pyepic

Release 0.0.1

Zenotech

Jun 04, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

PyEPIC is a simple wrapper around the EPIC API code. It is designed to allow users to integrate with EPIC for the submission and management of HPC jobs. PyEPIC wraps the autogenerated API code. The autogenerated code can be used directly and is available on PyPi as epiccore .

PyEPIC can be used as a reference for making use of the epiccore.

The REST API documentation can be viewed in epic at https://epic.zenotech.com/api/explorer/ or https://epic.zenotech.com/api/docs/

CHAPTER

TWO

INSTALLATION

Python 3.4+ is required. The package can be installed from PyPi using pip.

```
pip install pyepic
```

To use the API you will need an 'API token'. This can be retieved by logging into EPIC and viewing the API Token section on the Your Profile -> Your Credentials page.

CHAPTER

THREE

EXAMPLES

3.1 General Usage

3.1.1 Setting up the client

To initialise the client simply import EPICClient and then create a client instance, passing in your API Token from EPIC as a parameter.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")
```

You can then access the appropriate client api using the corresponding api member variable.

3.2 Catalog

The catalog API allows you to list the current configurations available for Jobs, Applications and Desktops in EPIC. This can be used to get the IDs necessary for launching a job or desktop with the correct configuration.

3.2.1 Listing Applications

To list the available applications you can use the list_applications() method. The applications returned can be viewed by iterating over the response results.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# List all applications
apps = client.catalog.list_applications()
print("App Code | Application | Version | Cluster Codes")
for app in apps:
    for version in app.versions:
        print("{} | {} | {} | {}".format(version.app_code, app.product.name, version.
↪version, version.available_on))

# List applications but filter for "foam" in the application name
foam_apps = client.catalog.list_applications(product_name="foam")
```

An example of the output of list_applications is shown below. The App code listed in the versions dictionary is the app_code used when creating a job for that application.

```
[
            {'product': {'description': 'The goal of the Extend-Project is to '
                                        'open the FOAM CFD toolbox to '
                                        'community contributed extensions in '
                                        'the spirit of the OpenSource '
                                        'development model.',
                         'image': 'https://s3-eu-west-1.amazonaws.com/epic-media-
↪zenotech/media/products/openfoam-extend.png',
                         'name': 'FOAM Extend',
                         'small_print': ' This offering is not approved or '
                                        'endorsed by ESI Group or '
                                        'ESI-OpenCFD®, the producer of the '
                                        'OpenFOAM® software and owner of the '
                                        'OpenFOAM® trademark.'},
            'versions': [{'id': 6, 'queue_ids': [5], 'version': '3.1'},
                         {'id': 4, 'queue_ids': [5], 'version': '1.6'}]},
            {'id': 3,
            'product': {'description': 'OpenFOAM is free, open source '
                                        'software for computational fluid '
                                        'dynamics (CFD), developed primarily '
                                        'by OpenCFD.',
                         'image': 'https://s3-eu-west-1.amazonaws.com/epic-media-
↪zenotech/media/products/openfoam.png',
                         'name': 'OpenFOAM',
                         'small_print': 'This offering is not approved or '
                                        'endorsed by ESI Group or '
                                        'ESI-OpenCFD®, the producer of the '
                                        'OpenFOAM® software and owner of the '
                                        'OpenFOAM® trademark.'},
            'versions': [{'id': 12, 'queue_ids': [5], 'version': 'v1606+'},
                         {'id': 11, 'queue_ids': [5], 'version': '4.1'},
                         {'id': 10, 'queue_ids': [5], 'version': '3.0.1'},
                         {'id': 9, 'queue_ids': [5], 'version': '2.4.0'},
                         {'id': 8, 'queue_ids': [5], 'version': '2.3.1'}]}
]
```

3.2.2 Listing Queues

To list queues use the list_clusters() method. You can filter by cluster name or by available application version id.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# List all clusters
clusters = client.catalog.list_clusters()
for cluster in clusters:
    print("{} | {} | {}".format(cluster.id, cluster.name, cluster.description))

# List clusters with a filter for a cluster name
clusters = client.catalog.list_clusters(cluster_name="csd3")

# List cluster with a filter for a queue name
```

(continues on next page)

(continued from previous page)

```
clusters = client.catalog.list_clusters(queue_name="gpu")

# List clusters with a filter for a particular application versions, filter using the␣
↪app_codes from the catalog endpoint
clusters = client.catalog.list_clusters(allowed_apps="cfx:16.1")
```

An example json response is shown below. The id listed is the queue_code is used when submitting an EPIC job to that queue.

```
[
           {'display_description': 'The CFMS cluster is built using the Cray '
                                   'CS-400 solution, with parallel file '
                                   'storage provided by ArcaStream, based '
                                   'upon IBM Spectrum Scale (formerly known '
                                   'as IBM GPFS). The cluster includes '
                                   'latest generation Intel E5-26XX v4 '
                                   '(Broadwell) Xeon CPUs. The GPU nodes '
                                   'each have two Nvidia K80 GPUs.',
            'display_name': 'CFMS - GPU',
            'queue_code': 'cfms:gpu',
            'maintenance_mode': False,
            'max_allocation': 8,
            'max_runtime': 72,
            'reported_avail_tasks': None,
            'reported_max_tasks': None,
            'resource_config': "{'cpus': 2, 'cores_per_cpu': 8, "
                               "'threads_per_core': 1, 'accelerator': "
                               "{'name': 'K80 x 2', 'acc_class': 'CUDA', "
                               "'quantity': 2, 'description': '2 x Nvidia "
                               "K80'}, 'memory': "
                               "'64.0'}",
            'sla': {'description': 'The jobs will be scheduled using the '
                                   'clusters standard batch scheduling '
                                   'policy.',
                    'name': 'Standard'}},
           {'display_description': 'The CFMS cluster is built using the Cray '
                                   'CS-400 solution, with parallel file '
                                   'storage provided by ArcaStream, based '
                                   'upon IBM Spectrum Scale (formerly known '
                                   'as IBM GPFS). The cluster includes '
                                   'latest generation Intel E5-26XX v4 '
                                   '(Broadwell) Xeon CPUs. The High Memory '
                                   'nodes each have 256GB of RAM.',
            'display_name': 'CFMS - High Memory',
            'queue_code': 'cfms:highmem',
            'maintenance_mode': False,
            'max_allocation': 20,
            'max_runtime': 72,
            'reported_avail_tasks': None,
            'reported_max_tasks': None,
            'resource_config': "{'cpus': 2, 'cores_per_cpu': 8, "
                               "'threads_per_core': 1, 'accelerator': None, "
                               "'memory': '256.0'}",
            'sla': {'description': 'The jobs will be scheduled using the '
                                   'clusters standard batch scheduling '
                                   'policy.',
```

(continues on next page)

(continued from previous page)

```
                'name': 'Standard'}},
            {'display_description': 'The CFMS cluster is built using the Cray '
                                    'CS-400 solution, with parallel file '
                                    'storage provided by ArcaStream, based '
                                    'upon IBM Spectrum Scale (formerly known '
                                    'as IBM GPFS). The cluster includes '
                                    'latest generation Intel E5-26XX v4 '
                                    '(Broadwell) Xeon CPUs. The Low SLA gives '
                                    'access to more resources but your job '
                                    'may be pre-empted.',
            'display_name': 'CFMS - Low',
            'queue_code': 'cfms:low',
            'maintenance_mode': False,
            'max_allocation': 120,
            'max_runtime': 72,
            'reported_avail_tasks': None,
            'reported_max_tasks': None,
            'resource_config': "{'cpus': 2, 'cores_per_cpu': 12, "
                               "'threads_per_core': 1, 'accelerator': None, "
                               "'accelerator_count': 0, 'memory': '128.0'}",
            'sla': {'description': 'The Low SLA provides access to a low '
                                    'priority queue. This queue provides '
                                    'access to more resources than the '
                                    'standard queue BUT please be aware that '
                                    'your jobs are at risk of being stopped '
                                    'if a higher priority job requires the '
                                    'resources.',
                    'name': 'Low'}}
]
```

3.2.3 Listing Desktop Types

To list the types of desktop nodes available in epic use the catalog.list_desktops() method.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# List desktop types
desktops = client.catalog.list_desktops()

# Look at the results
print("Name | Version Name | Version ID | Valid Node Types | Valid connection Types")
for desktop in desktops:
    valid_connections = [conn.id for conn in desktop.connection_types]
    valid_node_types = [node_type.id for node_type in desktop.node_types]
    for version in desktop.versions:
        print("{} | {} | {} | {} | {}".format(
            desktop.name,
            version.application_version,
            version.id,
            valid_node_types,
            valid_connections
        ))
```

An example json output from list_desktops is shown below

```
[
        {'connection_types': [{'description': 'Connect using Nice DCV in '
                                              'your browser',
                               'id': 3,
                               'name': 'DCV'}],
        'description': 'NICE Desktop Cloud Visualization (DCV) enables '
                       'remote access 2D/3D interactive applications '
                       'over a standard network. EPIC will start a DCV '
                       'instance that you can connect to with your '
                       'browser with several versions of Paraview '
                       'installed and ready to go.',
        'id': 2,
        'image': '/media/viz/dcv.png',
        'name': 'DCV (Paraview)',
        'node_types': [{'cores': 4,
                        'description': '4 Broadwell CPU Cores, 30.5GiB '
                                       'Memory, 1 x Tesla M60 GPU with '
                                       '2048 CUDA cores and 8GB GPU '
                                       'Memory',
                        'gpus': 1,
                        'id': 1,
                        'name': 'Standard GPU Node'},
                        {'cores': 32,
                        'description': '32 Broadwell CPU Cores, 244GiB '
                                       'Memory, 2 x Tesla M60 GPU with '
                                       '2048 CUDA cores and 8GB GPU '
                                       'Memory',
                        'gpus': 2,
                        'id': 2,
                        'name': 'Large GPU Node'},
                        {'cores': 64,
                        'description': '64 Broadwell CPU Cores, 488GiB '
                                       'Memory, 4 x Tesla M60 GPU with '
                                       '2048 CUDA cores and 8GB GPU '
                                       'Memory',
                        'gpus': 4,
                        'id': 3,
                        'name': 'Extra Large GPU Node'}],
        'versions': [{'application_version': 'DCV 2017', 'id': 4}]},
        {'connection_types': [{'description': 'Connect using Nice DCV in '
                                              'your browser',
                               'id': 3,
                               'name': 'DCV'}],
        'description': 'zCAD is an CAD repair and mesh generation tool '
                       'from Zenotech. EPIC will start a DCV instance '
                       'that you can connect to with your browser with '
                       'zCAD and other Zenotech tools installed and '
                       'ready to go.',
        'id': 3,
        'image': '/media/viz/zcad.png',
        'name': 'zCAD',
        'node_types': [{'cores': 4,
                        'description': '4 Broadwell CPU Cores, 30.5GiB '
                                       'Memory, 1 x Tesla M60 GPU with '
                                       '2048 CUDA cores and 8GB GPU '
                                       'Memory',
                        'gpus': 1,
```

(continues on next page)

(continued from previous page)

```
                        'id': 1,
                        'name': 'Standard GPU Node'},
                        {'cores': 32,
                        'description': '32 Broadwell CPU Cores, 244GiB '
                                       'Memory, 2 x Tesla M60 GPU with '
                                       '2048 CUDA cores and 8GB GPU '
                                       'Memory',
                        'gpus': 2,
                        'id': 2,
                        'name': 'Large GPU Node'},
                        {'cores': 64,
                        'description': '64 Broadwell CPU Cores, 488GiB '
                                       'Memory, 4 x Tesla M60 GPU with '
                                       '2048 CUDA cores and 8GB GPU '
                                       'Memory',
                        'gpus': 4,
                        'id': 3,
                        'name': 'Extra Large GPU Node'}],
        'versions': [{'application_version': '2016.9', 'id': 5}]}
]
```

3.3 Jobs

The job client gives access to job related methods.

3.3.1 Listing Jobs

To list jobs use the list_jobs() method. You can filter by cluster name or by available application version id.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

jobs = client.job.list()

print("ID | Name | Application | Status")
for job in jobs:
    print("{} | {} | {} | {}".format(job.id, job.name, job.app, job.status))
```

An example output is shown below.

```
[
            {'app': 'OpenFOAM (v1606+)',
            'application_version': "openfoam:v1606+",
            'config': {'data_sync_interval': 0,
                       'overwrite_existing': True,
                       'upload': ['failure', 'cancel', 'complete']},
            'cost': '£5.18',
            'finished': True,
            'id': 16,
            'invoice_reference': None,
            'name': 'motorBike',
            'project': None,
```

(continues on next page)

(continued from previous page)

```
            'resource': {'display_description': 'Amazon Web Services offers '
                                                'flexible infrastructure '
                                                'services on demand. '
                                                'Zenotech use these services '
                                                'to offer HPC on demand via '
                                                'EPIC. This cluster is built '
                                                'from C4.8xlarge Compute '
                                                'Optimised instances '
                                                'connected by the AWS '
                                                'Enhanced networking. The '
                                                'queue uses the AWS Spot '
                                                'Market, this gives access '
                                                'to unused resources at a '
                                                'reduced cost but please be '
                                                'aware there is a risk that '
                                                'the nodes may be reclaimed '
                                                'if demand rises.',
                         'display_name': 'AWS C5 Spot',
                         'queue_code': 'aws:c5',
                         'maintenance_mode': False,
                         'max_allocation': 20,
                         'max_runtime': 36,
                         'reported_avail_tasks': None,
                         'reported_max_tasks': None,
                         'resource_config': "{'cpus': 2, 'cores_per_cpu': "
                                            "18, 'threads_per_core': 2, "
                                            "'accelerator': None, "
                                            "'memory': '60.0'}",
                         'sla': {'description': 'The nodes used may be '
                                                'reclaimed if demand for '
                                                'resources increases, if '
                                                'this happens your job may '
                                                'be stopped and requeued.',
                                 'name': 'Spot'}},
            'status': 'Job Cancelled',
            'submitted_at': '2020-10-01T09:37:40.674500Z',
            'submitted_by': 'Mike Turner'},
            {'app': 'OpenFOAM (v1606+)',
            'application_version': "openfoam:v1606+",
            'config': {'data_sync_interval': 0,
                       'overwrite_existing': True,
                       'upload': ['failure', 'cancel', 'complete']},
            'cost': '£5.18',
            'finished': True,
            'id': 17,
            'invoice_reference': None,
            'name': 'motorBike',
            'project': None,
            'resource': {'display_description': 'Amazon Web Services offers '
                                                'flexible infrastructure '
                                                'services on demand. '
                                                'Zenotech use these services '
                                                'to offer HPC on demand via '
                                                'EPIC. This cluster is built '
                                                'from C4.8xlarge Compute '
                                                'Optimised instances '
                                                'connected by the AWS '
```

(continues on next page)

(continued from previous page)

```
                                        'Enhanced networking. The '
                                        'queue uses the AWS Spot '
                                        'Market, this gives access '
                                        'to unused resources at a '
                                        'reduced cost but please be '
                                        'aware there is a risk that '
                                        'the nodes may be reclaimed '
                                        'if demand rises.',
                    'display_name': 'AWS C5 Spot',
                    'queue_code': 'aws:c5',
                    'maintenance_mode': False,
                    'max_allocation': 20,
                    'max_runtime': 36,
                    'reported_avail_tasks': None,
                    'reported_max_tasks': None,
                    'resource_config': "{'cpus': 2, 'cores_per_cpu': "
                                       "18, 'threads_per_core': 2, "
                                       "'accelerator': None, "
                                       "'memory': '60.0'}",
                    'sla': {'description': 'The nodes used may be '
                                           'reclaimed if demand for '
                                           'resources increases, if '
                                           'this happens your job may '
                                           'be stopped and requeued.',
                            'name': 'Spot'}},
 'status': 'Job Complete',
 'submitted_at': '2020-10-01T13:33:54.569241Z',
 'submitted_by': 'Mike Turner'},
 {'app': 'OpenFOAM (v1606+)',
 'application_version': "openfoam:v1606+",
 'config': {'data_sync_interval': 0,
            'overwrite_existing': True,
            'upload': ['failure', 'cancel', 'complete']},
 'cost': '£5.18',
 'finished': True,
 'id': 18,
 'invoice_reference': None,
 'name': 'motorBike',
 'project': None,
 'resource': {'display_description': 'Amazon Web Services offers '
                                     'flexible infrastructure '
                                     'services on demand. '
                                     'Zenotech use these services '
                                     'to offer HPC on demand via '
                                     'EPIC. This cluster is built '
                                     'from C4.8xlarge Compute '
                                     'Optimised instances '
                                     'connected by the AWS '
                                     'Enhanced networking. The '
                                     'queue uses the AWS Spot '
                                     'Market, this gives access '
                                     'to unused resources at a '
                                     'reduced cost but please be '
                                     'aware there is a risk that '
                                     'the nodes may be reclaimed '
                                     'if demand rises.',
              'display_name': 'AWS C5 Spot',
```

(continues on next page)

(continued from previous page)

```
                        'queue_code': 'aws:c5',
                        'maintenance_mode': False,
                        'max_allocation': 20,
                        'max_runtime': 36,
                        'reported_avail_tasks': None,
                        'reported_max_tasks': None,
                        'resource_config': "{'cpus': 2, 'cores_per_cpu': "
                                           "18, 'threads_per_core': 2, "
                                           "'accelerator': None, "
                                           "'memory': '60.0'}",
                        'sla': {'description': 'The nodes used may be '
                                               'reclaimed if demand for '
                                               'resources increases, if '
                                               'this happens your job may '
                                               'be stopped and requeued.',
                                'name': 'Spot'}},
            'status': 'Job Complete',
            'submitted_at': '2020-10-01T13:40:45.102124Z',
            'submitted_by': 'Mike Turner'}
]
```

To get the details of a specific job with a known ID using the get_job_details method.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# Get details for job id 18
jobs = client.job.get_details(18)
```

3.3.2 Checking job logs

Job logs are available for each step that makes up the job. The step id's for each job are listed in the job details and with that ID you can fetch the current log tails.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# Get the latest tail of the log files, EPIC will request an update of the logs for␣
↪running jobs
log_obj = client.job.get_step_logs(50)

# Print stdout from the logs
print(log_obj.stdout)

# Get the latest tail of the log files without requesting a refresh
log_obj = client.job.refresh_step_logs(50, refresh=False)
```

3.3.3 Fetching job residuals

For applications that support residuals you can fetch the available variable names and then request the data for specific variables.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# Get the list of available variables to plot for job id 101
available_variables = client.job.get_job_residual_names(101)

# Print variable names
print(available_variables)

# Get the data for variables "Ux" & "Uy". By default a value of xaxis is always␣
↪returned.
variables = client.job.get_job_residual_values(50, ['Ux','Uy'])

for var in variables:
    print("Var name = {}".format(var.variable_name))
    print("Var values = {}".format(var.values))
```

3.3.4 Submitting Jobs

Submitting jobs is done with the client.job.submit() method. PyEpic has application specfic helper classes to make the submission as simple as possible, see the application examples below.

3.3.5 OpenFOAM

To create and submit an OpenFOAM job you can use the *pyepic.applications.openfoam.OpenFoamJob* class. Prior to creating the job you need to know the code of the application version you wish to use, the code of the batch queue you want to submit to and the path to the root of the openfoam case. The data for this case is assumed to have already been uploaded to your EPIC data store. The app and queue codes can be obtained from the catalog endpoints.

```
from pyepic import EPICClient
from pyepic.applications.openfoam import OpenFoamJob

client = EPICClient("your_api_token_goes_here")

# Create the job using application version with id "openfoam:v1606"
openfoam_job = OpenFoamJob("openfoam:v1606", "job_name", "epic://my_data/foam/")

# Configure the solver to run on 24 paritions for a maximum of 12 hours
openfoam_job.solver.partitions = 24
openfoam_job.solver.runtime = 12

# Create the specification for submission to queue with code "aws:c5"
job_spec = openfoam_job.get_job_create_spec("aws:c5")

# Submit the job
job = client.job.submit(job_spec)
```

The submit_job method will return a job object. The job_id can be extraced from this object for future queries.

3.3.6 zCFD

To create and submit an zCFD job you can use the *pyepic.applications.zcfd.ZCFDJob* class. Prior to creating the job you need to know the code of the application version you wish to use, the id of the batch queue you want to submit to and the path to the root of the zcfd case. The data for this case is assumed to have already been uploaded to your EPIC data store. If your data is in your EPIC data store in a folder called 'work/zcfd' then the data path for the method would be 'epic://work/zcfd/'. The app and queue codes can be obtained from the catalog endpoints.

```
from pyepic import EPICClient
from pyepic.applications.zcfd import ZCFDJob

client = EPICClient("your_api_token_goes_here")

# Create a zCFD job using application version id "zcfd:2021.1.1"
zcfd_job = ZCFDJob("zcfd:2021.1.1", "zcfd_case", "epic://work/zcfd/", "fv.py", "box.
↪hdf5", cycles=1000, restart=False, partitions=24)

# Configure the solver to run for a maximum of 12 hours
zcfd_job.zcfd.runtime = 12

# Create the specification for submission to queue "aws:p4d"
job_spec = zcfd_job.get_job_create_spec("aws:p4d")

# Submit the job
job = client.job.submit(job_spec)

job_id = job[0].id

print(f"Submitted job with id {id}")
```

3.4 Job Arrays

Job arrays allow you to submit a set of jobs in one submission. Jobs in an array can share common data to reduce the volume of data that you need to transfer. To use arrays you should structure your input data to have a shared root folder. This root folder can then contain the "common" folder and multiple job folders.

The example below shows a job array for zCFD. The example folder structure for this case is:

epic://work/zcfd/ The array root folder for the case.

epic://work/zcfd/common/ The folder containing files common to all jobs in the array, for example the *box.hdf5* mesh. This must be called "common"

epic://work/zcfd/run.1/ The folder with the customised input for the first job, for example the *fv_1.py* python control file.

epic://work/zcfd/run.2/ The folder with the customised input for the second job, for example the *fv_2.py* python control file.

```
import pyepic
from pyepic.applications.zcfd import ZCFDJob
from pyepic.applications.base import JobArray

client = EPICClient("your_api_token_goes_here")
```

(continues on next page)

(continued from previous page)

```
# Create a new JobArray called my_job_array with epic://work/zcfd/ as the array_root_
↪folder folder
job_array = JobArray("my_job_array", "epic://work/zcfd/")

# Create two zCFD jobs using application version id "zcfd:2021.1.1"
zcfd_job_1 = ZCFDJob("zcfd:2021.1.1", "zcfd_run_1", "epic://work/zcfd/run.1/", "fv_1.
↪py", "box.hdf5", cycles=1000, restart=False, partitions=24)
zcfd_job_2 = ZCFDJob("zcfd:2021.1.1", "zcfd_run_2", "epic://work/zcfd/run.2/", "fv_2.
↪py", "box.hdf5", cycles=1000, restart=False, partitions=24)

# Add the jobs to the array
job_array.add_job(zcfd_job_1)
job_array.add_job(zcfd_job_2)

# Create the specification for submission to queue "aws:p4d"
array_spec = job_array.get_job_create_spec("aws:p4d")

# Submit the job array
jobs = client.job.submit(array_spec)

job_1_id = job[0].id
job_2_id = job[1].id
```

3.5 Data

EPIC uses AWS S3 as an object store for data. The commands in this API use the boto3 library to communicate with the backend S3 services. Using PyEpic data in your EPIC data store can be referenced using an EPIC data url. The client class for data functions is `pyepic.client.EPICClient.data`. For example if you have a folder in your EPIC data store called "MyData" then the data url would be "epic://MyData/", a file called "data.in" in that folder would be "epic://MyData/data.in".

3.5.1 Listing a folder

List a folder using the ls method.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

directory_listing = client.data.ls("epic://Folder/data/")

print("Path | Name | Is folder? | File size")
for item in directory_listing:
    print("{} | {} | {} | {}".format(item.obj_path, item.name, item.folder, item.
↪size))
```

3.5.2 Downloading a file

PyEpic lets you download files directly to the local disk or to a File-like object.

To download to a file:

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

client.data.download_file("epic://MyData/data.in", "./data.in")
```

To download to an in-memory object, for example BytesIO:

```
import io
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# Create a new BytesIO object
my_data = io.BytesIO()

# Download contents of epic file into my_data
client.data.download_file("epic://MyData/data.in", my_data)

# Do something with the data in memory
my_data.seek(0)
my_data.read()
```

3.5.3 Uploading a file

In a similar way to downloading, PyEpic lets you upload from a local file of a file-like object. If you specify a directory as the target then the filename will be taken from the localfile if available.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# Upload data.new to epic://MyData/data.new
client.data.upload_file("./data.new", "epic://MyData/")
```

To upload to an in-memory object, for example BytesIO:

```
import io
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# Create a new BytesIO object
my_data = io.BytesIO(b"This is new data")

# Upload contents of my_data to epic file
client.data.upload_file(my_data, "epic://MyData/data.new")
```

3.5.4 Copying whole folders/directories

upload_file and download_file are useful for dealing with single files but often you will need to upload or download whole folders. To do this you can use the sync method. This takes a source_path and a target_path than can either be a local path or a remote epic:// url. This means you can either sync from your local files upto EPIC or from EPIC back to your local files.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# Copy everything in my local dir ./data/ to a path on EPIC call new_data.
# If the files already exist in epic://new_data/ then still copy them if the local␣
↪ones are newer.
client.data.sync("./data/", "epic://new_data/", overwrite_existing=True)
```

You can get more information about the copy progress my passing a method in the "callback" kwarg.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

def my_callback(source_path, target_path, uploaded, dryrun):
    print("Callback. Source={} Target={} Uploaded={} Dryrun={}".format(source_path,␣
↪target_path, uploaded, dryrun))

# Copy everything in my local dir ./data/ to a path on EPIC call new_data
client.data.sync("./data/", "epic://new_data/", callback=my_callback, overwrite_
↪existing=True)
```

When uploading large datasets then the "dryrun" kwarg lets you see what PyEpic will do without actually performming the copies.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

def my_callback(source_path, target_path, uploaded, dryrun):
    print("Callback. Source={} Target={} Uploaded={} Dryrun={}".format(source_path,␣
↪target_path, uploaded, dryrun))

# Copy everything in my local dir ./data/ to a path on EPIC call new_data
client.data.sync("./data/", "epic://new_data/", dryrun=True, callback=my_callback,␣
↪overwrite_existing=True)
```

3.5.5 Deleting files or folders

PyEpic lets you delete indivdual files or whole folders from EPIC.

To delete to a single file:

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

client.data.delete("epic://MyData/data.in")
```

To delete a folder and its contents:

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

client.data.delete("epic://MyData/")
```

3.6 Desktops

3.6.1 Listing Desktop Instances

To list your desktop instances use the list and get_details methods in `pyepic.client.EPICClient.desktops`.

```
from pyepic import EPICClient

client = EPICClient("your_api_token_goes_here")

# List all of my desktop instances
desktops = client.desktops.list()

# Get the details of desktop id 3
desktop_instance = client.desktops.get_details(3)
```

3.6.2 Getting a quote for a Desktop

PyEpic provides the helper class `pyepic.desktops.Desktop` to help create Desktops in EPIC. To get a quote create an instance of this class and then use that the retrieve the quote via the desktop client class. The valid application_version, node_type and connection_type values can be retrieved via `pyepic.EPICClient.catalog`..

```
from pyepic import EPICClient
from pyepic.desktops import Desktop

client = EPICClient("your_api_token_goes_here")

# Create a desktop spec
my_desktop = Desktop("epic://data_path/", application_version=5, node_type=1,␣
↪connection_type=3)

# Set the runtime to two hours
my_desktop.runtime = 2

# Get a quote for this desktop
quote = client.desktops.get_quote(my_desktop.get_quote_spec())
```

An example response for the quote is shown below.

```
{'cost': {'amount': 0.71, 'currency': 'GBP'}, 'reason': '', 'valid': True}
```

3.6.3 Launching a desktop

PyEpic provides the helper class `pyepic.desktops.Desktop` to help create Desktops in EPIC. To launch a desktop create an instance of this class and then use that to launch the desktop via the desktop client class. The valid application_version, node_type and connection_type values can be retrieved via `pyepic.EPICClient.catalog`.

```
from pyepic import EPICClient
from pyepic.desktops import Desktop

client = EPICClient("your_api_token_goes_here")

# Create a desktop spec
my_desktop = Desktop("epic://data_path/", application_version=5, node_type=1,␣
↪connection_type=3)

# Set the runtime to two hours
my_desktop.runtime = 2

# Launch this desktop
instance = client.desktops.launch(my_desktop.get_launch_spec())

# Get the newly created desktop instance id.
id = instance.id
```

The launch method returns a `epiccore.models.DesktopInstance` object that includes the newly created desktop instance ID. If there is an issue with the specification then launch will return the list of validation errors. An example response is shown below.

```
{'application': {'application': {'description': 'zCAD is an CAD repair and '
                                                'mesh generation tool from '
                                                'Zenotech. EPIC will start a '
                                                'DCV instance that you can '
                                                'connect to with your browser '
                                                'with zCAD and other Zenotech '
                                                'tools installed and ready to '
                                                'go.',
                                 'image': '/media/viz/zcad.png',
                                 'name': 'zCAD'},
                 'application_version': '2016.9',
                 'id': 5},
'connection_string': None,
'connection_type': {'description': 'Connect using Nice DCV in your browser',
                    'id': 3,
                    'name': 'DCV'},
'created': datetime.datetime(2020, 11, 27, 9, 19, 47, 127429, tzinfo=tzutc()),
'id': 11,
'launched_by': 'Danny Develop',
'status': 'new',
'team': None}
```

3.6.4 Terminating a desktop

Terminate a desktop using the terminate client method and the Desktops ID.

```
from pyepic import EPICClient
from pyepic.desktops import Desktop

client = EPICClient("your_api_token_goes_here")

# Terminate desktop with ID 3
client.desktops.terminate(3)
```

3.7 Teams

```
from pyepic import EPICClient
from pyepic.desktops import Desktop

client = EPICClient("your_api_token_goes_here")

# List teams
teams = client.teams.list()

for team in teams:
    print(team)

# Get team ID 334
team = client.teams.get_details(334)
```

3.8 Projects

```
from pyepic import EPICClient
from pyepic.desktops import Desktop

client = EPICClient("your_api_token_goes_here")

# List projects
projects = client.projects.list()

for project in projects:
    print(project)

# Get project ID 102
project = client.projects.get_details(102)
```

CHAPTER

FOUR

FAQ

4.1 Where do I get a token from?

You generate an API token by logging into EPIC and then look for the "API Token" section on the Accounts -> Credentials page.

4.2 What about uploading/downloading data?

At the time we don't support uploading and downloading data directly via the API. This is because it is more efficient to transfer your data directly to the data store without going via EPIC. To do this any S3 compatable tool or SDK can be used, for example boto3. Search for data in the EPIC knowledge base to find details on how to do this.

CHAPTER

FIVE

PYEPIC

5.1 pyepic package

5.1.1 Subpackages

pyepic.applications package

Submodules

pyepic.applications.base module

class pyepic.applications.base.**Config**
Bases: object

The Job Configuration

Variables

- **overwrite_existing** (*bool*) – Should data created on the remote cluster overwrite older data that exists in the epic data store
- **upload** (*list*) – Which job states should trigger a data upload
- **data_sync_interval** (*int*) – How frequently should the data be periodically uploaded while the job is still running, set to 0 to disable.
- **project_id** (*int, optional*) – ID of the EPIC project to run this job in

get_configuration()
Get a JobConfiguration for this job :return: Job Configuration :rtype: class:*epiccore.models.JobConfiguration*

class pyepic.applications.base.**Distribution**(*value*)
Bases: enum.Enum

How should the partitions/processes or tasks be distrubuted on the remote cluster, 1 per CORE/SOCKET/NODE or DEVICE

CORE = 'core'

DEVICE = 'device'

NODE = 'node'

SOCKET = 'socket'

class pyepic.applications.base.**Job**(*application_version*, *job_name*, *path*)
Bases: object

An EPIC Job Definition

Parameters

- **application_version** (*str*) – The Code of the BatchApplicationVersion that this job will user
- **job_name** (*str*) – A user friendly name for the job
- **path** (*str*) – The path to the root of the OpenFOAM job, formed as an epic url (e.g. "epic://path_to/data")

Variables

- **job_name** (*str*) – A user friendly name for the job
- **path** (*str*) – The path to the root of the OpenFOAM job, formed as an epic url (e.g. "epic://path_to/data")
- **config** (*Config*) – The Job configuration options object
- **steps** (*list*) – The job steps that make up this job

add_step(*job_step*)
Add a new step to this job

Parameters **job_step** (*JobStep*) – The step to append to this job

get_job_create_spec(*queue_code*)
Get a JobArraySpec for this job. The JobArraySpec can be used to submit the job to EPIC via the client.

Parameters **queue_code** (*str*) – The code of the EPIC batch queue to submit to

Returns Job ArraySpecification

Return type class:*epiccore.models.JobArraySpec*

get_job_spec()
Get a JobSpec for this job

Returns Job Specification

Return type class:*epiccore.models.JobSpec*

class pyepic.applications.base.**JobArray**(*array_name*, *array_root_folder*)
Bases: object

A helper class for submitting an array of jobs to EPIC.

Parameters

- **array_name** (*str*) – The name to give the array in EPIC
- **array_root_folder** (*str*) – The epic data path to the root of the array data folder, formed as an epic url (e.g. "epic://path_to/data"). Any data in a folder called "common" within this folder will be shared between all jobs in the array.

Variables

- **config** (*Config*) – The Job configuration options object, common to all jobs in the array
- **jobs** (*list*) – The jobs that make up this array

add_job(*job*)
Add a job to this array

Parameters **job** (class:*Job*) – The code of the EPIC batch queue to submit to

get_job_create_spec(*queue_code*)

Get a JobArraySpec for this array. The JobArraySpec can be used to submit the array to EPIC via the client.

Parameters **queue_code** (*str*) – The code of the EPIC batch queue to submit to

Returns Job ArraySpecification

Return type class:*epiccore.models.JobArraySpec*

class pyepic.applications.base.**JobStep**(*execute_step=True*)

Bases: object

A Step within an EPIC Job

Parameters **execute_step** (*int*) – Enable this step as part of this job

Variables

- **step_name** (*str*) – Name of step, this is application specific
- **execute** (*bool*) – Should this step execute when the job is submitted
- **partitions** (*int*) – How many partitions/processes make up this step
- **task_distribution** (*Distribution*) – How are the partitions distributed to the hardware
- **runtime** (*int*) – The maximum runtime of this step in hours
- **run_if_previous_step_fails** (*bool*) – Should this step execute if the previous step fails
- **hyperthreading** (*bool*) – Does this step count hyperthreaded cores as individual CPUs?

get_task_spec()

Get a JobTaskSpec for this job step

Returns Job Task Specification

Return type epiccore.models.JobTaskSpec

class pyepic.applications.base.**Upload**(*value*)

Bases: enum.Enum

Should excluded files be uploaded? Yes, No or only when the job finishes in an error state.

NO = 'no'

ON_ERROR = 'error'

YES = 'yes'

pyepic.applications.openfoam module

class pyepic.applications.openfoam.**BlockMeshStep**(**args*, ***kwargs*)
Bases: *pyepic.applications.base.JobStep*

BlockMeshStep step of OpenFOAM

class pyepic.applications.openfoam.**DecomposeParStep**(**args*, ***kwargs*)
Bases: *pyepic.applications.base.JobStep*

DecomposeParStep step of OpenFOAM

class pyepic.applications.openfoam.**OpenFoamJob**(*foam_version*, *job_name*, *data_path*)
Bases: *pyepic.applications.base.Job*

A helper class for submitting an OpenFOAM job to EPIC.

Parameters

- **foam_version** (*str*) – The code of the BatchApplicationVersion of OpenFOAM to use
- **job_name** (*str*) – The name to give the job in EPIC
- **data_path** (*str*) – The epic data path to the OpenFOAM case directory

Variables

- **blockMesh** (*BlockMeshStep*) – blockMesh JobStep object
- **decomposePar** (*DecomposeParStep*) – decomposePar JobStep object
- **solver** (*SolverStep*) – initial solver JobStep object
- **reconstructPar** (*ReconstructParStep*) – reconstructPar JobStep object
- **clear_partitions** – Delete any existing processor directories before running job
- **sync_processor_directories** (*base.Upload*) – Upload processor after job completion, default No

get_applications_options()
Get application configuration options for submission to EPIC

Returns Dictionary of the job configuration

Return type dict

get_job_create_spec(*queue_code*)
Get a JobSpec for this job

Returns Job Specification

Return type class:*epiccore.models.JobSpec*

class pyepic.applications.openfoam.**Reconstruct**(*value*)
Bases: enum.Enum

An enumeration.

ALL = 'all'

LATEST = 'latest'

TIME = 'time'

class pyepic.applications.openfoam.**ReconstructParStep**(**args*, ***kwargs*)
Bases: *pyepic.applications.base.JobStep*

ReconstructPar step of OpenFOAM

Variables

- **run_if_previous_step_fails** (*bool*) – Run if solver fails, defaults to True
- **reconstruct_option** (*Reconstruct*) – Which time step to reconstruct. Defaults to ALL
- **reconstruct_time** (*int*) – If reconstruct_option == TIME then which timestep to reconstruct.

class pyepic.applications.openfoam.**SolverStep**(**args*, ***kwargs*)
Bases: *pyepic.applications.base.JobStep*

Solver step of OpenFOAM

Variables

- **run_if_previous_step_fails** (*bool*) – Run if previous step fails, defaults to False
- **stopAt** (*StopAt*) – When to stop the solver. Defaults to END_TIME
- **startFrom** (*StartFrom*) – Which timestep to start the solver from. Defaults to LATEST
- **endTime** (*int*) – If stopAt == END_TIME then which timestep to stop the solver at.
- **startTime** (*int*) – If startFrom == START then which timestep to start the solver from.

class pyepic.applications.openfoam.**StartFrom**(*value*)
Bases: enum.Enum

An enumeration.

FIRST = 'firstTime'

LATEST = 'latestTime'

START = 'startTime'

class pyepic.applications.openfoam.**StopAt**(*value*)
Bases: enum.Enum

An enumeration.

END_TIME = 'endTime'

NEXT_WRITE = 'nextWrite'

NO_WRITE_NOW = 'noWriteNow'

WRITE_NOW = 'writeNow'

pyepic.applications.zcfd module

class pyepic.applications.zcfd.**ZCFDJob**(*zcfd_version*, *job_name*, *data_path*, *case_name*, *problem_name*, *override_file=None*, *cycles=100*, *restart=False*, *partitions=1*)

Bases: *pyepic.applications.base.Job*

A helper class for submitting an zCFD job to EPIC.

Parameters

- **zcfd_version** (*str*) – The code of the BatchApplicationVersion of zCFD to use
- **job_name** (*str*) – The name to give the job in EPIC
- **data_path** (*str*) – The epic data path to the zCFD case directory
- **case_name** (*str*) – The name of the python control file for the case
- **problem_name** (*str*) – The name of the hdf5 mesh file
- **override_file** (*str, optional*) – The name of the zcfd override file for overset meshes. Defaults to None.
- **cycles** (*int, optional*) – How many cycles to run for. Default 100.
- **restart** (*bool, optional*) – Is the case a restart from a previous solution. Default False.
- **partitions** (*int, optional*) – How many parallel partitions should the case use. Default 1.

Variables ***zcfd*** (*ZCFDStep*) – zCFD JobStep object

get_applications_options()

Get application configuration options for submission to EPIC

Returns Dictionary of the job configuration

Return type dict

get_job_create_spec(*queue_code*)

Get a JobSpec for this job

Returns Job Specification

Return type class:*epiccore.models.JobSpec*

class pyepic.applications.zcfd.**ZCFDStep**(*case_name*, *problem_name*, *override_file*, *cycles*, *restart=False*, *partitions=1*, *execute_step=True*)

Bases: *pyepic.applications.base.JobStep*

zCFD Solver

Variables

- **case_name** (*str*) – The name of the python control file for the case
- **problem_name** (*str*) – The name of the hdf5 mesh file
- **override_file** (*str*) – The name of the zcfd override file for overset meshes
- **cycles** (*int*) – How many cycles to run for
- **restart** (*bool*) – Is the case a restart from a previous solution
- **partitions** (*int*) – How many parallel partitions should the case use

pyepic.applications.msc_nastran module

class pyepic.applications.msc_nastran.**NastranJob**(*nastran_version*, *job_name*, *data_path*, *dat_file*, *nastran_licence_server*, *partitions=1*)

Bases: *pyepic.applications.base.Job*

A helper class for submitting an Nastran job to EPIC.

Parameters

- **nastran_version** (*str*) – The code of the BatchApplicationVersion of Nastran to use
- **job_name** (*str*) – The name to give the job in EPIC
- **data_path** (*str*) – The epic data path to the Nastran case directory
- **dat_file** (*str*) – The name of the nastran data file
- **nastran_licence_server** (*str*) – The licence server and port for nastran

Variables **nastran** (*NastranStep*) – Nastran solver JobStep object

get_applications_options()
Get application configuration options for submission to EPIC

Returns Dictionary of the job configuration

Return type dict

get_job_create_spec(*queue_code*)
Get a JobSpec for this job

Returns Job Specification

Return type class:*epiccore.models.JobSpec*

class pyepic.applications.msc_nastran.**NastranStep**(*dat_file*, *nastran_licence_server*, *cycles*, *restart=False*, *partitions=1*, *execute_step=True*)

Bases: *pyepic.applications.base.JobStep*

Nastran Solver

Variables

- **dat_file** (*str*) – The name of the nastran data file
- **nastran_licence_server** (*str*) – The licence server and port for nastran
- **partitions** (*int*) – How many parallel partitions should the case use

Module contents

pyepic.client package

Submodules

pyepic.client.base module

class pyepic.client.base.**Client**(*connection_token*, *connection_url='https://epic.zenotech.com/api/v2'*)

Bases: object

Base client class for API wrappers

Parameters

- **connection_token** (*str*) – Your EPIC API authentication token
- **connection_url** (*str, optional*) – The API URL for EPIC, defaults to "https://epic.zenotech.com/api/v2"

set_limt(*limit*)

class pyepic.client.base.**EPICClient**(*connection_token*, *connection_url='https://epic.zenotech.com/api/v2'*)

Bases: object

A wrapper class around the epiccore API.

Parameters

- **connection_token** (*str*) – Your EPIC API authentication token
- **connection_url** (*str, optional*) – The API URL for EPIC, defaults to "https://epic.zenotech.com/api/v2"

Variables

- ***job*** (JobClient) – API to Job functions
- ***catalog*** (CatalogClient) – API to Catalog functions
- ***desktops*** (DesktopClient) – API to Desktops functions
- ***projects*** (ProjectClient) – API to Projects functions
- ***teams*** (TeamsClient) – API to Teams functions

pyepic.client.catalog module

class pyepic.client.catalog.**CatalogClient**(*connection_token*, *connection_url='https://epic.zenotech.com/api/v2'*)

Bases: *pyepic.client.base.Client*

A wrapper class around the epiccore Catalog API.

Parameters

- **connection_token** (*str*) – Your EPIC API authentication token
- **connection_url** (*str, optional*) – The API URL for EPIC, defaults to "https://epic.zenotech.com/api/v2"

application_details(*application_id*)

Get the details of application with id application_id

Parameters **application_id** (*int*) – ID of application to get details for

Returns BatchQueueDetails

Return type epiccore.models.BatchApplicationDetails

list_applications(*product_name=None*)

List the applications available in EPIC

Parameters **product_name** (*str, optional*) – Filter clusters by application name.

Returns Iterable collection of BatchApplicationDetails

Return type collections.Iterable[`epiccore.models.BatchApplicationDetails`]

list_clusters(*cluster_name=None*, *queue_name=None*, *allowed_apps=None*)
List the clusters available in EPIC

Parameters

- **cluster_name** (*str, optional*) – Filter clusters by cluster name.
- **queue_name** (*str, optional*) – Filter clusters by queue name.
- **allowed_apps** (*str, optional*) – Filter clusters by those with application code available.

Returns Iterable collection of BatchQueueDetails

Return type collections.Iterable[`epiccore.models.BatchQueueDetails`]

list_desktops()
List the available Desktops in EPIC

Returns Iterable collection of DesktopNodeApp

Return type collections.Iterable[`epiccore.models.DesktopNodeApp`]

queue_details(*queue_id*)
Get the details of queue with id queue_id

Parameters **queue_id** (*int*) – ID of queue to get details for

Returns BatchQueueDetails

Return type `epiccore.models.BatchQueueDetails`

pyepic.client.data module

class pyepic.client.data.**DataClient**(*connection_token*, *connection_url='https://epic.zenotech.com/api/v2'*)
Bases: *pyepic.client.base.Client*

A wrapper class around the epiccore Data API.

Parameters

- **connection_token** (*str*) – Your EPIC API authentication token
- **connection_url** (*str, optional*) – The API URL for EPIC, defaults to "https://epic.zenotech.com/api/v2"

delete(*epic_path*, *dryrun=False*)

Delete the file of folder at epic_path

param epic_path Path of a file or folder to delete in the form epic://[<folder>]/<file>

type epic_path str

param dryrun If dryrun is True then return a list of files that would be deleted without actually deleting them

type dryrun bool

return List of the files deleted

rtype List[str]

download_file(*epic_path*, *destination*)

Download the contents of epic_path

param epic_path Path of a file in the form epic://[<folder>]/<file>

type epic_path str

param destination Location to download file to, can be a string or a writable file-like object

type destination str

ls(*epic_path*)

List the files and folders at the given path

param epic_path Path in the form epic://[<folder>]/

type epic_path str

return Iterable collection of DataObject

rtype collections.Iterable[*pyepic.client.data.DataObject*]

meta_source = 'SDK'

sync(*source_path*, *target_path*, *dryrun=False*, *overwrite_existing=False*, *callback=None*, *threads=3*, *cancel_event=None*)

Synchronize the data from one directory to another, source_path or target_path can be a remote folder or a local fold

param source_path Source folder to syncronise from. For remote folders use form epic://[<folder>]/<file>.

type source_path str

param target_path Target folder to syncronise to. For remote folders use form epic://[<folder>]/<file>.

type target_path str

param dryrun If dryrun == True then no actual copy will take place but the callback will be called with the generated source and target paths. This can be useful for checking before starting a large upload/download.

type dryrun bool, optional

param overwrite_existing If overwrite_existing == True then files with newer modification timestamps in source_path will replace existing files in target_path

type overwrite_existing bool, optional

param callback A callback method that accepts four parameters. These are source, destination, a boolean indicating if a copy has taken place and a boolean to indicate if the copy was a dryrun. The callback is called after each file is processed.

type callback method, optional

param threads Number of threads to use for sync

type threads int, optional

param cancel_event An instance of threading.Event that can be set to cancel the sync.

type cancel_event `threading.Event`

upload_file(*file*, *epic_path*)

Upload the contents of file to epic_path

param destination Location of the file to upload OR a readable file-like object

type destination str

param epic_path Destination path of a file in the form epic://[<folder>]/<file>

type epic_path str

class pyepic.client.data.**DataObject**(*name*, *obj_path*, *folder=False*, *size=None*, *last_modified=None*)

Bases: object

Class representing a file or folder

Parameters

- **name** (*int*) – Name of the file/folder
- **obj_path** (*str*) – Path of the file/folder
- **folder** (*bool*) – Is the object a folder?
- **name** – Size of the object if available
- **last_modified** (*str*) – Last modified time if available. Datetime in ISO 8601 format, UTC timezone.

class pyepic.client.data.**DataThread**(*s3_client*, *bucket_name*, *s3_prefix*, *local_path*, *file_queue*, *cancel_event=None*, *dryrun=False*, *callback=None*, *download_thread=True*, *overwrite_existing=False*, *meta_data={}*)

Bases: threading.Thread

Thread class used internally by pyepic for managaing upload/download functions

download_key(*key_name*)

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

upload_file(*file_full_path*)

pyepic.client.desktop module

class pyepic.client.desktop.**DesktopClient**(*connection_token*, *connection_url='https://epic.zenotech.com/api/v2'*)

Bases: *pyepic.client.base.Client*

A wrapper class around the epiccore Desktop API.

Parameters

- **connection_token** (*str*) – Your EPIC API authentication token
- **connection_url** (*str, optional*) – The API URL for EPIC, defaults to "https://epic.zenotech.com/api/v2"

get_details(*id*)

Get details of desktop with ID id

Parameters **id** (*int*) – The ID of the desktop to fetch details on

Returns A desktop instance

Return type class:*epiccore.models.DesktopInstance*

get_quote(*desktop_spec*)
Get a Quote for launching the desktop on EPIC

Parameters **desktop_spec** (class:*epiccore.models.DesktopNodeQuote*) – The EPIC Desktop Quote specification

Returns A quote giving the price for the job on the available HPC queues

Return type class:*epiccore.models.PriceQuote*

launch(*desktop_spec*)
Launch the Desktop described by desktop_spec in EPIC

Parameters **desktop_spec** (class:*epiccore.models.DesktopNodeQuote*) – The EPIC Desktop Launch specification

Returns A quote giving the price for the job on the available HPC queues

Return type class:*epiccore.models.DesktopInstance*

list()
List all of your Desktops in EPIC.

Returns Iterable collection of DesktopInstance

Return type collections.Iterable[`epiccore.models.DesktopInstance`]

terminate(*id*)
Terminate Desktop job with ID id

Parameters **id** (*int*) – The ID of the Desktop to terminate

pyepic.client.job module

class pyepic.client.job.**JobClient**(*connection_token*, *connection_url='https://epic.zenotech.com/api/v2'*)
Bases: *pyepic.client.base.Client*

A wrapper class around the epiccore Job API.

Parameters

- **connection_token** (*str*) – Your EPIC API authentication token
- **connection_url** (*str, optional*) – The API URL for EPIC, defaults to "https://epic.zenotech.com/api/v2"

cancel(*job_id*)
Cancel job with ID job_id

Parameters **job_id** (*int*) – The ID of the job to cancel

get_details(*job_id*)
Get details of job with ID job_id

Parameters **job_id** (*int*) – The ID of the job to fetch details on

Returns A Job instance

Return type class:*epiccore.models.Job*

get_job_residual_names(*job_id*)
Get the names of the residual variables available forjob with id job_id

Parameters **job_id** (*int*) – The ID of the job to get the residual list for

Returns A list of variable names

Return type List[str]

get_job_residual_values(*job_id*, *variable_list*)
Get the names of the residual variables available forjob with id job_id

Parameters

- **job_id** (*int*) – The ID of the job to get the variables for
- **variable_names** (*List[str]*) – A list of the variables to return

Returns A JobResidualData object

Return type class:*epiccore.models.JobResidualData*

get_quote(*job_spec*)
Get a Quote for running a series of tasks on EPIC.

Parameters **job_spec** (class:*epiccore.models.JobSpec*) – The EPIC job specification

Returns A quote giving the price for the job on the available HPC queues

Return type class:*epiccore.models.JobQuote*

get_step_details(*step_id*)
Get the details of the step ID step_id

Parameters **step_id** (*int*) – The ID of the job step to fetch

Returns A Job Step instance

Return type class:*epiccore.models.JobStep*

get_step_logs(*step_id*, *refresh=True*, *refresh_timeout=10*)
Get the step logs for step with id step_id

Parameters

- **step_id** (*int*) – The ID of the step to fetch the logs for
- **dryrun** (*int, optional*) – If refresh is True then the request will attempt to refresh the logs before returning the latest update
- **dryrun** – How many seconds should we wait for a refresh before giving up, default 10 seconds

Returns A Job Log instance

Return type class:*epiccore.models.JobLog*

list(*limit=10*)
List all of the jobs in EPIC.

Parameters **limit** (*int*) – Maximum number of jobs to list

Returns Iterable collection of Jobs

Return type collections.Iterable[epiccore.models.Job]

list_steps(*parent_job=None*)
List all of the job steps in EPIC.

Parameters **parent_job** (*int, optional*) – The ID of the parent job to list the steps for

Returns Iterable collection of Job Steps

Return type collections.Iterable[epiccore.models.JobStep]

submit(*job_array_spec*)
Submit new job in EPIC as described by job_array_spec.

Parameters **job_array_spec** (class:*epiccore.models.JobArraySpec*) – The EPIC job specification

Returns The newly created job instance

Return type class:*epiccore.models.Job*

pyepic.client.projects module

class pyepic.client.projects.**ProjectClient**(*connection_token*, *connection_url='https://epic.zenotech.com/api/v2'*)
Bases: *pyepic.client.base.Client*

A wrapper class around the epiccore Projects API.

Parameters

- **connection_token** (*str*) – Your EPIC API authentication token
- **connection_url** (*str, optional*) – The API URL for EPIC, defaults to "https://epic.zenotech.com/api/v2"

get_details(*id: int*)
Get the details for project with ID id

Returns The Project

Return type epiccore.models.ProjectDetails

list()
List all of the projects you have access to on EPIC.

Returns An interable list of Projects

Return type collections.Iterable[epiccore.models.Project]

pyepic.client.teams module

class pyepic.client.teams.**TeamsClient**(*connection_token*, *connection_url='https://epic.zenotech.com/api/v2'*)
Bases: *pyepic.client.base.Client*

A wrapper class around the epiccore Teams API.

Parameters

- **connection_token** (*str*) – Your EPIC API authentication token
- **connection_url** (*str, optional*) – The API URL for EPIC, defaults to "https://epic.zenotech.com/api/v2"

get_details(*id: int*)
Get the details for team with ID id

Returns The Team details

Return type epiccore.models.TeamDetails

list()
List all of the teams you have access to on EPIC.

Returns An interable list of Teams

Return type collections.Iterable[epiccore.models.Team]

Module contents

pyepic.desktops package

Submodules

pyepic.desktops.desktop module

class pyepic.desktops.desktop.**Desktop**(*data_path*, *node_code*)
Bases: object

An EPIC Desktop Definition

Parameters

- **data_path** – The path to the data to be loaded on the Desktop, formed as an epic url (e.g. "epic://path_to/data")
- **node_code** (*str*) – The node_code of the Desktop Node Type to launch

Variables

- **runtime** (*int*) – The runtime for the Desktop in hours
- **mount_type** (*MountType*) – How should the data folder be mounted to the desktop. Offline takes a copy of the data and will not be automatically synced back to the data store.
- **secure_ip** (*bool, optional*) – Should we restrict which IPs can connect to this node? (defaults to False)
- **ip_address** (*str, optional*) – If secure_ip is True, which IP should connections be restricted to?
- **invoice_reference** (*str, optional*) – Reference string for this desktop to appear on invoices.
- **project_id** (*int, optional*) – ID of the EPIC project to run this job in

get_launch_spec()
Get a Specification for this desktop for launching it

Returns Desktop Specification

Return type class:*epiccore.models.DesktopNodeLaunchSpec*

get_quote_spec()
Get a Specification for this desktop for quotes

Returns Desktop Quote Specification

Return type class:*epiccore.models.DesktopNodeQuote*

class pyepic.desktops.desktop.**MountType**(*value*)
Bases: enum.Enum

How should the data folder be mounted to the desktop. Offline takes a copy of the data and will not be automatically synced back to the data store.

OFFLINE = 'offline'

ONLINE = 'online'

Module contents

5.1.2 Module contents

PYTHON MODULE INDEX

p

INDEX

J

L

M

N

O

P

Q

R

S

T

U

W

Y

Z