BaseSpacePy Documentation

Release 0.1

Morten Kallberg

August 23, 2012

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 Introduction

BaseSpacePy is a Python based SDK to be used in the development of Apps and scripts for working with Illumina's BaseSpace cloud-computing solution for next-gen sequencing data analysis. The primary purpose of the SDK is to provide an easy-to-use Python environment enabling developers to authenticate a user, retrieve data, and upload data/results from their own analysis to BaseSpace.

If you haven't already done so, you may wish to familiarize yourself with the BaseSpace developer documentation (https://developer.basespace.illumina.com/) prior to working through the example scripts below.

Note: It will be necessary to have created a BaseSpace account with a new App and have the client_key and client_secret codes for the App available to run a number of the following examples.

1.1.1 Availability

Version 0.1 of BaseSpacePy can be checked out here:

```
svn checkout svn://ilmnhw-biolinsd.hwus.illumina.com:/data/tmann/svn/svnrepos/BaseSpacePy_v0.1
```

1.1.2 Setup

Requirements: Python 2.6 with the packages 'urllib2', 'pycurl', and 'shutil' available.

To install 'BaseSpacePy' run the 'setup.py' script in the src directory (for a global install you will need to run this command with root privileges):

```
cd BaseSpacePy_v0.1/src
python setup.py install
```

If you do not have root access, you may use the --prefix option to specify the install directory (make sure this directory is in you PYTHONPATH):

```
python setup.py install --prefix=/folder/in/my/pythonpath
```

For more install options type:

```
python setup.py --help
```

Altenatively you may include the src directory in your PYTHONPATH by doing the following export:

```
export PYTHONPATH=$PYTHONPATH:/my/path/BaseSpacePy_vx.x/src
```

or add it to the PYTHONPATH at the top of your Python scripts using BaseSpacePy:

```
import sys
sys.path.append('/my/path/BaseSpacePy_vx.x/src')
import BaseSpacePy
```

To test that everything is working as expected, launch a Python prompt and try importing 'BaseSpacePy':

```
mkallberg@ubuntu:~/$ python
>>> import BaseSpacePy
```

1.2 Application triggering

This section demonstrates how to retrieve the `AppLaunch` object produced when a user triggers a BaseSpace App. Further, we cover how to automatically generate the scope strings to request access to the data object (be it a project, a sample, or an analysis) that the App was triggered to analyze.

The initial http request to our App from BaseSpace is identified by an `ApplicationActionId`, using this piece of information we are able to obtain information about the user who launched the App and the data that is sought analyzed by the App. First, we instantiate a BaseSpaceAuth object using the `client_key` and `client_secret` codes provided on the BaseSpace developers website when registering our App:

```
from BaseSpacePy.api.BaseSpaceAuth import BaseSpaceAuth

# initialize an authentication object using the key and secret from your app
# Fill in with your own values
client_key                  = <my key>
client_secret               = <my secret>
ApplicationActionId         = <my action id>
BaseSpaceUrl                = 'https://api.cloud-endor.illumina.com/'
version                     = 'v1pre2/'

# First we will initialize a BaseSpace authentication object
BSauth = BaseSpaceAuth(client_key,client_secret,BaseSpaceUrl,version)

# By supplying the application trigger id we can get out an AppLaunch object
triggerObj = BSauth.getAppTrigger(ApplicationActionId)
print str(triggerObj)

Output[]:

https://api.cloud-endor.illumina.com/v1pre2/applicationactions/<my action id>
```

We can get the type of object the app was triggered on from the `getLaunchType`-method in the `BaseSpaceAuth` instance:

```
# The trigger type is a list with two items, the first a string taking the one of the values ('Proje
# and the second a list of the objects of that type
triggerType = triggerObj.getLaunchType()
print "\nType of data the app was triggered on"
print triggerType
print "\nWe can get a handle for the user who triggered the app\n" + str(triggerObj.User)
```

```
Output[]:

Type of data the app was triggered on
['Projects', [YourProject]]

We can get a handle for the user who triggered the app
152152: Morten Kallberg
```

To start working, we will want to expand our permission scope for the trigger object so we can read and write data. The details of this process is the subject of the next section. We end this section by demonstrating how one can easily obtain the so-called "scope string," used when requesting further access, from the trigger object:

```
triggerObj = triggerType[1][-1]
print "\nThe scope string for requesting write access to the trigger object is:"
print triggerObj.getAccessStr(scope='write')

Output[]:

The scope string for requesting write access to the trigger object is:
write project 89
```

1.3 Requesting an access-token for data browsing

Here we demonstrate the basic BaseSpace authentication process. The work-flow outlined here is

1. Request of access to a specific data-scope
2. User approval of access request
3. Browsing data

Note: It will be useful if you are logged in to the BaseSpace web-site before launching this example to make the access grant procedure faster.

Again we will start out by initializing a `BaseSpaceAuth` object:

```
from BaseSpacePy.api.BaseSpaceAuth import BaseSpaceAuth
import time

# initialize an authentication object using the key and secret from your app
client_key                = <my key>
client_secret             = <my secret>
BaseSpaceUrl              = 'https://api.cloud-endor.illumina.com/'
version                   = 'v1pre2/'
BSauth = BaseSpaceAuth(client_key,client_secret,BaseSpaceUrl,version)
```

First get verification code and uri for scope 'browse global'

```
deviceInfo = BSauth.getVerificationCode('browse global')
```

At this point the user must visit the verification uri to grant us access

```
## PAUSE HERE
# Have the user visit the verification uri to grant us access
print "Please visit the uri within 30 seconds and grant access"
print deviceInfo['verification_with_code_uri']
```

```
time.sleep(30)
## PAUSE HERE

Output[]:

Please visit the uri within 10 seconds and grant access
https://cloud-endor.illumina.com/oauth/device?code=<my device code>
```

There are two options for obtaining the access-token and instantiating a `BaseSpaceAPI` object:

```
# Get the access-token directly and instantiate an api yourself
#token = BSauth.getAccessToken(deviceInfo['device_code'])
#print "My token " + str(token)

# Alternatively we can generate an access-token and request a BaseSpaceApi instance
# with the newly generated token in one step
myAPI = BSauth.getBaseSpaceApi(deviceInfo['device_code'])
print myAPI

Output[]:

BaseSpaceAPI instance - using token=<my access token>
```

At this point we can start using the `BaseSpaceAPI` instance to browse the available data for the current user, the details of this process is the subject of the next section. Here we will end with showing how the API object can be used to list all BaseSpace genome instances:

```
# We will get all available genomes with our new api!
allGenomes  = myAPI.getAvailableGenomes()
print "\nGenomes \n" + str(allGenomes)

Output[]:

Genomes
[Arabidopsis thaliana, Bos Taurus, Escherichia coli, Homo sapiens, Mus musculus, Phix,\
 Rhodobacter sphaeroides, Rattus norvegicus, Saccharomyces cerevisiae, Staphylococcus aureus, Bacillu
```

1.4 Browsing data with `global browse`-access

This section demonstrates basic browsing of BaseSpace objects once an access-token for global browsing has been obtained. We will see how objects can be retrieved using either the `BaseSpaceAPI` class or by use of method calls on related object instances (for example once a `user` instance we can use it to retrieve all project belonging to that user).

First we will initialize a `BaseSpaceAPI` using our access-token for `global browse`:

```
from BaseSpacePy.api.BaseSpaceAPI import BaseSpaceAPI

# REST server information and user access token
server          = 'https://api.cloud-endor.illumina.com/'
version         = 'v1pre2'
access_token    = <my access token>

# First, create a client for making calls for this user session
myAPI   = BaseSpaceAPI(AccessToken=access_token,apiServer= server + version)
```

First we will try to retrieve a genome object:

```
# Now grab the genome with id=4
myGenome     = myAPI.getGenomeById('4')
print "\nThe Genome is " + str(myGenome)
print "We can get more information from the genome object"
print 'Id: ' + myGenome.Id
print 'Href: ' + myGenome.Href
print 'DisplayName: ' + myGenome.DisplayName
```

```
Output[]:

The Genome is Homo sapiens
We can get more information from the genome object
Id: 4
Href: v1pre2/genomes/4
DisplayName: Homo Sapiens - UCSC (hg19)
```

Using a comparable method we can get a list of all available genomes:

```
# Get a list of all genomes
allGenomes  = myAPI.getAvailableGenomes()
print "\nGenomes \n" + str(allGenomes)
```

```
Output[]:

Genomes
[Arabidopsis thaliana, Bos Taurus, Escherichia coli, Homo sapiens, Mus musculus, Phix,\
 Rhodobacter sphaeroides, Rattus norvegicus, Saccharomyces cerevisiae, Staphylococcus aureus, Bacillu
```

Now, let us retrieve the `User` objects for the current user, and list all projects for this user:

```
# Take a look at the current user
user         = myAPI.getUserById('current')
print "\nThe current user is \n" + str(user)

# Now list the projects for this user
myProjects    = myAPI.getProjectByUser('current')
print "\nThe projects for this user are \n" + str(myProjects)
```

```
Output[]:

The current user is
152152: Morten Kallberg

The projects for this user are
[HiSeq 2500, Bolt, YourProject, 2X151 Rhodobacter Resequencing, EColi resequencing]
```

We can also achieve this by making a call using the `user` instance. Notice that these calls take an instance of `BaseSpaceAPI` with apporpriate priviliges to complete the transaction as parameter, this true for all retrieval method calls made on data objects:

```
myProjects2 = user.getProjects(myAPI)
print "\nProjects retrieved from the user instance \n" + str(myProjects2)

# List the runs available for the current user
runs = user.getRuns(myAPI)
print "\nThe runs for this user are \n" + str(runs)
```

```
Output[]:

Projects retrieved from the user instance
[HiSeq 2500, Bolt, YourProject, 2X151 Rhodobacter Resequencing, EColi resequencing]

The runs for this user are
[2X151 Rhodobacter Resequencing, 2x26 Validation Human 4-Plex, EColi resequencing]
```

In the same manner we can get a list of accessible user runs:

```
runs = user.getRuns(myAPI)
print "\nRuns retrieved from user instance \n" + str(runs)

Output[]:

Runs retrieved from user instance
[2X151 Rhodobacter Resequencing, 2x26 Validation Human 4-Plex, EColi resequencing]
```

1.5 Accessing file-trees and querying BAM/VCF files

In this section we demonstrate how to access samples and analysis from a projects and how to work with the available file data for such instances. In addition, we take a look at some of the special queuring methods associated with BAM- and VCF-files.

Again, start out by initializing a `BaseSpacePy` instance and retrieving all projects belonging to the current user:

```
# First, create a client for making calls for this user session
myAPI    = BaseSpaceAPI(AccessToken=access_token,apiServer= server + version)
user         = myAPI.getUserById('current')
myProjects    = myAPI.getProjectByUser('current')
```

Now we can list all the analyses and samples for these projects

```
for singleProject in myProjects:
    print "# " + str(singleProject)
    analyses = singleProject.getAnalyses(myAPI)
    print "    The analysis for project " + str(singleProject) + " are \n\t" + str(analyses)
    samples = singleProject.getSamples(myAPI)
    print "    The samples for project " + str(singleProject) + " are \n\t" + str(samples)

Output[]:

# HiSeq 2500
    The analysis for project HiSeq 2500 are
        [Resequencing - Completed]
    The samples for project HiSeq 2500 are
        [NA18507]
# Bolt
    The analysis for project Bolt are
        [Amplicon - Completed, Amplicon - Completed, Amplicon ...
    The samples for project Bolt are
        [sample_1, sample_2, sample_3, ...

......
```

We'll take a further look at the files belonging to the sample from the last project in the loop above:

```
for s in samples:
    print "Sample " + str(s)
    ff = s.getFiles(myAPI)
    print ff

Output[]:

Sample Ecoli
[s_G1_L001_R1_001.fastq.1.gz, s_G1_L001_R1_002.fastq.1.gz, s_G1_L001_R2_001.fastq.1.gz, s_G1_L001_R2_
```

Now, have a look at some of the methods calls specific to `Bam` and `VCF` files. First, we will get a `Bam`-file and then retrieve the coverage information available for chromosome 2 between positions 1 and 20000:

```
# Now do some work with files
# we'll grab a BAM by id and get the coverage for an interval + accompanying meta-data
myBam = myAPI.getFileById('2150156')
print myBam
cov     = myBam.getIntervalCoverage(myAPI,'chr2','1','20000')
print cov
covMeta = myBam.getCoverageMeta(myAPI,'chr2')
print covMeta

Output[]:

sorted.bam
Chrchr2: 1-20096: BucketSize=16
CoverageMeta: max=20483 gran=128
```

For `VCF`-files we can filter variant calls based on chromosome and location as well:

```
# and a vcf file
myVCF = myAPI.getFileById('2150158')
# Get the variant meta info
varMeta = myVCF.getVariantMeta(myAPI)
print varMeta
var     = myVCF.filterVariant(myAPI,'2','1', '11000')
print var

Output[]:

VariantHeader: SampleCount=1
[Variant - chr2: 10236 id=['.'], Variant - chr2: 10249 id=['.'], ....
```

1.6 Creating an analysis and uploading results

In this section we will see how to create a new analysis object, change its state and upload result files to it as well as retrieve files from it.

First, create a client for making calls for this user session:

```
myBaseSpaceAPI    = BaseSpaceAPI(AccessToken=access_token,apiServer= server + version)
#
## Now we'll do some work of our own. First get a project to work on
## we'll need write permission, for the project we are working on
## meaning we will need get a new token and instantiate a new BaseSpaceAPI
p = myBaseSpaceAPI.getProjectById('89')
# A short-cut for getting a scope string if we already have a project-instance:
```

```
print p.getAccessStr(scope='write')
# or simply
p.getAccessStr()

Output[]:

write project 89
```

Assuming we now have write access for the project, we will list the current analyses for the project:

```
ana = p.getAnalyses(myBaseSpaceAPI)
print "\nThe current analyses are \n" + str(ana)

Output[]:

The current analyses are
[Results for sample 123 - Working, Results for sample 124 - Working...
```

To create an analysis for a project, simply give the name and description to the `createAnalysis`-method:

```
analysis = p.createAnalysis(myBaseSpaceAPI,"My very first analysis!!","This is my analysis")
print "\nSome info about our new analysis"
print analysis
print analysis.Id
print analysis.Status
# we can change the status of out analysis and add a status-summary as follows
analysis.setStatus(myBaseSpaceAPI,'completed',"We worked hard.")
print "\nAfter a change of status we get\n" + str(analysis)

### List the analyses again and see if our new object shows up
ana = p.getAnalyses(myBaseSpaceAPI)
print "\nThe updated analyses are \n" + str(ana)

Output[]:

Some info about our new analysis
My very first analysis!! - Working
94094
Working

After a change of status we get
My very first analysis!! - Completed

The updated analyses are
[Results for sample 123 - Working, Results for sample 124 - Working, Results for sample 124 - Workin
```

Now we will make another analysis and try to upload some files to it:

```
analysis2 = p.createAnalysis(myBaseSpaceAPI,"My second analysis","This one I will upload to")
analysis2.uploadFile(myBaseSpaceAPI, '/my/file/dir/testFile2.txt', 'BaseSpaceTestFile.txt', '/mydir/'
print "\nMy analysis number 2 \n" + str(analysis2)
#
## Check to see if our new file made it
analysisFiles = analysis2.getFiles(myBaseSpaceAPI)
print "\nThese are the files in the analysis"
print analysisFiles
f = analysisFiles[-1]
```

```
Output[]:

My analysis number 2
My second analysis - Working

These are the files in the analysis
[BaseSpaceTestFile.txt]
```

We can even download our newly uploaded file in the following manner:

```
f.downloadFile(myBaseSpaceAPI,'/path/to/place/file/in/')
```

CHAPTER

TWO

AVAILABLE MODULES

2.1 API

class BaseSpacePy.api.BaseSpaceAPI.**BaseSpaceAPI**(*AccessToken*, *apiServer*)
The main API class used for all communication with with the REST server

analysisFileUpload(*Id*, *localPath*, *fileName*, *directory*, *contentType*, *multipart=0*)
Uploads a file associated with an analysis to BaseSpace and returns the corresponding file object

Parameters

- **Id** – Analysis id.
- **localPath** – The local path to the file to be uploaded.
- **fileName** – The desired filename in the Analysis folder on the BaseSpace server.
- **directory** – The directory the file should be placed in.
- **contentType** – The content-type of the file.

createAnalyses(*Id*, *name*, *desc*)
Create an analysis object

Parameters

- **Id** – The id for the project in which the analysis is to be added
- **name** – The name of the analysis
- **desc** – A describtion of the analysis

fileDownload(*Id*, *localDir*, *name*, *range=[]*)
Downloads a BaseSpace file to a local directory

Parameters

- **Id** – The file id
- **localDir** – The local directory to place the file in
- **name** – The name of the local file
- **range** – (Optional) The byte range of the file to retrieve (not yet implemented)

filterVariantSet(*Id*, *Chrom*, *StartPos*, *EndPos*, *Format*, *queryPars={'Limit': '100', 'SortBy': 'Position', 'SortDir': 'Asc', 'Offset': '0'}*)
List the variants in a set of variants. Maximum returned records is 1000

Parameters

- **Id** – The id of the variant file
- **Chrom** – The chromosome of interest
- **StartPos** – The start position of the sequence of interest
- **EndPos** – The start position of the sequence of interest
- **Format** – Set to 'vcf' to get the results as lines in VCF format
- **queryPars** – An (optional) object of type QueryParameters for custom sorting and filtering

getAccessToken()
Returns the access-token that was used to initialize the BaseSpaceAPI object.

getAccessibleRunsByUser(*Id*, *queryPars={'Limit': '100', 'SortBy': 'Id', 'SortDir': 'Asc', 'Offset': '0'}*)
Returns a list of accessible runs for the User with id=Id

Parameters

- **Id** – An user id
- **queryPars** – An (optional) object of type QueryParameters for custom sorting and filtering

getAnalysisById(*Id*)
Returns an Analysis object corresponding to Id

Parameters **Id** – The Id of the Analysis

getAnalysisByProject(*Id*, *queryPars={'Limit': '100', 'SortBy': 'Id', 'SortDir': 'Asc', 'Offset': '0'}*)
Returns a list of Analysis object associated with the project with Id

Parameters

- **Id** – The project id
- **queryPars** – An (optional) object of type QueryParameters for custom sorting and filtering

getAnalysisFiles(*Id*, *queryPars={'Limit': '100', 'SortBy': 'Id', 'SortDir': 'Asc', 'Offset': '0'}*)
Returns a list of File object for the Analysis with id = Id

Parameters

- **Id** – The id of the analysis.
- **queryPars** – An (optional) object of type QueryParameters for custom sorting and filtering

getAvailableGenomes(*queryPars={'Limit': '100', 'SortBy': 'Id', 'SortDir': 'Asc', 'Offset': '0'}*)
Returns a list of all available genomes

Parameters **queryPars** – An (optional) object of type QueryParameters for custom sorting and filtering

getCoverageMetaInfo(*Id*, *Chrom*)
Returns Metadata about coverage as a CoverageMetadata instance

Parameters

- **Id** – he Id of the Bam file
- **Chrom** – Chromosome to query

getFileById(*Id*)
Returns a file object by Id

Parameters **Id** – The id of the file

getFilesBySample(*Id*, *queryPars={'Limit': '100', 'SortBy': 'Id', 'SortDir': 'Asc', 'Offset': '0'}*)
Returns a list of File objects associated with sample with Id

Parameters

- **Id** – A Sample id
- **queryPars** – An (optional) object of type QueryParameters for custom sorting and filtering

getGenomeById(*Id*)
Returns an instance of Genome with the specified Id

Parameters **Id** – The genome id

getIntervalCoverage(*Id*, *Chrom*, *StartPos=None*, *EndPos=None*)
Mean coverage levels over a sequence interval

Parameters

- **Id** – Chromosome to query
- **Chrom** – The Id of the resource
- **StartPos** – Get coverage starting at this position. Default is 1
- **EndPos** – Get coverage up to and including this position. Default is StartPos + 1280

:return:CoverageResponse – an instance of CoverageResponse

getProjectById(*Id*)
Request a project object by Id

Parameters **Id** – The Id of the project

getProjectByUser(*Id*, *queryPars={'Limit': '100', 'SortBy': 'Id', 'SortDir': 'Asc', 'Offset': '0'}*)
Returns a list available projects for a User with the specified Id

Parameters

- **Id** – The id of the user
- **qp** – An (optional) object of type QueryParameters for custom sorting and filtering

getSampleById(*Id*)
Returns a Sample object

Parameters **Id** – The id of the sample

getSamplesByProject(*Id*, *queryPars={'Limit': '100', 'SortBy': 'Id', 'SortDir': 'Asc', 'Offset': '0'}*)
Returns a list of samples associated with a project with Id

Parameters

- **Id** – The id of the project
- **queryPars** – An (optional) object of type QueryParameters for custom sorting and filtering

getServerUri()
Returns the server uri used by this instance

getUserById(*Id*)
Returns the User object corresponding to Id

Parameters **Id** – The Id of the user

getVariantMetadata(*Id*, *Format*)
Returns a VariantMetadata object for the variant file

Parameters

- **Id** – The Id of the VCF file
- **Format** – Set to 'vcf' to get the results as lines in VCF format

markAnalysisState(*Id*, *Status*, *Summary*)
Set the status of an Analysis object

Parameters

- **Id** – The id of the analysis
- **Status** – The status assignment string must
- **Summary** – The summary string

multipartFileUpload(*Id*, *localPath*, *fileName*, *directory*, *contentType*, *tempdir=''*, *cpuCount=2*, *partSize=25*, *verbose=0*)
Method for multi-threaded file-upload for parallel transfer of very large files (currently only runs on unix systems)

Parameters

- **Id** – The analysis ID
- **localPath** – The local path of the file to be uploaded
- **fileName** – The desired filename on the server
- **directory** – The server directory to place the file in (empty string will place it in the root directory)
- **contentType** – The content type of the file
- **tempdir** – Temp directory to use, if blank the directory for 'localPath' will be used
- **cpuCount** – The number of CPUs to be used
- **partSize** – The size of individual upload parts (must be between 5 and 25mb)
- **verbose** – Write process output to stdout as upload progresses

2.2 Models

2.2.1 AppLaunch

class BaseSpacePy.model.AppLaunch.**AppLaunch**
AppLaunch contains the data returned

getLaunchType()
Returns a list [<launch type name>, list of objects] where <launch type name> is one of Projects, Samples, Analyses

2.2.2 Project

class BaseSpacePy.model.Project.**Project**
Represents a BaseSpace Project object.

createAnalysis(*api*, *name*, *desc*)
Return a newly created Analysis object

Parameters

- **api** – An instance of BaseSpaceAPI
- **name** – The name of the analysis
- **desc** – A describtion of the analysis

getAccessStr(*scope='write'*)
Returns the scope-string to used for requesting BaseSpace access to the object

Parameters **scope** – The scope-type that is request (write|read)

getAnalyses(*api*)
Returns a list of Analysis objects.

Parameters **api** – An instance of BaseSpaceAPI

getSamples(*api*)
Returns a list of Sample objects.

Parameters **api** – An instance of BaseSpaceAPI

isInit()
Is called to test if the Project instance has been initialized.

Throws: ModelNotInitializedException - Indicates the object has not been populated yet.

2.2.3 Analysis

class BaseSpacePy.model.Analysis.**Analysis**

getAccessStr(*scope='write'*)
Returns the scope-string to be used for requesting BaseSpace access to the object

Parameters **scope** – The scope-type that is request (write|read)

getFiles(*api*, *myQp={}*)
Returns a list of file objects

Parameters

- **api** – An instance of BaseSpaceAPI
- **myQp** – (Optional) QueryParameters for sorting and filtering the file list

isInit()
Is called to test if the Project instance has been initialized

Throws: ModelNotInitializedException - if the instance has not been populated.

setStatus(*api*, *Status*, *Summary*)
Sets the status of the analysis (note: once set to 'completed' or 'aborted' no more work can be done to the instance)

Parameters

- **api** – An instance of BaseSpaceAPI
- **Status** – The status value, must be completed, aborted, working, or suspended
- **Summary** – The status summary

uploadFile(*api*, *localPath*, *fileName*, *directory*, *contentType*)
Uploads a local file to the BaseSpace analysis

Parameters

- **api** – An instance of BaseSpaceAPI
- **localPath** – The local path of the file
- **fileName** – The filename
- **directory** – The remote directory to upload to
- **contentType** – The content-type of the file

uploadMultipartFile(*api*, *localPath*, *fileName*, *directory*, *contentType*, *tempDir=''*, *cpuCount=1*, *partSize=10*, *verbose=0*)

Upload a file in multi-part mode. Returns an object of type MultipartUpload used for managing the upload.

:param api:An instance of BaseSpaceAPI :param localPath: The local path of the file :param fileName: The filename :param directory: The remote directory to upload to :param contentType: The content-type of the file :param cpuCount: The number of CPUs to used for the upload :param partSize:

2.2.4 Sample

class BaseSpacePy.model.Sample.**Sample**

Representation of a BaseSpace Sample object.

getAccessStr(*scope='write'*)

Returns the scope-string to used for requesting BaseSpace access to the sample.

Parameters **scope** – The scope type that is request (write|read).

getFiles(*api*, *myQp={}*)

Returns a list of File objects

Parameters

- **api** – A BaseSpaceAPI instance
- **myQp** – Query parameters to sort and filter the file list by.

isInit()

Is called to test if the sample instance has been initialized.

Throws: ModelNotInitializedException - Indicated the Id variable is not set.

2.2.5 File

class BaseSpacePy.model.File.**File**

Represents a BaseSpace file object.

downloadFile(*api*, *localDir*, *range=[]*)

Download the file object to the specified localDir or a byte range of the file, by specifying the start and stop byte in the range.

Parameters

- **api** – A BaseSpaceAPI with read access on the scope including the file object.
- **loadlDir** – The local directory to place the file in.
- **range** – Specify the start and stop byte of the file chunk that needs retrieved.

filterVariant(*api*, *Chrom*, *StartPos*, *EndPos*, *q=None*)

Returns a list of Variant objects available in the specified region

Parameters

- **api** – An instance of BaseSpaceAPI
- **Chrom** – Chromosome as a string - for example 'chr2'
- **StartPos** – The start position of region of interest as a string
- **EndPos** – The end position of region of interest as a string
- **q** – An instance of

getCoverageMeta(*api*, *Chrom*)
Return an object of CoverageMetadata for the selected region

Parameters

- **api** – An instance of BaseSpaceAPI.
- **Chrom** – The chromosome of interest.

getIntervalCoverage(*api*, *Chrom*, *StartPos*, *EndPos*)
Return a coverage object for the specified region and chromosome.

Parameters

- **api** – An instance of BaseSpaceAPI
- **Chrom** – Chromosome as a string - for example 'chr2'
- **StartPos** – The start position of region of interest as a string
- **EndPos** – The end position of region of interest as a string

getVariantMeta(*api*)
Return the the meta info for a VCF file as a VariantInfo object

Parameters **api** – An instance of BaseSpaceAPI

isInit()
Is called to test if the File instance has been initialized.

Throws: ModelNotInitializedException if the instance has not been populated yet.

isValidFileOption(*filetype*)
Is called to test if the File instance is matches the filtype parameter

Parameters **filetype** – The filetype for coverage or variant requests

2.2.6 QueryParameters

class BaseSpacePy.model.QueryParameters.**QueryParameters**(*pars={}, required=['SortBy', 'Offset', 'Limit', 'SortDir']*)
The QueryParameters class can be passed as an optional arguments for a specific sorting of list-responses (such as lists of sample, analysis, or variants)

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

INDEX

I

M

P

Q

S

U