aDMIX Documentation

Release 0.2.0

Boris Bauermeister

Oct 03, 2019

Contents

Contents:

CHAPTER 1

aDMIX

Python Boilerplate contains all the boilerplate you need to create a Python package.

- Free software: BSD license
- Documentation: https://advance-data-management-in-xenon.readthedocs.io/en/latest/

1.1 Features

- Improve in Rucio uploads on a multi-scale level

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Required Pre-Installation:

2.1.1 Rucio - Scientific Data Management

The aDMIX tools uses activly the Rucio - Scientific Data Management tool to run uploads to grid locations according to your configuration. Therefore it is mandatory to install the Rucio client in the same Python environment such as aDMIX is installed. Further installations in different environments which requieres side loads (== sourcing from another Anaconda environment) is partially supported by the legacy module (Rucio CLI) but not recommended for further work since it is a) slow and the b) the legacy module may not fully developed.

The Rucio - Scientific Data Management tools is found on Github (https://github.com/rucio/rucio) and further information are given here https://rucio.cern.ch/

2.1.2 gfal

2.2 Stable release

To install aDMIX, run this command in your terminal:

```
$ pip install admix
```

This is the preferred method to install aDMIX, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.3 From sources

The sources for aDMIX can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/XeBoris/admix
```

Or download the tarball:

```
$ curl  -OL https://github.com/XeBoris/admix/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Configuration

The main purpose of aDMIX is to take over the interaction with the data management tool Rucio (source) what is provided by ATLAS. Therefore you need to setup aDMIX with several configuration files which present the data outline of your experiment on local disks and the later data naming convention in Rucio with containers, datasets and files.

This part of the documentation guides you through the necessary configuration.

3.1 aDMIX basic configuration

The basic configuration from an example configuration file is given here in /admix/config/host_config_dummy.config

```
{
    "host": "A short abbreviation of hostname to which individual database data␣
↪locations refer",
    "hostname": "The hostname where aDMIX is executed",
    "log_path": "/path/to/your/logfile.log",
    "type": [], #["datatype1", "datatype2", ..., "datatypeN"], A list of individual␣
↪data types you like to handle in your experiment
    "detector": [], #["tpc", "mv"], Specify detector names. NOT YET SUPPORTED
    "source": [], #["ambe", "none"], Specify a certain sources. Select data by␣
↪sources. NOT YET SUPPORTED
    "template":"/path/to/your/datatype/template/xenon1t_template.config",
    "rucio_backend": "API", # Allow API or CLI, depending what configuration you␣
↪prefer. CLI is legacy
    "rucio_account": "YOUR RUCIO ACCOUNT",
    "rucio_x509": "/path/to/x509/proxy/ticket",
    "rucio_template": "/path/to/Rucio/datatype/template/format.config",
    "rucio_cli": "/path/to/Rucio/CLI/calls/rucio_cli/",
    "database": {
                    "type":"MongoDB",
                    "address": "mongodb://address of your mongoDB (r/w requiered)",
                    "user": "mongodb_user",
```

(continues on next page)

(continued from previous page)

```
                    "password": "mongdb_password",
                    "collection": "xenon1t-runs",
                    "projection": {"_id": true,
                                   "name": true,
                                   "number": true,
                                   "data": true,
                                   "detector": true,
                                   "start":true
                    }
                },
    "sleep_time": 5,
    "experiment": "Xenon1T"
}
```

To begin with you need to setup certain keys beforehand:

- host: A short abbreviation of hostname to which individual database data locations refer later.

  The hostname of your data facility is data-host1.cluster.aws.com. To manage your data locations later in a database or simplicity, you will use an abbreviation for the long hostname (e.g. data-host1):

  data-host1:

  - /data/path/to/dataset/dataset_01
  - /data/path/to/dataset/dataset_02
  - /data/path/to/dataset/dataset_03

- hostname: The hostname of your data facility. For practial reasons this should be same name such it is used in your HOSTNAME bash variable.
- log_path: Specify the path to your log file. All logs will go into one log file. Log file rotation is not yet supported.

CHAPTER 4

Usage

advanced data management in XENON (aDMIX) allows you to run //stand alone// tasks which are decribed below. The purpose of the standalone tasks is to connect Rucio interactions (uploads, downloads and transfers) with the meta database (mongoDB interface). Furthermore, the aDMIX package offers a simple way interact with the Rucio catalogue or with the grid locations directly.

4.1 Standalone Tasks in aDMIX

4.1.1 Upload with MongoDB

Run aDMIX to upload data sets (e.g. XENONnT plugins) into Rucio.

Basic command outline:

```
admix UploadMongoDB --admix-config <CONFIG-FILE>
```

Further comand arguments are:

- '–once': Run the command only once and exit aDMIX
- '–select-run-numbers': Select run numbers according the following scheme. The run number is defined by the MongoDB entry with the field 'number'. Hint: The Separator is '-'. Do not use spaces in between.
  - 00001: Only one run with run number
  - 00001-00002: All run numbers between 1 and (including) 2
- '–select-run-times': Select run times according to the start times of a run. The run time is defined by the MongoDB entry with the field 'start'. A correct definition of the run time is "<Date>_<Time>-<Date>_<Time>". No single time stamps are allowed. Separators are '_' between <Date> and <Time> and '-' between two timestamps.

You can run the aDMIX upload command from any machine where the data are located. The database needs to hold a list of dictionaries and each dictionary presents a data type with location, status, rse (if Rucio) and destinations. An overview example could be:

host	location	status	rse	destination
login	/xenon/xenon1t_processed/pax_v6.6.2/170319_1211.root	transferred	None	None
login	/xenon/xenon1t_processed/pax_v6.6.5/170319_1211.root	transferred	None	None
midway-login1	/project2/lgrandi/xenon1t/processed/pax_v6.8.0/170319_1211.root	transferred	None	None
rucio-catalogue	x1t_SR001_170319_1211_tpc_raw	transferred	[‘CNAF_TAPE_USERDISK:REPLICATING:None’, ‘NIKHEF_USERDISK:OK:None’, ‘UC_OSG_USERDISK:OK:None’]	None
midway-login1	/project/lgrandi/xenon1t/processed/pax_v6.10.1/170319_1211.root	transferred	None	None
dali	/dali/lgrandi/tunnell/strax_data/170319_1211-records-943c2b3857b2a913583557452451b5049362ecbd	transferred	None	**[‘rucio-catalogue:UC_OSG_USER** DISK:None’]

The location must be definied according to the definition in the *template.config* configuration file. An example for an inidivdual plugin looks like:

```
{
"raw_records":{
    "login":"{abs_path}/{number}-{plugin}-{hash}",
    "midway":"{abs_path}/{number}-{plugin}-{hash}",
    "dali":"{abs_path}/{number}-{plugin}-{hash}",
    "xe1t-datamanager":"{abs_path}/{number}-{plugin}-{hash}",
    "eb1":"{abs_path}/{number}-{plugin}-{hash}"
    },
"another_plugin":{
    "login":"{abs_path}/{number}-{plugin}-{hash}",
    },
}
```

The keywords such as abs_path, number,. . . are detemined by aDMIX automatically. The abs_path is detemined from the location where the data are stored at a certain location.

The aDMIX uploader runs continously on the pre-selected run numbers or time stamps and looks for *destinations* which are set for **the same host** whereas aDMIX is running. The destination field is detemined by the destination (rucio-catalogue), the Rucio RSE and a possible lifetime for this upload. You can set several Rucio RSE destinations (strings which are separated by komma) but the first element of the list is the RSE where the data are uploaded the first time.

Once the destination(s) are fullfilled successfully aDMIX the destination field is deleted from the database and the actual location with transferring status is written to the database in the “rse” field. An example is given in the table above.

4.1.2 Update the Run Database (with MongoDB)

Run aDMIX to upload data sets (e.g. XENONnT plugins) into Rucio.

Basic command outline:

```
admix UpdateRunDBMongoDB --admix-config <CONFIG-FILE>
```

Further comand arguments are:

- '–once': Run the command only once and exit aDMIX
- '–select-run-numbers': Select run numbers according the following scheme. The run number is defined by the MongoDB entry with the field 'number'. Hint: The Separator is '-'. Do not use spaces in between.
  - 00001: Only one run with run number
  - 00001-00002: All run numbers between 1 and (including) 2
- '–select-run-times': Select run times according to the start times of a run. The run time is defined by the MongoDB entry with the field 'start'. A correct definition of the run time is "<Date>_<Time>-<Date>_<Time>". No single time stamps are allowed. Separators are '_' between <Date> and <Time> and '-' between two timestamps.

Since several transfers within the Rucio catalogue are ongoing (see :_table1: for plugin 'raw' in column rse) we need to update the experiment database from time to time with the latest locations from Rucio. Run this command continously on *any location* with an installed Rucio catalogue.

Attention: Due to deletion processes for Rucio transfer rules in the Rucio catalogue it might be possible that a certain dataset *does not* have any Rucio transfer rule. In this situation, the command set the status of the according Rucio database entry (rucio-catalogue) to *RucioClearance*. This status acts a as a pre-stage to remove the whole rucio-catalogue entry for the given plugin type from the database with the *ClearTransfersMongoDB* option.

4.1.3 Init Rucio Transfers (with MongoDB)

Run aDMIX to upload data sets (e.g. XENONnT plugins) into Rucio.

Basic command outline:

```
admix InitTransfersMongoDB --admix-config <CONFIG-FILE>
```

Further comand arguments are:

- '–once': Run the command only once and exit aDMIX
- '–select-run-numbers': Select run numbers according the following scheme. The run number is defined by the MongoDB entry with the field 'number'. Hint: The Separator is '-'. Do not use spaces in between.
  - 00001: Only one run with run number
  - 00001-00002: All run numbers between 1 and (including) 2
- '–select-run-times': Select run times according to the start times of a run. The run time is defined by the MongoDB entry with the field 'start'. A correct definition of the run time is "<Date>_<Time>-<Date>_<Time>". No single time stamps are allowed. Separators are '_' between <Date> and <Time> and '-' between two timestamps.

aDMIX is able to fetch *new* destinations for a given rucio-catalogue entry and plugin type. These destinations are defined similar to upload destinations. It is a list of strings:

```
destination = ['rucio-catalogue:UC_DALI_USERDISK:None',
               'rucio-catalogue:NIKHEF_USERDISK:86400']
```

You can set up the database entries from any location and run the aDMIX instance from any location with a pre-installed Rucio software package. aDMIX will fullfill all demanded destinations for the Rucio transfer rules.

Attention

- Each rule can be initialized with a lifetime (third argument). This lifetime is given in seconds (always). You are able to extend the lifetime at any point as long as there is a rule existing in the Rucio catalogue.

- You can use the lifetime to **purge** data from the Rucio catalogue. If the lifetime is set to 10 seconds, Rucio will remove the transfer rule after 10 seconds automatically and the Rucio services in the background will start to purge data from the according RSE. Be aware that Rucio services crash sometimes in the background. If data do not disappear automatically you need to check manually for it.

4.1.4 Clear Rucio Information from Run Database (with MongoDB)

Basic command outline:

```
admix ClearTransfersMongoDB --admix-config <CONFIG-FILE>
```

Further comand arguments are:

- '–once': Run the command only once and exit aDMIX
- '–select-run-numbers': Select run numbers according the following scheme. The run number is defined by the MongoDB entry with the field 'number'. Hint: The Separator is '-'. Do not use spaces in between.
  - 00001: Only one run with run number
  - 00001-00002: All run numbers between 1 and (including) 2
- '–select-run-times': Select run times according to the start times of a run. The run time is defined by the MongoDB entry with the field 'start'. A correct definition of the run time is "<Date>_<Time>-<Date>_<Time>". No single time stamps are allowed. Separators are '_' between <Date> and <Time> and '-' between two timestamps.

This command clears the database entries for the host rucio-catalogue when the status is set to *RucioClearance*. You can do this manually or it is set to *RucioClearance* by the UpdateRunDBMongoDB command of aDMIX. You can run this command from any location.

Attention

- No cross check for the number of locations! Keep this in mind in case you fear Rucio-database issues. Run a manual cross check before to avoid data loss from the database.

4.1.5 Purge Physical Data Locations on Disks (with MongoDB)

This module allows you to purge data from physical data disks in a safe way. As an outcome, the physical disk location are de-registered from the meta database.

You can run this command only at locations where do you intend to purge data. In that sense it becomes also important to specify host location in your configuration file to avoid uncertain host conditions.

This command requests by default a two fold data existence before purging data from a physical disk. This assures that no datasets are deleted at the Rucio entry point before there are enough copies of the data spread. The two fold requirement is defined as:

- Two copies in Rucio which are in replication status OK and marked in the meta database with "transferred"
- One copy in Rucio with replication status OK and marked in meta database with "transferred". In addition one ore more disk copies at several sites. Disk locations are determined by database only.

The '–force' command can be used to enable a manual mode to purge data on disks which do not fulfil the minimum data safety requirement. Each dataset must be confirmed with 'yes'.

Basic command outline:

```
admix PurgeMongoDB --admix-config <CONFIG-FILE>
```

Due to the complex file structure our data products the purge command became quite extensive in terms of selections to narrow down the datasets (depending on type, version (hash), host and location). The supported terminal arguments are:

- '–once': Run the command only once and exit aDMIX
- '–select-run-numbers': Select run numbers according the following scheme. The run number is defined by the MongoDB entry with the field 'number'. Hint: The Separator is '-'. Do not use spaces in between.
  - 00001: Only one run with run number
  - 00001-00002: All run numbers between 1 and (including) 2
- '–select-run-times': Select run times according to the start times of a run. The run time is defined by the MongoDB entry with the field 'start'. A correct definition of the run time is "<Date>_<Time>-<Date>_<Time>". No single time stamps are allowed. Separators are '_' between <Date> and <Time> and '-' between two timestamps.
- '–type': Define a specific data product for purging (e.g. raw_records). The input allows multiple arguments but that specific application (PurgeMongoDB) makes only use of one argument at one time.
- '–hash': The hash sum is part of the data location and refers to a specific version of the chosen data product type. Choose the hash from the meta database in advance.
- '–force': Enforces a user to purge datasets which are prevented from purging. (Not enough copies in Rucio or other disks)

4.2 aDMIX as a Module

4.2.1 Create a Rucio Template Dictionary in aDMIX

A Rucio template dictionary is defined in aDMIX as a dictionary with the following (example) structure:

```
{
    "L0": {
        "type": "rucio_container",
        "did": "x1t_{science_run}:x1t_{science_run}_data",
        "tag_words":["science_run"],
        },
    "L1": {
        "type": "rucio_container",
        "did": "x1t_{science_run}:x1t_{date}_{time}_{detector}",
        "tag_words": ["science_run", "date", "time", "detector"],
        },
    "L2": {
        "type": "rucio_dataset",
        "did": "x1t_{date}_{time}_{detector}:{plugin}-{hash}",
        "tag_words": ["date", "time", "detector", "plugin", "hash"],
        }
}
```

aDMIX is shipped out with two modules which help you create this structure: *templater* and *keyword*. The aim of the *templater* module is to load a specific Rucio data structure from a configuration file. This helps you to provide several Rucio configurations for different experimental setups and allow you create automatically a complex Rucio structure, such as a dataset which is attached to container.

Once the Rucio configuration file is loaded from a template file, the *keyword* method is able to create the complex nested structure (goes by the definition of levels to identify what is attached to what) and provides empty keywords which need to be filled. Once the *keyword* method has filled the template dictionary completely it is ready to use.

The following example shows how to load a Rucio structure template and fill it with keywords. The *keyword* method receives simply a dictionary with all requested keywords from Ruico template dictionary.

To begin with, a Rucio structure template for XENON1T looks like this:

```
{
    "raw":"$Cx1t_SR{science_run}:xe1t_SR{science_run}_data|->|$Cx1t_SR{science_run}
↪:x1t_SR{science_run}_{date}_{time}_{detector}|->|$Dx1t_SR{science_run}_{date}_{time}
↪_{detector}:raw",
    "processed":"$Cx1t_SR{science_run}:x1t_SR{science_run}_data|->|$Cx1t_SR{science_
↪run}:x1t_SR{science_run}_{date}_{time}_{detector}|->|$Dx1t_SR{science_run}_{date}_
↪{time}_{detector}:processed",
}
```

The entries "raw" and "processed" defining the plugin type (e.g. "raw_records" in XENONnT) and each string afterwards describe a complex Rucio data sturucture which is used to sort data into the Rucio catalogue.

For example we have:

- $Cx1t_SR{science_run}:xe1t_SR{science_run}_data: It defines by default a Rucio container (introduced by $C at the begin of the string).
- Another container ($Cx1t_SR{science_run}:x1t_SR{science_run}_{date}_{time}_{detector}) is attached to the top level container $Cx1t_SR{science_run}:xe1t_SR{science_run}_data. This is introduced by the arrow feature ("|->|").
- Finally we have another Rucio dataset attached ($Dx1t_SR{science_run}_{date}_{time}_{detector}:raw). A Rucio dataset is introduced by "$D" at the begin.

Have in mind that each level is defined by a Rucio data identifier which consist of a scope and name (scope:name) which are separated by a ':' character. The lowest structure (Rucio dataset) will receive the files during the upload process later). Each Rucio structure template contains keywords which ({date} or {science_run}). We are going to fill these keywords later by the *keyword* method:

A full code example for XENONnT is given here:

```
from admix.interfaces.rucio_dataformat import ConfigRucioDataFormat
from admix.interfaces.keyword import Keyword
from admix.interfaces.templater import Templater

#Init the method to load a specific Rucio template configuration file:

path_to_your_rucio_configuration_file = "/.../.."

rc_reader = ConfigRucioDataFormat()
rc_reader.Config(path_to_your_rucio_configuration_file)

#Receive the empty plugin structure from the configuration file:
plugin_type = "raw_records"
rucio_template = rc_reader.GetPlugin(plugin_type)


#Init the keyword method
keyw = Keyword()

#Prepare the keyword method to fill the keywords from the template:

rucio_in = "x1t_SR001_170319_1011_tpc:raw_records-58340a130"

db = {}
```

(continues on next page)

(continued from previous page)

```
db['plugin']    = rucio_in.split(":")[1].split("-")[0]
db['date']      = rucio_in.split(":")[0].split("_")[2]
db['time']      = rucio_in.split(":")[0].split("_")[3]
db['detector'] = rucio_in.split(":")[0].split("_")[4]
db['hash']      = rucio_in.split(":")[1].split("-")[1]
db['science_run'] = rucio_in.split(":")[0].split("_")[1].replace("SR", "")

keyw.SetTemplate(db)
#(we assume here that the dictionary db is filed from a string. But it can come from␣
↪any location (e.g. database, textfile)!

#Complete the Rucio template:
rucio_template = keyw.CompleteTemplate(rucio_template)
```

The variable rucio_template holds the desired complex Rucio structure for a given plugin type.

4.2.2 Download a Rucio Data Identifier (DID)

How to download a given Rucio DID which is defined as “x1t_SR001_171230_1818_tpc:raw_records-7k65yaooed”?

```
#imports:
from admix.interfaces.rucio_dataformat import ConfigRucioDataFormat
from admix.interfaces.rucio_summoner import RucioSummoner
from admix.interfaces.destination import Destination
from admix.interfaces.keyword import Keyword
from admix.interfaces.templater import Templater

#Be aware of the template files and the config "fake" setup":
config_file = "/home/bauermeister/Development/software/admix_config/host_config_login_
↪el7_api.config"

#Load your config file
config = load_config(config_file)

#Set up the RucioSummoner: You could also fill it manually!
rc = RucioSummoner(config.get("rucio_backend"))
rc.SetRucioAccount(config.get('rucio_account'))
rc.SetConfigPath(config.get("rucio_cli"))
rc.SetProxyTicket(config.get('rucio_x509'))
rc.SetHost(config.get('host'))
rc.ConfigHost()

#Most likely you are getting the run locations for a type
did = "x1t_SR001_171230_1818_tpc:raw_records-7k65yaooed"

#Extract scope and name:
scope = did.split(":")[0]
dname = did.split(":")[1]

result = rc.Download(download_structure=did,
                     download_path="/your/download/path/",
                     rse="YOUR_DOWNLOAD_RSE",
                     no_subdir=False #if true, the DID name is not used in the␣
↪download path
                     )
```

(continues on next page)

(continued from previous page)

```
print(result)
```

4.2.3 Download Single Chunks from a Rucio Data Identifier (DID)

How to download three chunks from a given Rucio DID which is defined as “x1t_SR001_171230_1818_tpc:raw_records-7k65yaooed”?

Let’s assume the three chunks are ‘00001’, ‘00002’ and ‘00003’. Based on the general Strax file definition, you can also request the ‘metadata.json’ as a chunk.

```
#imports:
from admix.interfaces.rucio_dataformat import ConfigRucioDataFormat
from admix.interfaces.rucio_summoner import RucioSummoner
from admix.interfaces.destination import Destination
from admix.interfaces.keyword import Keyword
from admix.interfaces.templater import Templater

#Be aware of the template files and the config "fake" setup":
config_file = "/home/bauermeister/Development/software/admix_config/host_config_login_
↪el7_api.config"

#Load your config file
config = load_config(config_file)

#Set up the RucioSummoner: You could also fill it manually!
rc = RucioSummoner(config.get("rucio_backend"))
rc.SetRucioAccount(config.get('rucio_account'))
rc.SetConfigPath(config.get("rucio_cli"))
rc.SetProxyTicket(config.get('rucio_x509'))
rc.SetHost(config.get('host'))
rc.ConfigHost()

#Most likely you are getting the run locations for a type
did = "x1t_SR001_171230_1818_tpc:raw_records-7k65yaooed"

#Extract scope and name:
scope = did.split(":")[0]
dname = did.split(":")[1]

#Create a list of three chunks:
chunks = [ str(i).zfill(6) for i in range(0, 3)]
download_path = "/your/download/path/"
rse = "YOUR_DOWNLOAD_RSE"
no_subdir = False #if true, the DID name is not used in the download path

result = rc.DownloadChunks(download_structure=did,
                           chunks=chunks,
                           download_path=download_path,
                           rse=rse,
                           no_subdir=no_subdir)

#result is a list of dictionaries:
for i_result i result:
    print(i_result)
```

4.2.4 Download from Rucio with a Template Dictionary (with chunks)

In additon to the Rucio downloads with a DID, aDMIX supports a Rucio template dictionary download too. It is important to notice that by default only the lowest level Rucio dataset is downloaded. It is possible to adjust it by specifing the level manually when calling the Download(…) function of aDMIX. Be aware that due to a complex Rucio structure, the download volume can be increased tremendously.

```
#imports:
from admix.interfaces.rucio_dataformat import ConfigRucioDataFormat
from admix.interfaces.rucio_summoner import RucioSummoner
from admix.interfaces.destination import Destination
from admix.interfaces.keyword import Keyword
from admix.interfaces.templater import Templater

#Be aware of the template files and the config "fake" setup":
config_file = "/home/bauermeister/Development/software/admix_config/host_config_login_
↪el7_api.config"

#Load your config file
config = load_config(config_file)

#Set up the RucioSummoner: You could also fill it manually!
rc = RucioSummoner(config.get("rucio_backend"))
rc.SetRucioAccount(config.get('rucio_account'))
rc.SetConfigPath(config.get("rucio_cli"))
rc.SetProxyTicket(config.get('rucio_x509'))
rc.SetHost(config.get('host'))
rc.ConfigHost()


...CODE TO CREATE/LOAD/USE A RUCIO TEMPLATE DICTIONARY...

rucio_template = keyw.CompleteTemplate(rucio_template)


result = rc.Download(download_structure=rucio_template,
                     download_path="/your/download/path/",
                     rse="YOUR_DOWNLOAD_RSE",
                     no_subdir=False, #if true, the DID name is not used in the␣
↪download path
                     level=-1 #Select level by int (e.g. 2) to download the top-level␣
↪Rucio container (see Rucio template description)
                     )
print(result)
```

In addition, it is possible to modify the Download command for chunks again:

```
chunks = [ str(i).zfill(6) for i in range(0, 3)]
result = rc.DownloadChunks(download_structure=rucio_template,
                           chunks=chunks,
                           download_path=download_path,
                           rse=rse,
                           no_subdir=no_subdir)
```

4.2.5 Upload with a Template Dictionary

Uploads are made similar to the downloads and need a Rucio template dictionary if it is wished to upload data into a complex Rucio structure. Once the Rucio template dictionary is created you need to provide a data location and an initial RSE (with lifetime if needed). The location of the data does only need to contain individual files under folder. The common Rucio scope is determined from the Rucio template dictionary for the lowest level if not specified otherwise.

Of course you can also use UploadToDid(…) or UploadToScope(…) from the RucioSummoner to upload data to Rucio. These functions do not offer the abilty to build a complex Rucio structure beforehand.

```
#imports:
from admix.interfaces.rucio_dataformat import ConfigRucioDataFormat
from admix.interfaces.rucio_summoner import RucioSummoner
from admix.interfaces.destination import Destination
from admix.interfaces.keyword import Keyword
from admix.interfaces.templater import Templater


#Be aware of the template files and the config "fake" setup":
config_file = "/home/bauermeister/Development/software/admix_config/host_config_login_
↪el7_api.config"

#Load your config file
config = load_config(config_file)

#Set up the RucioSummoner: You could also fill it manually!
rc = RucioSummoner(config.get("rucio_backend"))
rc.SetRucioAccount(config.get('rucio_account'))
rc.SetConfigPath(config.get("rucio_cli"))
rc.SetProxyTicket(config.get('rucio_x509'))
rc.SetHost(config.get('host'))
rc.ConfigHost()

#prepare a rucio_template as described above:
rucio_template = ...


upload_result = self.rc.Upload(upload_structure=rucio_template,
                               upload_path=origin_location, #A valid path with data␣
↪to upload
                               rse="INITIAL_RSE_UPLOAD",
                               rse_lifetime=None, #Or lifetime in seconds
                               )
print(upload_result) # 0 if successful, 1 if failed
```

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/XeBoris/admix/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

aDMIX could always use more documentation, whether as part of the official aDMIX docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/XeBoris/admix/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *admix* for local development.

1. Fork the *admix* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/admix.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv admix
$ cd admix/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 admix tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/XeBoris/admix/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_admix
```

CHAPTER 6

Credits

6.1 Development Lead

- Boris Bauermeister <Boris.Bauermeister@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.1.0 (2018-02-05)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search