[1].get_data("chr11", 10550488, 11554489)
result.head()
```

The reponse is a tuple, DataFrame that contains all results and an error if there is any.

1.2.4 Compute a Function over files

We can define and create new measurements that can be computed using a *Numpy* function over the files loaded from the previous step.

Note: you can also write a custom statistical function, that applies to every row in the DataFrame. It must follow the same syntax as any *Numpy* row-apply function.

As an example to demonstrate, we can calculate the average ChIP-seq expression for *H3K27me3* marker.

```
computed_measurement = mMgr.add_computed_measurement("computed", "avg_ChIP_seq",
↪"Average ChIP seq expression",
                                                      measurements=fmeasurements, computeFunc=numpy.
↪mean)
```

After defining a computed measurement, we can query this measurement for a genomic location.

```
result, err = await computed_measurement.get_data("chr11", 10550488, 11554489)
result.head()
```

1.2.5 Setup a REST API

Often times, developers would like to include data from genomic files into a web application for visualization or into their workflows. We can quickly setup a REST API web server from the measurements we loaded -

```
from epivizfileserver import setup_app
    app = setup_app(mMgr)
    app.run(port=8000)
```

The REST API is an asynchronous web server that is built on top of SANIC.

1.2.6 Query Files from AnnotationHub

We can also use the Bioconductor's AnnotationHub to search for files and setup the file server. We are working on simplifying this process.

Annotation Hub API is hosted at https://annotationhub.bioconductor.org/.

We first download the annotationhub sqlite database for available data resources.

```
wget http://annotationhub.bioconductor.org/metadata/annotationhub.sqlite3
```

After download the resource database from AnnotatiobnHub, we can now load the sqlite database into python and query for datasets.

```
import pandas
import os
import sqlite3

conn = sqlite3.connect("annotationhub.sqlite3")
```

(continues on next page)

(continued from previous page)

```
cur = conn.cursor()
cur.execute("select * from resources r JOIN input_sources inp_src ON r.id = inp_src.
↪resource_id;")
results = cur.fetchall()
pd = pandas.DataFrame(results, columns = ["id", "ah_id", "title", "dataprovider",
↪"species", "taxonomyid", "genome",
                                          "description", "coordinate_1_based",
↪"maintainer", "status_id",
                                          "location_prefix_id", "recipe_id",
↪"rdatadateadded", "rdatadateremoved",
                                          "record_id", "preparerclass", "id",
↪"sourcesize", "sourceurl", "sourceversion",
                                          "sourcemd5", "sourcelastmodifieddate",
↪"resource_id", "source_type"])
pd.head()
```

For the purpose of the tutorial, we will filter for Sigmoid Colon ("E106") and Esophagus ("E079") tissues, and the ChipSeq Data for "H3K27me3" histone marker files from the roadmap epigenomics project.

```
roadmap = pd.query('dataprovider=="BroadInstitute" and genome=="hg19"')
roadmap = roadmap.query('title.str.contains("H3K27me3") and (title.str.contains("E106
↪") or title.str.contains("E079"))')
# only use fc files
roadmap = roadmap.query('title.str.contains("fc")')
roadmap
```

After filtering for resources we are interested in, we can load them into the file server using the *import_ahub* helper function.

```
mMgr = MeasurementManager()
ahub_measurements = mMgr.import_ahub(roadmap)
ahub_measurements
```

The rest of the process is similar as described in the beginning of this tutorial.

1.3 License

1.4 Contributors

- Jayaram Kancherla <jayaram.kancherla@gmail.com>
- Yifan Yang <yang7832@umd.edu>
- Hector Corrada Bravo <hcorrada@gmail.com>

1.5 Changelog

1.5.1 Version 0.1.2

- Package pushed to PyPi
- Updated readme and documentation

1.5.2 Version 0.1

- First release!

1.6 epivizfileserver

1.6.1 epivizfileserver package

Subpackages

epivizfileserver.client package

Submodules

epivizfileserver.client.EpivizClient module

class epivizfileserver.client.EpivizClient.**EpivizClient**(*server*)
Bases: object

Client implementation of the epiviz server

Parameters **server** – endpoint where the API is running

get_data(*measurement*, *chr*, *start*, *end*)
Get data for a genomic region from the API

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end

Returns a json with results

get_measurements()

get_seq_info()

version = 5

Module contents

epivizfileserver.handler package

Submodules

epivizfileserver.handler.handler module

class epivizfileserver.handler.handler.**FileHandlerProcess**(*fileTime*, *MAXWORKER*)

Bases: object

Class to manage query, transformation and cache using dask distributed

Parameters

- **fileTime** (*int*) – time to keep file objects in memory
- **MAXWORKER** (*int*) – maximum workers that can be used

records
a dictionary of all file objects

client
asynchronous dask server client

binFileData(*fileName*, *data*, *chr*, *start*, *end*, *bins*, *columns*, *metadata*)
submit tasks to the dask client

cleanFileOBJ()
automated task to pickle all fileobjects to disk

getRecord(*name*)
get file object from *records* by name

Parameters **name** (*str*) – file name

Returns file object

handleFile(*fileName*, *fileType*, *chr*, *start*, *end*, *bins=2000*)
submit tasks to the dask client

Parameters

- **fileName** – file location
- **fileType** – file type
- **chr** – chromosome
- **start** – genomic start
- **end** – genomic end
- **points** – number of base-pairse to group per bin

pickleFileObject(*fileName*)
automated task to load a pickled file object

Parameters **fileName** – file name to load

setRecord(*name*, *fileObj*, *fileType*)
add or update *records* with new file object

Parameters

- **name** (*str*) – file name
- **fileObj** – file object
- **fileType** – file type

epivizfileserver.handler.utils module

epivizfileserver.handler.utils.**create_parser_object**(*format*, *source*)
Create appropriate File class based on file format

Parameters

- **format** – Type of file
- **request** – Other request parameters

Returns An instance of parser class

Module contents

epivizfileserver.measurements package

Submodules

epivizfileserver.measurements.measurementClass module

class epivizfileserver.measurements.measurementClass.**ComputedMeasurement**(*mtype*, *mid*, *name*, *measurements*, *source='computed'*, *computeFunc=None*, *datasource='computed'*, *annotation={'group': 'computed'}*, *metadata=None*, *isComputed=True*, *isGenes=False*, *fileHandler=None*, *columns=None*, *computeAxis=1*)

Bases: *epivizfileserver.measurements.measurementClass.Measurement*

Class for representing computed measurements

In addition to params on base *Measurement* class -

Parameters

- **computeFunc** – a *NumPy* function to apply on our dataframe
- **source** – defaults to 'computed'
- **datasource** – defaults to 'computed'

computeWrapper(*computeFunc*, *columns*)

a wrapper for the 'computeFunc' function

Parameters

- **computeFunc** – a *NumPy* compute function
- **columns** – columns from file to apply

Returns a dataframe with results

get_columns()

get columns from file

get_data(*chr*, *start*, *end*, *bins*, *dropna=True*)
Get data for a genomic region from files and apply the *computeFunc* function

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **dropna** (*bool*) – True to dropna from a measurement since any computation is going to fail on this row

Returns a dataframe with results

class epivizfileserver.measurements.measurementClass.**DbMeasurement**(*mtype*, *mid*, *name*, *source*, *datasource*, *dbConn*, *annotation=None*, *metadata=None*, *isComputed=False*, *isGenes=False*, *minValue=None*, *maxValue=None*, *columns=None*)
Bases: *epivizfileserver.measurements.measurementClass.Measurement*

Class representing a database measurement

In addition to params from the base measurement class -

Parameters **dbConn** – a database connection object

connection
a database connection object

get_data(*chr*, *start*, *end*, *bin=False*)
Get data for a genomic region from database

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **bin** (*bool*) – True to bin the results, defaults to False

Returns a dataframe with results

query(*obj*, *params*)
Query from db/source

Parameters

- **obj** – the query string
- **query_params** – query parameters to search

Returns a dataframe of results from the database

class epivizfileserver.measurements.measurementClass.**FileMeasurement**(*mtype*, *mid*, *name*, *source*, *datasource='files'*, *annotation=None*, *metadata=None*, *isComputed=False*, *isGenes=False*, *minValue=None*, *maxValue=None*, *fileHandler=None*, *columns=None*)

Bases: *epivizfileserver.measurements.measurementClass.Measurement*

Class for file based measurement

In addition to params from the base *Measurement* class

Parameters **fileHandler** – an optional file handler object to process query requests (uses dask)

create_parser_object(*type*, *name*, *columns=None*)

Create appropriate File class based on file format

Parameters

- **type** (*str*) – format of file
- **name** (*str*) – location of file
- **columns** (*[str]*) – list of columns from file

Returns An file object

get_data(*chr*, *start*, *end*, *bins*, *bin=True*)

Get data for a genomic region from file

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **bin** (*bool*) – True to bin the results, defaults to False

Returns a dataframe with results

class epivizfileserver.measurements.measurementClass.**Measurement**(*mtype*, *mid*, *name*, *source*, *datasource*, *annotation=None*, *metadata=None*, *isComputed=False*, *isGenes=False*, *minValue=None*, *maxValue=None*, *columns=None*)

Bases: object

Base class for managing measurements from files

Parameters

- **mtype** – Measurement type, either 'file' or 'db'
- **mid** – unique id to use for this measurement
- **name** – name of the measurement
- **source** – location of the measurement, if mtype is 'db' use table name, if file, file location
- **datasource** – is the database name if mtype is 'db' use database name, else 'files'
- **annotation** – annotation for this measurement, defaults to None
- **metadata** – metadata for this measurement, defaults to None
- **isComputed** – True if this measurement is Computed from other measurements, defaults to False
- **isGenes** – True if this measurement is an annotation (for example: reference genome hg19), defaults to False
- **minValue** – min value of all values, defaults to None
- **maxValue** – max value of all values, defaults to None
- **columns** – column names for the file

bin_rows(*data*, *chr*, *start*, *end*, *bins=2000*)

Bin genome by bin length and summarize the bin

Parameters

- **data** – DataFrame from the file
- **chr** – chromosome
- **start** – genomic start
- **end** – genomic end
- **length** – max rows to summarize the data frame into

Returns a binned data frame whose max rows is length

get_columns()
get columns from file

get_data(*chr*, *start*, *end*)
Get Data for this measurement

Parameters

- **chr** – chromosome
- **start** – genomic start
- **end** – genomic end

get_measurement_annotation()
Get measurement annotation

get_measurement_id()
Get measurement id

get_measurement_max()
Get measurement max value

get_measurement_metadata()
Get measurement metadata

get_measurement_min()
Get measurement min value

get_measurement_name()
Get measurement name

get_measurement_source()
Get source

get_measurement_type()
Get measurement type

is_computed()
Is measurement computed ?

is_file()
Is measurement a file ?

is_gene()
is the file a genome annotation ?

query(*obj*, *query_params*)
Query from db/source

Parameters

- **obj** – db obj
- **query_params** – query parameters to search

class epivizfileserver.measurements.measurementClass.**WebServerMeasurement**(*mtype*, *mid*, *name*, *source*, *datasource*, *datasourceGroup*, *annotation=None*, *metadata=None*, *isComputed=False*, *isGenes=False*, *minValue=None*, *maxValue=None*)

Bases: *epivizfileserver.measurements.measurementClass.Measurement*

Class representing a web server measurement

In addition to params from the base measurement class, source is now server API endpoint

get_data(*chr*, *start*, *end*, *bin=False*, *requestId=124*)

Get data for a genomic region from the API

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **bin** (*bool*) – True to bin the results, defaults to False

Returns a dataframe with results

epivizfileserver.measurements.measurementManager module

class epivizfileserver.measurements.measurementManager.**MeasurementManager**

Bases: object

Measurement manager class

measurements

list of all measurements managed by the system

add_computed_measurement(*mtype*, *mid*, *name*, *measurements*, *computeFunc*, *annotation=None*, *metadata=None*, *computeAxis=1*)

Add a Computed Measurement

Parameters

- **mtype** – measurement type, defaults to 'computed'
- **mid** – measurement id
- **name** – name for this measurement
- **measurements** – list of measurement to use
- **computeFunc** – *NumPy* function to apply

Returns a *ComputedMeasurement* object

add_genome(*genome*, *fileHandler=None*, *url='http://obj.umiacs.umd.edu/genomes/'*)

Add a genome to the list of measurements. The genome has to be tabix indexed for the file server to make remote queries. Our tabix indexed files are available at https://obj.umiacs.umd.edu/genomes/index.html

Parameters

- **genome** – for example : hg19
- **url** – url to the genome file

get_measurements()
Get all available measurements

import_ahub(*ahub*, *handler=None*)
Import measurements from annotationHub objects.

Parameters

- **ahub** – list of file records from annotationHub
- **handler** – an optional filehandler to use

import_dbm(*dbConn*)
Import measurements from a database.The database needs to have a *measurements_index* table with information of files imported into the database.

Parameters **dbConn** – a database connection

import_files(*fileSource*, *fileHandler=None*)
Import measurements from a file.

Parameters

- **fileSource** – location of the configuration file to load
- **fileHandler** – an optional filehandler to use

import_trackhub(*hub*, *handler=None*)
Import measurements from annotationHub objects.

Parameters

- **ahub** – list of file records from annotationHub
- **handler** – an optional filehandler to use

Module contents

epivizfileserver.parser package

Submodules

epivizfileserver.parser.BamFile module

class epivizfileserver.parser.BamFile.**BamFile**(*file*, *columns=None*)
Bases: *epivizfileserver.parser.SamFile.SamFile*

Bam File Class to parse bam files

Parameters

- **file** (*str*) – file location can be local (full path) or hosted publicly
- **columns** (*[str]*) – column names for various columns in file

file
a pysam file object

fileSrc
location of the file

cacheData
cache of accessed data in memory

columns
column names to use

getRange(*chr*, *start*, *end*, *bins=2000*, *zoomlvl=-1*, *metric='AVG'*, *respType='DataFrame'*)
Get data for a given genomic location

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **respType** (*str*) – result format type, default is "DataFrame

Returns

result a DataFrame with matched regions from the input genomic location if respType is DataFrame else result is an array

error if there was any error during the process

get_bin(*x*)

get_col_names(*result*)

to_DF(*result*)

to_msgpack(*result*)

epivizfileserver.parser.BaseFile module

Genomics file classes

class epivizfileserver.parser.BaseFile.**BaseFile**(*file*)
Bases: object

Base file class for parser module

This class provides various useful functions

Parameters **file** – file location

local
if file is local or hosted on a public server

endian
check for endianess

HEADER_STRUCT = <Struct object>

SUMMARY_STRUCT = <Struct object>

bin_rows(*data*, *chr*, *start*, *end*, *columns=None*, *metadata=None*, *bins=400*)
Bin genome by bin length and summarize the bin

decompress_binary(*bin_block*)
decompress a binary string

Parameters **bin_block** – binary string

Returns a zlib decompressed binary string

formatAsJSON(*data*)
Encode a data object as JSON

Parameters **data** – any data object to encode

Returns data encoded as JSON

get_bytes(*offset*, *size*)
Get bytes within a given range

Parameters

- **offset** (*int*) – byte start position in file
- **size** (*int*) – size of bytes to access from offset

Returns binary string from offset to (offset + size)

get_data(*chr*, *start*, *end*)

is_local(*file*)
Checks if file is local or hosted publicly

Parameters **file** – location of file

parse_header()

epivizfileserver.parser.BigBed module

class epivizfileserver.parser.BigBed.**BigBed**(*file*, *columns=None*)
Bases: *epivizfileserver.parser.BigWig.BigWig*

Bed file parser

Parameters **file** (*str*) – bigbed file location

getZoom(*zoomlvl*, *binSize*)
Get Zoom record for the given bin size

Parameters

- **zoomlvl** (*int*) – zoomlvl to get

- **binSize** (*int*) – bin data by bin size

Returns zoom level

get_autosql()
parse autosql stored in file

Returns an array of columns in file parsed from autosql

magic = '0x8789F2EB'

parseLeafDataNode(*chrmId*, *start*, *end*, *zoomlvl*, *rStartChromIx*, *rStartBase*, *rEndChromIx*, *rEndBase*, *rdataOffset*, *rDataSize*)
Parse leaf node

epivizfileserver.parser.BigWig module

class epivizfileserver.parser.BigWig.**BigWig**(*file*, *columns=None*)
Bases: *epivizfileserver.parser.BaseFile.BaseFile*

BigWig file parser

Parameters **file** (*str*) – bigwig file location

tree
chromosome tree parsed from file

columns
column names

cacheData
locally cached data for this file

daskWrapper(*fileObj*, *chr*, *start*, *end*, *bins=2000*, *zoomlvl=-1*, *metric='AVG'*, *respType='JSON'*)
Dask Wrapper

getHeader()
get header byte region in file

getId(*chrmzone*)
Get mapping of chromosome to id stored in file

Parameters **chrmzone** (*str*) – chromosome

Returns id in file for the given chromosome

getRange(*chr*, *start*, *end*, *bins=2000*, *zoomlvl=-1*, *metric='AVG'*, *respType='DataFrame'*)
Get data for a given genomic location

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **respType** (*str*) – result format type, default is "DataFrame

Returns

result a DataFrame with matched regions from the input genomic location if respType is DataFrame else result is an array

error if there was any error during the process

getTree(*zoomlvl*)
Get chromosome tree for a given zoom level

Parameters **zoomlvl** (*int*) – zoomlvl to get

Returns Tree binary bytes

getTreeBytes(*zoomlvl*, *start*, *size*)

getValues(*chr*, *start*, *end*, *zoomlvl*)
Get data for a region

Note: Do not use this directly, use getRange

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end

Returns data for the region

getZoom(*zoomlvl*, *binSize*)
Get Zoom record for the given bin size

Parameters

- **zoomlvl** (*int*) – zoomlvl to get
- **binSize** (*int*) – bin data by bin size

Returns zoom level

getZoomHeader()

get_autosql()
parse autosql in file

Returns an array of columns in file parsed from autosql

get_cache()

locateTree(*chrmId*, *start*, *end*, *zoomlvl*, *offset*)
Locate tree for the given region

Parameters

- **chrmId** (*int*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **zoomlvl** (*int*) – zoom level
- **offset** (*int*) – offset position in the file

Returns nodes in the stored R-tree

magic = '0x888FFC26'

parseLeafDataNode(*chrmId*, *start*, *end*, *zoomlvl*, *rStartChromIx*, *rStartBase*, *rEndChromIx*, *rEndBase*, *rdataOffset*, *rDataSize*)
Parse an Rtree leaf node

parse_header()
parse header in file

Returns attributed stored in the header

readRtreeHeaderNode(*zoomlvl*)
Parse an Rtree Header node

Parameters **zoomlvl** (*int*) – zoom level

Returns header node Rtree object

readRtreeNode(*zoomlvl*, *offset*)
Parse an Rtree node

Parameters

- **zoomlvl** (*int*) – zoom level
- **offset** (*int*) – offset in the file

Returns node Rtree object

set_cache(*cache*)

traverseRtreeNodes(*node*, *zoomlvl*, *chrmId*, *start*, *end*, *result=[]*)
Traverse an Rtree to get nodes in the given range

epivizfileserver.parser.GtfFile module

class epivizfileserver.parser.GtfFile.**GtfFile**(*file*, *columns=None*)
Bases: *epivizfileserver.parser.SamFile.SamFile*

GTF File Class to parse gtf/gff files

Parameters

- **file** (*str*) – file location can be local (full path) or hosted publicly
- **columns** (*[str]*) – column names for various columns in file

file
a pysam file object

fileSrc
location of the file

cacheData
cache of accessed data in memory

columns
column names to use

getRange(*chr*, *start*, *end*, *bins=2000*, *zoomlvl=-1*, *metric='AVG'*, *respType='DataFrame'*, *ensembl=True*)
Get data for a given genomic location

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **respType** (*str*) – result format type, default is "DataFrame

Returns

result a DataFrame with matched regions from the input genomic location if respType is DataFrame else result is an array

error if there was any error during the process

get_bin(*x*)

get_col_names(*result*)

toDF(*result*)

epivizfileserver.parser.HDF5File module

class epivizfileserver.parser.HDF5File.**HDF5File**(*file*)
Bases: object

HDF5 File Class to parse only local hdf5 files

Parameters

- **file** (*str*) – file location can be local (full path) or hosted publicly
- **columns** (*[str]*) – column names for various columns in file

file
a pysam file object

fileSrc
location of the file

cacheData
cache of accessed data in memory

columns
column names to use

getRange(*chr*, *start=None*, *end=None*, *row_names=None*)
Get data for a given genomic location

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **respType** (*str*) – result format type, default is "DataFrame

Returns

result a DataFrame with matched regions from the input genomic location if respType is DataFrame else result is an array

error if there was any error during the process

read_10x_hdf5(*chr*, *query_names*)
read a 10xGenomics hdf5 file

Parameters

- **chr** (*str*) – chromosome
- **query_names** (*[str]*) – genes to filter

Returns

result a DataFrame with matched regions from the input genomic location if respType is DataFrame else result is an array

error if there was any error during the process

epivizfileserver.parser.Helper module

epivizfileserver.parser.Helper.**get_range_helper**(*toDF*, *get_bin*, *get_col_names*, *chr*, *start*, *end*, *file_iter*, *columns*, *respType*)

epivizfileserver.parser.SamFile module

class epivizfileserver.parser.SamFile.**SamFile**(*file*, *columns=None*)

Bases: object

SAM File Class to parse sam files

Parameters

- **file** (*str*) – file location can be local (full path) or hosted publicly
- **columns** (*[str]*) – column names for various columns in file

file
a pysam file object

fileSrc
location of the file

cacheData
cache of accessed data in memory

columns
column names to use

getRange(*chr*, *start*, *end*, *bins=2000*, *zoomlvl=-1*, *metric='AVG'*, *respType='DataFrame'*)
Get data for a given genomic location

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **respType** (*str*) – result format type, default is "DataFrame

Returns

result a DataFrame with matched regions from the input genomic location if respType is DataFrame else result is an array

error if there was any error during the process

get_bin(*x*)

get_cache()

get_col_names(*result*)

set_cache(*cache*)

toDF (*result*)

epivizfileserver.parser.TbxFile module

class epivizfileserver.parser.TbxFile.**TbxFile** (*file*, *columns=None*)
Bases: *epivizfileserver.parser.SamFile.SamFile*

TBX File Class to parse tbx files

Parameters

- **file** (*str*) – file location can be local (full path) or hosted publicly
- **columns** (*[str]*) – column names for various columns in file

file
a pysam file object

fileSrc
location of the file

cacheData
cache of accessed data in memory

columns
column names to use

getRange (*chr*, *start*, *end*, *bins=2000*, *zoomlvl=-1*, *metric='AVG'*, *respType='DataFrame'*)
Get data for a given genomic location

Parameters

- **chr** (*str*) – chromosome
- **start** (*int*) – genomic start
- **end** (*int*) – genomic end
- **respType** (*str*) – result format type, default is "DataFrame

Returns

result a DataFrame with matched regions from the input genomic location if respType is DataFrame else result is an array

error if there was any error during the process

get_bin (*x*)

get_col_names (*result*)

toDF (*result*)

epivizfileserver.parser.TsvFile module

epivizfileserver.parser.utils module

epivizfileserver.parser.utils.**create_parser_object** (*format*, *source*, *columns=None*)
Create appropriate File class based on file format

Parameters

- **format** (*str*) – format of file

- **source** (*str*) – location of file

Returns An instance of parser class

epivizfileserver.parser.utils.**toDataFrame**(*records*, *header=None*)

epivizfileserver.parser.utils.**toMsgpack**(*msg*)

Module contents

epivizfileserver.server package

Submodules

epivizfileserver.server.request module

class epivizfileserver.server.request.**DataRequest**(*request*)
Bases: *epivizfileserver.server.request.EpivizRequest*

Data requests class

get_data(*mMgr*)
Get Data for this request type

Returns JSON response for this request error: HTTP ERROR CODE

Return type result

validate_params(*request*)
Validate parameters for requests

Parameters **request** – dict of params from request

class epivizfileserver.server.request.**EpivizRequest**(*request*)
Bases: object

Base class to process requests

get_data(*mMgr*)
Get Data for this request type

Returns JSON response for this request error: HTTP ERROR CODE

Return type result

validate_params(*request*)
Validate parameters for requests

Parameters **request** – dict of params from request

class epivizfileserver.server.request.**MeasurementRequest**(*request*)
Bases: *epivizfileserver.server.request.EpivizRequest*

Measurement requests class

get_data(*mMgr*)
Get Data for this request type

Returns JSON response for this request error: HTTP ERROR CODE

Return type result

validate_params(*request*)
Validate parameters for requests

Parameters **request** – dict of params from request

class epivizfileserver.server.request.**SearchRequest**(*request*)
Bases: *epivizfileserver.server.request.EpivizRequest*

Search requests class

get_data(*mMgr*)
Get Data for this request type

Returns JSON response for this request error: HTTP ERROR CODE

Return type result

validate_params(*request*)
Validate parameters for requests

Parameters **request** – dict of params from request

class epivizfileserver.server.request.**SeqInfoRequest**(*request*)
Bases: *epivizfileserver.server.request.EpivizRequest*

SeqInfo requests class

get_data(*mMgr*)
Get Data for this request type

Returns JSON response for this request error: HTTP ERROR CODE

Return type result

validate_params(*request*)
Validate parameters for requests

Parameters **request** – dict of params from request

epivizfileserver.server.request.**create_request**(*action*, *request*)
Create appropriate request class based on action

Parameters

- **action** – Type of request
- **request** – Other request parameters

Returns An instance of EpivizRequest class

epivizfileserver.server.utils module

epivizfileserver.server.utils.**bin_rows**(*input*, *max_rows=2000*)
Helper function to bin rows to resolution

Parameters

- **input** – dataframe to bin
- **max_rows** – resolution to scale rows

Returns data frame with scaled rows

epivizfileserver.server.utils.**create_parser_object**(*format*, *source*)
Create appropriate File class based on file format

Parameters

- **format** – Type of file
- **request** – Other request parameters

Returns An instance of parser class

epivizfileserver.server.utils.**format_result**(*input*, *params*, *offset=True*)
Fromat result to a epiviz compatible format

Parameters

- **input** – input dataframe
- **params** – request parameters
- **offset** – defaults to True

Returns formatted JSON response

Module contents

epivizfileserver.server.**MAXWORKER = 10**
The server module allows users to instantly create a REST API from the list of measuremensts. The API can then be used to interactive exploration of data or build various applications.

epivizfileserver.server.**clean_up**(*app*, *loop*)

epivizfileserver.server.**create_fileHandler**()
create a dask file handler if one doesn't exist

epivizfileserver.server.**schedulePickle**()
Sanic task to regularly pickle file objects from memory

epivizfileserver.server.**setup_app**(*measurementsManager*)
Setup the Sanic Rest API

Parameters **measurementsManager** – a measurements manager object

Returns a sanic app object

epivizfileserver.server.**setup_connection**(*app*, *loop*)
Sanic callback for app setup before the server starts

epivizfileserver.trackhub package

Submodules

epivizfileserver.trackhub.TrackHub module

class epivizfileserver.trackhub.TrackHub.**TrackHub**(*file*)
Bases: object

Base class for managing trackhub files TrackHub documentation is available at https://genome.ucsc.edu/goldenPath/help/hgTrackHubHelp.html

Parameters **file** – location of trackhub directory

parse_genome()

parse_genomeTracks()

parse_hub()

parse_trackDb(*track_loc*)

Module contents

Submodules

epivizfileserver.formated_test module

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

D

E

F

G

H

I

L

M

P

Q