Rucio Documentation

Release 1.10.4-2-g618de35-dev1490693807

Vincent Garonne

August 10, 2017

Contents

Attention: ATLAS users should refer to https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/RucioClientsHowTo **instead of this page. This twiki is the entry point for ATLAS users learning the Rucio client. The ATLAS-specific information on this page will be removed.**

The Rucio project is the new version of ATLAS Distributed Data Management (DDM) system services for allowing the ATLAS collaboration to manage the large volumes of data, both taken by the detector as well as generated or derived, in the ATLAS distributed computing system. Rucio uses to manage accounts, files, datasets and distibuted storage systems.

This documentation is generated by the Sphinx toolkit. and lives in the source tree.

Links:

- Documentation - http://rucio.cern.ch
- Project tracker - https://its.cern.ch/jira/browse/RUCIO
- Code analytics and search services - https://www.ohloh.net/p/rucio/

CHAPTER 1

Concepts and terminology

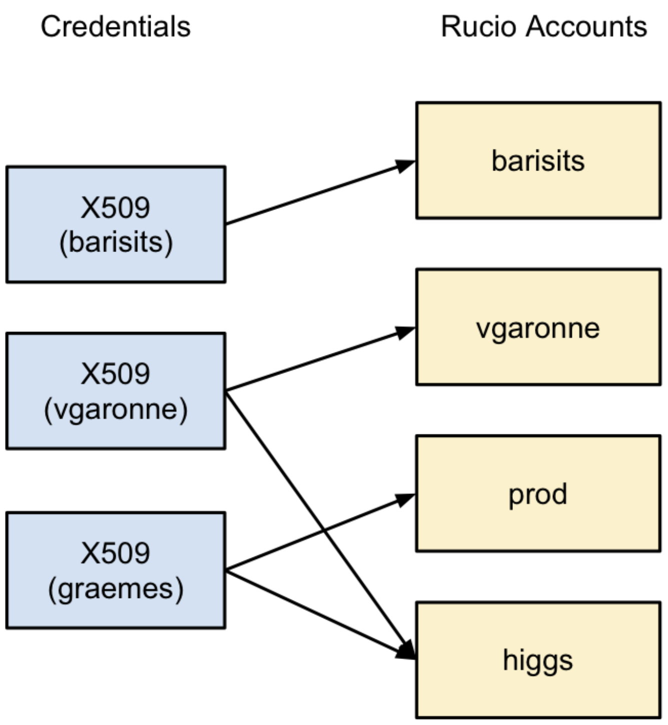
The following concepts define how data is organised, accessed, and catalogued by Rucio.

Rucio account

A Rucio account is the unit of assigning privileges in Rucio. It can represent individual users (like lgoossen, graemes, vgaronne, ...), a group of users (like bphys, higgs, susy, ...) or an organised production activity for the whole ATLAS collaboration (prod, tzero, ...). A Rucio account is identified by a string.

Rucio actions are always conducted by a Rucio account. Each account has a namespace identifier called scope that is included in every name assigned to a collection of data created by that account (see S 3.1). By default, Rucio accounts can only create identifiers in their own scope and not in any other.

A Rucio user is identified by his credentials, like X509 certificates, username/password, or token. Credentials can map to one or more accounts (N:M mapping). The Rucio authentication system checks if the used credentials are authorized to use the supplied Rucio account. The figure below gives an example of the mapping between credentials and Rucio accounts:



Files, Datasets and Containers

ATLAS has a large amount of data, which is physically stored in files. For the data management system these files are the smallest operational unit of data (so sub-file operations are not possible). Physicists need to be able to identify and operate on any arbitrary set of files.

Files can be grouped into datasets (a named set of files) and datasets can be grouped into containers (a named set of datasets or, recursively, containers). All three types of names refer to data so the term 'data identifier set' (DIS) is used to mean any set of file, dataset or container identifiers. A data identifier is just the name of a single file, dataset or container.

The figure below gives an example of an aggregation hierarchy:

Data identifiers and scope

Files, datasets and containers follow an identical naming scheme which is composed of two strings: the scope and a name. The combination of both is called a data identifier (DI). For instance a file identifier (LFN) is composed of a scope and a file name.

The scope string partitions the name space in several sub spaces. The primary use case for this is to have separate scopes for production and individual users.

By default accounts will have read access to all scopes and write access only to their own scope. Privileged accounts will have write access to multiple scopes, e.g., production might use scopes such as mc11, data12_8TeV, tmp.prod.#

Files, datasets and containers are uniquely identified over all time. This implies that an identifier, once used, can never be reused to refer to anything else at all, not even if the data it referred to has been deleted from the system.

File, dataset and container status

File status

The following status attributes are supported for files:

- `availability`: LOST/DELETED/AVAILABLE

A file is LOST if there are no known replicas of the file in Rucio, while at the same time at least one account requested a replica; a file is DELETED if no account requested a replica; otherwise the file is AVAILABLE. This is a derived attribute.

- `suppressed`: True/False

This is a user settable flag. It indicates that the owner of the scope no longer needs the name to be present in the scope. Files that are suppressed (by default) do not show up in search and list operations on the scope. The setting of this flag is subject to conditions, e.g., one can not suppress a file while at the same time requesting it to be replicated somewhere.

Note however that this flag will be ignored when explicitly listing contents of datasets/containers.

Dataset/Container status

The dataset/container status is reflected by a set of attributes:

- `open`: True/False

If a dataset/container is open, content can be added to it. Datasets/Containers are created open and once closed, they cannot be opened again.[2]_.

- `monotonic`: True/False

If the monotonic attribute is set, content cannot be removed from an open dataset/container. Datasets/Containers are, by default, created non-monotonic. Once set to monotonic, this cannot be reversed.

- `complete`: True/False

A dataset/container where all files have replicas available is complete. Any dataset/container which contains files without replicas is incomplete. This is a derived attribute.

Meta-data attributes

Meta-data associated with a dataset/file is represented using attribute/value pairs. The set of available attributes is restricted. Meta-data attributes are classified into four categories:

- `System-defined attributes`: e.g. size, checksum, creationtime, modificationtime, status
- `Physics attributes`: e.g. like number of events, cross-section, or GUID
- `Production attributes`: storing information like which task or job produced the file
- `Data management attributes`: necessary for the organisation of data on the grid (see Replica Management section)

For datasets, it is possible that the value of a meta-data attribute is a function of the meta-data of its constituents, e.g. the total size is the sum of the sizes of the constituents. In this case it is obviously not possible to assign a value to it.

Rucio Storage Element

A Rucio Storage Element (RSE) is a container for physical files. It is the smallest unit of storage space addressable within Rucio. It has an unique identifier and a set of meta attributes describing properties such as supported protocols, e.g., file, https, srm; host/port address; quality of service; storage type, e.g., disk, tape, ...; physical space properties, e.g., used, available, non-pledged; and geographical zone.

Rucio Storage Elements can be grouped in many logical ways, e.g., the UK RSEs, the Tier-1 RSEs, or the 'good' RSEs. One can reference groups of RSEs by metadata attributes or by explicit enumeration of RSEs.

RSE tags are expanded at transfer time to enumerate target sites. Post-facto changes to the sites in an RSE tag list will not affect currently replicated files.

A cache is storage service which keeps additional copies of files to reduce response time and bandwidth usage. In Rucio, a cache is an RSE, tagged as volatile. The control of the cache content is usually handled by an external process or applications (e.g. Panda) and not by Rucio. Thus, as Rucio doesn't control all file movements on these RSEs, the application populating the cache must register and unregister these file replicas in Rucio. The information about replica location on volatile RSEs can have a lifetime. Replicas registered on volatile RSEs are excluded from the Rucio replica management system (replication rules, quota, replication locks) described in the section Replica management. Explicit transfer requests can be made to Rucio in order to populate the cache.

Permission model

Rucio assigns permissions to accounts. Permissions are boolean flags designating whether an account may perform a certain action (read, write, delete) on a resource (RSE, account, replica, etc.).

Replica management

General Overview

Replica management is based on replication rules defined on logical files. A replication rule is owned by an account and defines the minimum number of replicas to be available on a list of RSEs. Accounts are allowed to set multiple rules [1]. Rules may optionally have a limited lifetime and can be added, removed or modified at any time.

An example listing of replication rules is given below:

- prod: 1x replica @ CERN, no lifetime
- barisits: 1x replica @ US-T2, until 2012-01-01
- vgaronne: 2x replica @ T1, no lifetime

A rule engine validates the rules and creates transfer primitives to fulfil all rules, e.g. transfer a file from RSE A to RSE B. The rule engine is triggered when a file is created in the system, when a new rule is added to a file or when one explicitly requests for the rule to be applied on existing data. The rule engine will only create the minimum set of necessary transfer primitives to satisfy all rules.

An account can inject transfer primitives directly, e.g., transient replicas required for production operations. Notifications can be provided for the transfer request. All transfer requests are transient.

Deletion is triggered per RSE when storage policy dictates that space must be freed. A reaper service will look for replicas on that RSE which can be deleted without violating any replication rules. The reaper will use a Least Recently

[1] The system may reject rules if these violate other policies, e.g., a normal ATLAS user would not be allowed to set a rule which committed the system to generate 5PB of new replicas or to request replicas on an RSE tape system.

Used (LRU) algorithm to select replicas for deletion. The reaper service will also immediately delete all replicas of any file which is declared obsolete.

Replication rule examples

Replica management is based on replication rules defined on data identifiers. A replication rule gets resolved and issues replica locks on the physical replicas.

A replication rule consists (besides other parameters) of a factor representing the numbers of replicas wanted and a Rucio Storage Element Expression that allows to select a set of probable RSEs to store the replicas.

The RSE Expression gets resolved into a set of RSEs, which are possible destination RSEs for the number of replicas the user wants to create.

Is possible to find detailed information and examples about how to write RSE Expressions here.

Example 1

I want to have 2 replicas of first_dataset and second_datset on Tier 1 RSEs

The number 2 *second_dataset* is the number of copies expected. At the end, the RSE Expression select all the Tier 1 RSEs as possible targets to store the replicas.:

```
username@host:~$ rucio add-rule scope:first_dataset scope:second_dataset 2 'tier=1'
```

To see all the possible targets, **rucio list-rses** command can be used:

```
username@host:~$ rucio list-rses --expression 'tier=1'
```

Example 2

I want to have 2 replicas on whatever T2 RSEs in the UK but it shouldn't be Glasgow:

```
username@host:~$ rucio add-rule scope:first_dataset scope:second_dataset 2 'country=uk\site=GLASGOW'
```

RSE Expressions

A **RSE Expression** allows to select a set of RSEs to create replication rules. The RSE Expression consists of one or more **terms**. A term can be a single RSE name or a condition over the RSE attributes. The RSE Expression Parser resolves each term to a set of RSEs. The resulting set of a term will be all those RSEs match the attribute or name. Terms can be connected by **operators** to form more complex expressions. For example, users can write RSE expressions to address all Tier 2 RSEs, all the RSEs in certain cloud, all Tier 2 RSEs not in certain clouds, etc.

Simple RSE Expressions

Rucio allows to test RSE Expressions, using the command **list-rses**. The most simple RSE Expression is the one containing the name of a particular RSE.

1. The following expression only return a set containing a single RSE:

```
    jbogadog@lxplus0058:~$ rucio list-rses --expression EELA-UNLP_SCRATCHDISK
    EELA-UNLP_SCRATCHDISK
```

2. Another simple RSE Expression allows to list the set of all the RSEs in a particular site:

```
    jbogadog@lxplus0058:~$ rucio list-rses --expression site=EELA-UNLP
    EELA-UNLP_PRODDISK
    EELA-UNLP_DATADISK
    EELA-UNLP_SCRATCHDISK
```

3. Or all the RSEs who's type is SCRATCHDISK:

```
    jbogadog@lxplus0058:~$ rucio list-rses --expression type=SCRATCHDISK
    UNI-SIEGEN-HEP_SCRATCHDISK
    NCG-INGRID-PT_SCRATCHDISK
    EELA-UNLP_SCRATCHDISK
    ...
    INFN-T1_SCRATCHDISK
    FMPHI-UNIBA_SCRATCHDISK
    INFN-FRASCATI_SCRATCHDISK
```

4. Or all the Spanish sites:

```
    jbogadog@lxplus0058:~$ rucio list-rses --expression SPAINSITES
    IFIC-LCG2_LOCALGROUPDISK
    IFAE_PRODDISK
    PIC_SCRATCHDISK
    EELA-UNLP_SCRATCHDISK
    ...
    EELA-UNLP_DATADISK
    UAM-LCG2_SCRATCHDISK
    IFIC-LCG2_DATADISK
    LIP-COIMBRA_LOCALGROUPDISK
```

Note that if the RSE Expresion returns an empty set, rucio returns an error. This could be because the name of the attribute doesn't exist or because there's no RSE that match the expression. It does not necessarily mean that the syntax of the expression is wrong.

In 2) and 3), the RSE Expression refers to an attribute in the RSE that must be equal to a given value to match the expression. While in 1) and 4), the expression match a RSE if the attribute is True. It is possible to see the list of attributes for a particular RSE with rucio:

```
jbogadog@lxplus0100:~$ rucio list-rse-attributes EELA-UNLP_SCRATCHDISK
  ftstesting: https://fts3-pilot.cern.ch:8446
  ALL: True
  ESTIER2S: True
  physgroup: None
  spacetoken: ATLASSCRATCHDISK
  fts: https://fts3.cern.ch:8446,https://lcgfts3.gridpp.rl.ac.uk:8446,https://fts.usatlas.bnl.gov:84
  site: EELA-UNLP
  EELA-UNLP_SCRATCHDISK: True
  datapolicyt0disk: False
  cloud: ES
  SPAINSITES: True
  datapolicyt0taskoutput: False
  fts_testing: https://fts3-pilot.cern.ch:8446
  tier: 3
  datapolicyt0tape: False
  type: SCRATCHDISK
  istape: False
```

Most of the RSEs share the same set of attributes, and is possible to create RSE Expressions based on all of them.

Operators

Operators are used to connect terms in order to get more complex RSE Expressions/terms. The syntactic functionality of the Rucio RSE Expressions Parser allows the basic operations defined in mathematical set theory, Union, Intersection and Complement. Using an operator on two sets of RSEs will construct a new set based on the given sets.

The symbols **A** and **B** in this table stand for a term.

Operator	Meaning	Interpretation	Example
A|B	UNION	A union B	EELA-UNLP_SCRATCHDISK | EELA-UNLP_PRODDISK
A&B	INTERSECT	A intersect B	tier=1&country=us
A\B	COMPLEMENT	A complement B	cloud=ES\type=SCRATCHDISK

Composing RSE Expressions

Using the operators described above, it's possible to create expressions to select whatever RSE you need to put your data in. Use the following list of examples to build your own RSE Expressions.

All Tier 2 sites in DE cloud:

```
jbogadog@lxplus0100:~$ rucio list-rses --expression 'tier=2&cloud=DE'
PRAGUELCG2_PPSLOCALGROUPDISK
FMPHI-UNIBA_LOCALGROUPDISK
...
UNI-FREIBURG_DATADISK
DESY-HH_PRODDISK
```

Note the use of the single quotes. Single quotes are needed to avoid the shell interpret the **&**, the **|** or the **** as commands.

All tier 1 but not the ones in country=us:

```
jbogadog@lxplus0100:~$ rucio list-rses --expression 'tier=1\country=us'
INFN-T1_MCTAPE
BNL-OSG2_DATATAPE
...
BNL-OSG2_DDMTEST
NIKHEF-ELPROD_PHYS-SUSY
```

However, take care of the subtle differences. While the first expression exclude United States' sites, the second doesn't:

```
jbogadog@lxplus0100:~$ rucio list-rses --expression 'tier=1\country=us'|wc -l
115
jbogadog@lxplus0100:~$ rucio list-rses --expression 'tier=1\country=US'|wc -l
117
```

The filters are processed from left to right. Is possible to use parenthesis to force the order of operation. See the following example to get all the SCRATCHDISKs in IT or FR clouds:

```
jbogadog@lxplus0100:~$ rucio list-rses --expression 'cloud=IT|cloud=FR&type=SCRATCHDISK'|wc -l
30
jbogadog@lxplus0100:~$ rucio list-rses --expression '(cloud=IT|cloud=FR)&type=SCRATCHDISK'|wc -l
```

```
30
jbogadog@lxplus0100:~$ rucio list-rses --expression 'type=SCRATCHDISK&(cloud=IT|cloud=FR)'|wc -l
30
jbogadog@lxplus0100:~$ rucio list-rses --expression 'type=SCRATCHDISK&cloud=IT|cloud=FR'|wc -l
92
```

While the first three operations are equivalent, the last return sites in cloud FR but not only the SCRATCHDISKs but the GROUPDISKs and DATADISKs too, among other types.

Accounting and quota

Accounting is the measure of how much resource, e.g. storage, an account has used as a consequence of its actions. Quota is a policy limit which the system applies to an account.

For storage accounting, Rucio accounts will only be accounted for the files they set replication rules on. The accounting is based on the replicas an account requested, not on the actual amount of physical replicas in the system.

Notifications

External applications can require synchronisation on events relative to data availability and can subscribe to particular events, e.g., dataset state changes. Rucio will then publish a message to external application when it detects these events.

Subscriptions / Policies

Policies are system entities which generate rules or transfer requests based on matching particular dataset metadata at registration time. Polices are owned by an account and can only generate rules for that account. Policies may have a lifetime, after which they will expire.

An example of a policy is given below:

Attribute	Value
Owner	tzero
match	project=data11 7TeV, dataType=RAW, stream=physics*
rule	1@CERNTAPE, 1@T1TAPE
lifetime	2012-01-01 00:00

Policies can also create transfer primitives, so generate extra copies of data as it is produced:

Attribute	Value
Owner	prod
match	project=mc11 7TeV, dataType=merge.AOD, tag=*(p795|p796|p805)*, replicationPolicy=RPValue
rule	1@T1DISK, 1@T2DISK,
transfer	1@T1DISK, 2@T2DISK
lifetime	2011-12-01 00:00

In this case the transfer request is for extra copies, in addition to those set by rules. (This is different behaviour to that for rules themselves, which are always independent.)

Rucio Subscriptions / Policies (name **TBD**) exist for the purpose of making data placement decisions before the actual data has been created. In the current DQ_2 system there are basically 3 applications which are responsible for this behavior:

- DaTRI
- SantaClaus
- AK47

Short application descriptions

DaTRI

See https://twiki.cern.ch/twiki/bin/viewauth/Atlas/DataTransferRequestInterface

DaTRI is a data-transfer tool which basically offers two functionalities:

- Transfers of existing data to a site X
- Transfer-Subscriptions of non-existing data, which match a certain pattern to a site X

Data is identified by dataset-names, patterns, container-names or patterns. Requests can either finish after a transfer is complete or stay active to look for newly-created data (which match a request pattern) to continously transfer the data to a site.

All requests (there are exceptions) have to be approved by either a DaTRI admin or a cloud coordinator.

There also exists a website which shows the status of all requests.

SantaClaus

TODO

AK47

TODO

Workflow

In order to represent all (or most) of this functonality in Rucio in a generic way, the respective steps of the workflow of the current applications have to be identified and described. At the moment we spotted three fundamental steps for all the applications which should be described in the following section.

- **Input Selection**: How is the input data selected? (Patterns on files/ds? etc.)
- **Destination Selection**: How is the destination selected?
- **Output Generation**: What are the characteristics of the output generation (source dataset just moved, subset of dataset moved into new dataset, etc.)?

DaTRI

Input Selection

Based on:

- Pattern: Either a full datasetname, dataset pattern, containername or containerpattern
- Type: Data type of the dataset (e.g.: DESDM_EGAMMA, EVGEN)

Destination Selection

Specific destination endpoint; However, based on the endpoint the request has to be **approved** by an admin or cloud coordinator.

Output Generation

The output generation can be based on:

- Volume: The user selects a percentage of the original data
- Files List: File names or File name patterns
- Both volume and file list

SantaClaus

Input Selection

TODO

Destination Selection

TODO

Output Generation

TODO

AK47

Input Selection

TODO

Destination Selection

TODO

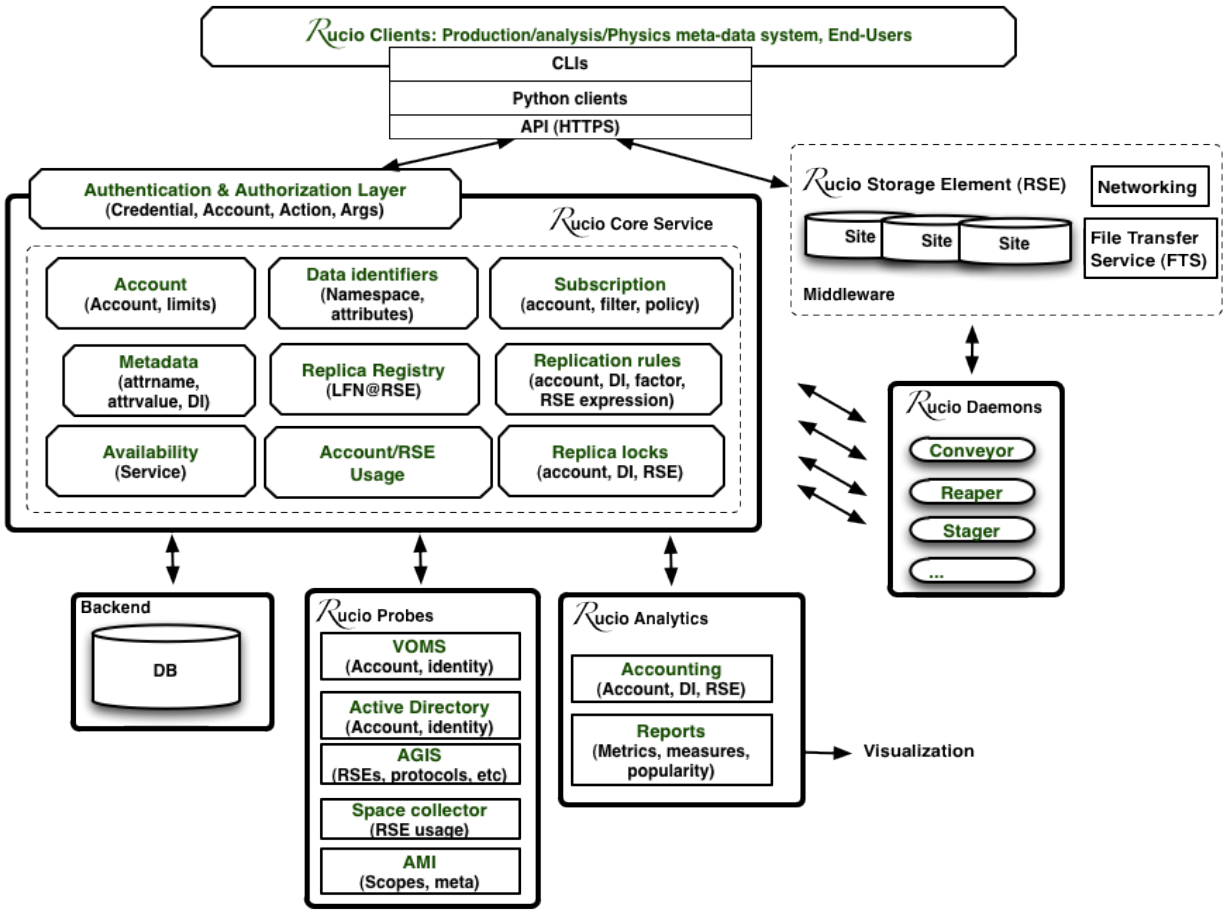
Output Generation

TODO

CHAPTER 2

Architecture

Overview of Rucio Architecture



Rucio Storage Element Wrapper and Manager

DRAFT VERSION: WORK IN PROGRESS

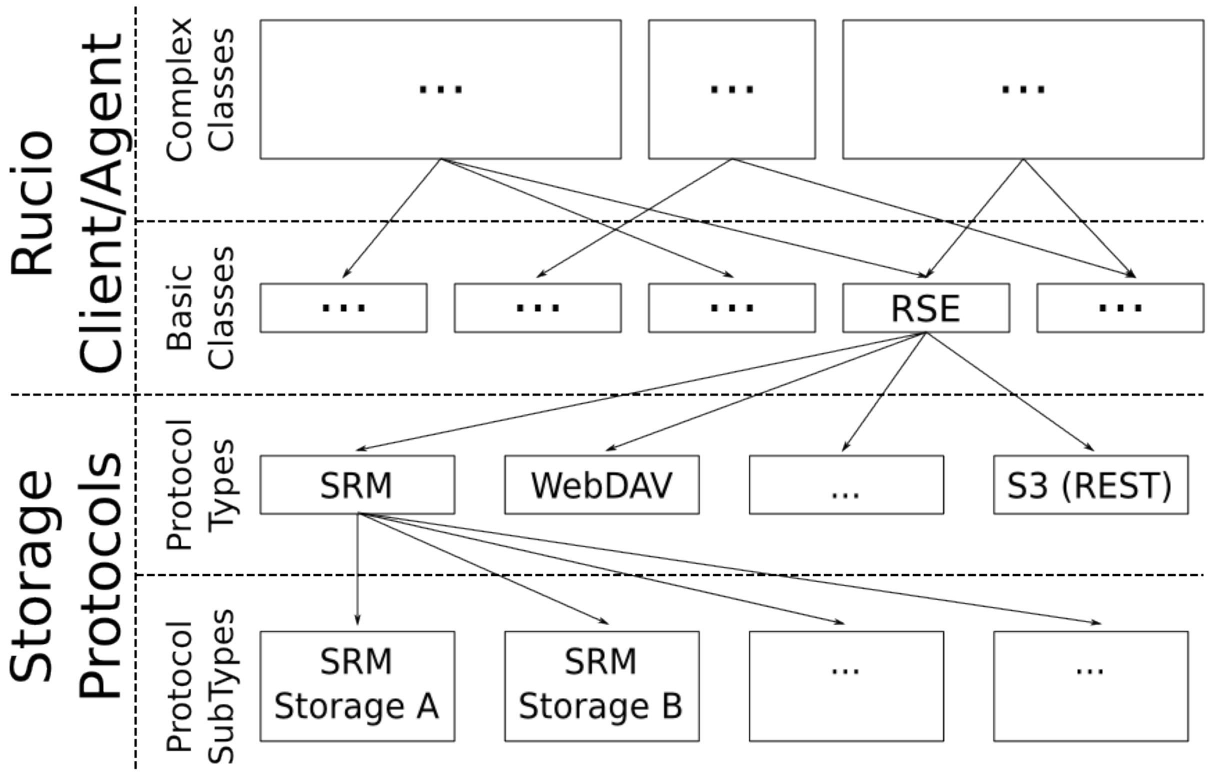
Rucio Storage Elements (RSE) are container for physical files. They are designed to provide a way to define each storage and all its supported protocols separately and include them in a transparent way for users and developers.

Doing so creates the flexibility to add new storages and protocols to the system without changing other classes using RSEs. It further allows for separate update strategies for each storage, as all information needed to interact with a specific storage is gathered at runtime (Rucio Storage Element Properties (RSEProperties) and Rucio Storage Element Protocol Type (RSEProtocol)).

Overview

Architectural Overview

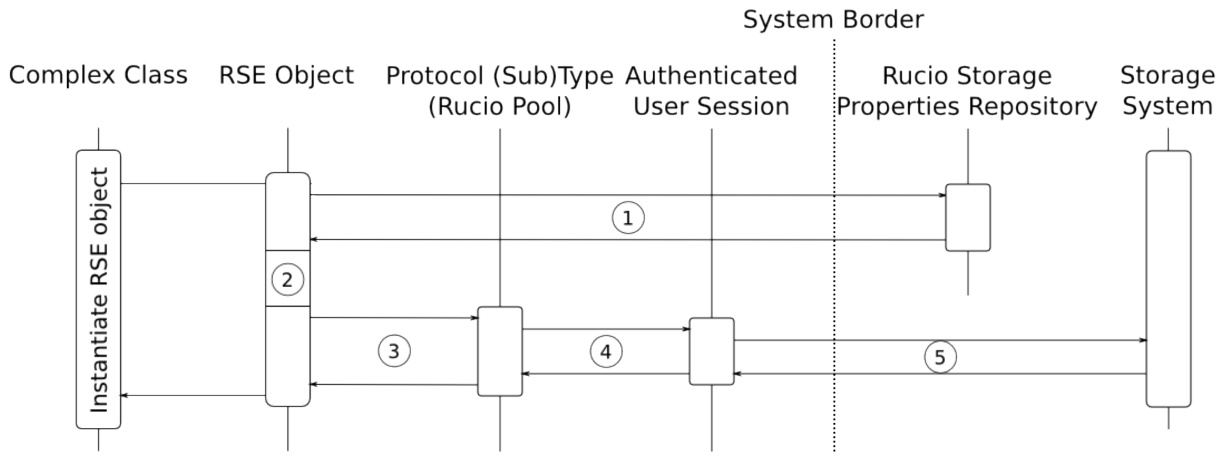
As illustrated below, each complex class (representing a particular workflow/use case) uses one or more basic (or even complex) classes to perform the intended operation. One of these basic classes is the RSE class, describing a generic storage element. Whenever interaction with a storage is requested by any other class inside Rucio it instantiates a generic RSE object. The ID of the actual storage must be provided as a parameter to the init-function. As this class offers a common interface, supported by all storages, no further knowledge about a specific storage or protocol is required at this time.



Because some storages use the same protocol, a Rucio-wide pool of protocol implementations for the system is also provided. What protocols (cloud be more than one) are supported by a specific storage is defined in its according properties (RSEProperties). Unfortunately, different version/implementations of one protocol, which differ in various ways, are implemented by the different storages. Due to re-usability reasons we therefore decided to allow various subtypes of each protocol. If for example a specific storage supports SRM, but not in the default implementation, it defines in its properties which particular subtype it supports. .. }}}

What Happens During Instantiation

In this section we explain what actually happens when (a) an RSE object is instantiated and (b) interaction with a specific storage is requested.


System Border
Complex Class
RSE Object
Protocol (Sub)Type (Rucio Pool)
Authenticated User Session
Rucio Storage Properties Repository
Storage System
Instantiate RSE object
1
2
3
4
5

Whenever an interaction with a storage is requested, an RSE object has to be instantiated. The illustration above gives an overview about what happens at run-time during its instantiation.

1. Repository Lookup: To allow the generic RSE object to interact with a specific storage, the properties defined for it must be requested from the Rucio Storage Properties Repository. What this properties are in particular will be discussed in the according section later in this document.

2. Interpreting The Properties: Based on the information received in Step One, the RSE object is now able to set its internal properties to allow future interaction with the requested storage. Examples for this internal values are the storage specific prefix for physical file names, the names of the supported protocols, or the URI of the storage, to name only a few. Again, a detailed discussion about this properties is given in the according section later in this document.

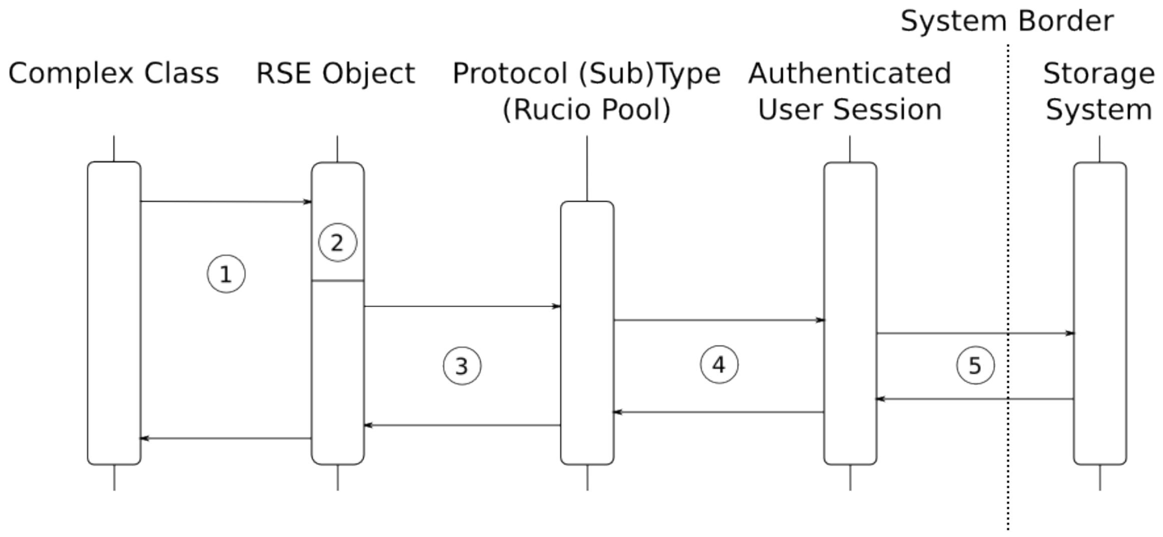
3. Instantiating The Protocol: Now all information about the requested storage is set up, the RSE object instantiates the specified protocol class. Because all protocols in the pool provide similar functionality, they can be used in an arbitrary way. The instance of the protocol is kept as a private member of the RSE object, as it is of no further interest for any other class.

4. Initializing an Authenticated User Session: Each protocol class in the pool initializes an authenticated user session which is stored inside the protocol object for later usage. This avoids unnecessary traffic and workload created by user authentication for each storage request and therefore compliments the overall system performance.

5. Instantiating the Authenticated User Session at the Storage System: Finally the user session must be authenticated at the specified storage system for later usage.

After these four steps, the generic RSE object has become storage specific. What operations are supported by these RSE objects will be discussed in the according section later in this document.

In the following we discuss what happens if storage interaction is requested from an already instantiated RSE object. It should be noted that in the illustration below all objects are provided/created at some point before the focus of the illustration.



1. Interaction Request: Whenever a complex class wants to interaction with the specified storage, it uses one of the operations provided by the RSE object. This operation could be for example GET a file, GET a directory listing, DELETE a file, … A detailed discussion about the provided operations is given in the according section later in this document.

2. Parameter Transformation: First the RSE object has to adapt the input parameters in a way to match the referred storage (e.g. transforming the logical file name (LFN) to the physical file name (PFN), adding storage specific prefixes). All adaptations are based on information received from the Rucio Storage Properties Repository during object instantiation. At the end of this block, the input parameters are matching the interface (by structure and value) of the according operation implemented by the specified protocol.

3. Calling the According Protocol Operation: Because now all parameters are ready to be used by the protocol, the RSE object can delegate the actual storage interaction to the protocol object.

4. Using the Authenticated User Session: The protocol object reuses the authenticated user session created during the instantiation to interact directly with the referred storage system.

5. The Actual Storage Interaction: The interaction with the storage system is performed as defined in the protocol.

Rucio Storage Element Object

As mentioned in the introduction, Rucio Storage Elements (RSE) are the main components used to access files inside a storage system. In this section we discuss its API and provide some Python snippets how a they are intended to be used.

Methods

CREATE

Whenever a new instance of an RSE class is created, the ID of the targeted storage and the protocol to use can be provided as paremters. If not, RSE will select a storage system and a protocol automatically, based on the current state of the various storage systems.

Instantiating an RSE object can be done on four different ways:

The first example instantiates an RSE to interact with the storage 'Ralph-Laptop' and specifies to use exclusively the S3 protocol for it.

```
# 1. Specific Storage, Specific Protocol
my_storage = rse.create( {'id':'Ralph-Laptop', 'protocol':'s3'} )
```

The second example instantiates an RSE object to interact with the storage 'Ralph-Laptop' and specifies to use the preferred protocol of the storage system for it.

```
# 2. Specific Storage, Unspecified  Protocol
my_storage = rse.create( {'id':'Ralph-Laptop'} )
```

The third example instantiates an RSE object to an unspecified storage supporting the S3 protocol. Which storage is actually used may vary from instantiation to instantiation, depending on the current state of each registered storage. See the section about Properties for further information on this topic.

```
# 3. Unspecific Storage, Specific Protocol
my_storage = rse.create( {'protocol':'s3'} )
```

The fourth example instantiates an RSE object to an unspecified storage using its preferred protocol. Which storage is actually used may vary from instantiation to instantiation, depending on the current state of each registered storage. See the section about Properties for further information on this topic.

```
# 4. Unspecific Storage, Unspecific Protocol
my_storage = rse.create()
```

GET

The get-method provides functionality to access either the content of a file or its meta-data. It can be used in two different ways:

1. The Object is Already Created by a Previous 'Create-Statement': As described above, if the object is already instantiated it is connected to a specific storage and therefore only the Logical File Name (LFN) must be provided to the get-method. If only the LFN is provided as input, the get-method responds the content of the referred file, while appending '/meta-data' to the LFN (REST-like resource addressing) indicates that only the meta-data of the file are requested. The meta-data of the file will be responded as a JSON object.

The following example assumes that the RSE object (my_rse) was already created by some preceding statements:

```
meta = my_rse.get('my_logical_file_name/meta-data')
content = my_rse.get('my_logical_file_name')
```

The snippet above first requests the meta-data of the file 'my_logical_file_name' and afterwards its content. Because this time the RSE object is already connected to a specific storage and a protocol, no further information must be provided.

2. Using the Get-Method from the Class Instead of the Object: If there is no RSE-object, it is possible to use the get-method implemented by the class itself. Because the RSE class is not connected to any specific storage or protocol, this information must be provided preceding the LFN (inspired by URIs). E.g.

```
# Bad if both statements are here, OK if only one is
meta = rse.get('s3://Ralph-Laptop/my_logical_file_name/meta-data')
content = rse.get('s3://Ralph-Laptop/my_logical_file_name')
```

The snippet above, again requests the meta-data of the file 'my_logical_file_name' located at the storage system 'Ralph-Laptop' using the S3 protocol and afterwards its content. The major difference to the snippet before is, that this time the user authenticated session at the storage system must be created before the request and will be closed immediately after the request is finished. Therefore **the usage of this operation should be omitted if one or more files are expected to be requested later on.**

PUT

Using this operation allows for updating already existing data in the storage system. Again, like for the get-method, two different modes (one with and one without a pre-existing RSE object) are implemented. The advantages and disadvantages of the two methods are the same as for the get-method and therefore not discussed here again.

The snippets below give an example how to update a file or its meta-data at a specific storage.

1. To update a files content in the storage system the LFN of the file and the local file object of the new version (local_file_content) must be provided as input.

```
# 1. Pre-Existing RSE object (my_rse)
my_rse.put('my_logical_file_name', local_file_content)

# 2. Using the class operation
rse.put('s3://Ralph-Laptop/my_logical_file_name', local_file_content)
```

2. To update the meta-data of a file the LFN and the JSON object, representing them, must be provided as input.

```
# 1. Pre-Existing RSE object (my_rse)
my_rse.put('my_logical_file_name/meta-data', local_meta_data)

# 2. Using the class operation
rse.put('s3://Ralph-Laptop/my_logical_file_name/meta-data', local_meta_data)
```

Note: Only meta-data included in the provided JSON object and the file meat-data will be updated. Meta-data that is present in the storage, but missing in the JSON object will stay unchanged. Meta-data present in the JSON object but missing in the meta-data of the file (or the according schema) will be ignored.

POST

Using this operation allows for creating new data in the storage system. Again, like for the get-method, two different modes (one with and one without a pre-existing RSE object) are implemented. The advantages and disadvantages of the two methods are the same as for the get-method and therefore not discussed here again.

To create data in the storage system the LFN of the file, the local file object (local_file_content) and it meta-data represented by a JSON object (local_meta_data) must be provided as input.

```
# Pre-Existing RSE object (my_rse)
my_rse.post('my_logical_file_name', local_file_content, local_meta_data)

# Using the class operation
rse.post('s3://Ralph-Laptop/my_logical_file_name', local_file_content, local_meta_data)
```

The snippets above give an example how to insert a file and its meta-data to a specific storage.

DELETE

Using this operation allows for deleting data from the storage system. Again, like for the get-method, two different modes (one with and one without a pre-existing RSE object) are implemented. The advantages and disadvantages of the two methods are the same as for the get-method and therefore not discussed here again.

To delete a specific file from a storage system only its LFN must be provided. Whenever a file is removed from a storage system, the according meta-data is too. At this point it should be noted that **no wildcard support** is implemented by this method.

```
# Pre-Existing RSE object (my_rse)
my_rse.delete('my_logical_file_name', local_file_content, local_meta_data)

# Using the class operation
rse.delete('s3://Ralph-Laptop/my_logical_file_name', local_file_content, local_meta_data)
```

The snippets above give an example how to delete a file from a specific storage.

Example Code

Here, a couple of sample use cases are provided to give an idea how RSE objects are intended to be used by developers.

Fetching File Data (multiple times)

```
# Fetching three particular files from a specified storage using its preferred
# protocol

lfns = ['some_very_logical_file_name', 'also_some_very_logical_file_name','again_some_very_logical_fi
storage_id = 'some_awsome_storage'
results = {}
try:
  my_rse = rse.create({'id': storage_id})
  for lfn in lfns:
    file_meta_data = my_rse.get(lfn + '/meta-data')
    file_content = my_rse.get(lfn)
    results[lfn] = {'content' : file_content, 'meta-data' : file_meta_data}
expect RSEException as e:
  print 'Error No: ' + e.get_id() + ': ' + e.get_message()
```

Updating File Meta-Data

```
# Updating the meta-data of one specific file at a specific storage.
# Because in this use-case we assume that no further interaction with
# the referred storage is planed, the class method is used:

updated_meta_data = ... # Representing the results of all the painful analyzing work
lfn = 'some_very_logical_file_name'
storage_id = 'some_awsome_storage'
try:
  rse.put('s3://' +  storage_id + '/' + lfn + '/meta-data', updated_meta_data)
expect RSEException as e:
  print 'Error No: ' + e.get_id() + ': ' + e.get_message()
```

Creating a new File

```
# Creating a new file in a specified storage, using a specified protocol.
# Because in this use-case we assume that some further interaction with this
# particular storage will follow, we create an RSE object instead of using the
# class method.

storage_id = 'some_awsome_storage'
protocol_id = 'S3'
lfn = 'some_very_logical_file_name'
file_meta_data = ... # Representing the meta-data as JSON
file_content = open('somewhere/on/my/disk/is/my_local_file')
try:
  my_rse = rse.create({'id' : storage_id, 'protocol' : protocol_id })
  my_rse.post(lfn, file_conent, file_meta_data)
```

```
expect RSEException as e:
  print 'Error No: ' + e.get_id() + ': ' + e.get_message()
```

Rucio Storage Properties Object

To enable an RSE object to interact with each storage system, specified inside the Rucio Storage Properties Repository, a common set of properties is needed. Further are these properties acting as a base for automatic decision making when automatic storage selection is used. All this information is represented by Rucio Storage Element Properties (RSEPs).

In order to achieve the intended functionality, the information of each storage is split into two major parts (namely static and dynamic). Each part must represent a common set of key-value pairs provided by all storage systems specified inside the repository.

In the following the understanding of the terms ‘static properties’, and ‘dynamic properties’ is discussed. At the end, the according JSON Schema, as one way to validate that each storage provides sufficient and well-formed information when added to the repository, is given.

Static Properties

Static properties, as understood here, do not vary on a regular basis. Therefore this information is kept (static) inside the repository. Having this information provided here saves bandwidth and storage computing resources every time a client requests information about a specific storage. Further is querying and/or filtering storages based on static information possible by performing only one request (to the repository), what otherwise would be at least one separate call for each storage (directly to the storage). This increase in resource efficiency justifies the more complex maintenance task by updating the information inside the repository whenever a static value changes.

The following listing gives some examples how static properties may look like.

```
'static' : {
  'id' : 'cern.storage.user.ralph.laptop',
  'name' : 'Ralph's Laptop at CERN',
  'location' : {
    'address' : '1-R-024, CERN CH-1211, Genève 23',
    'country' : 'CH'
  },
  'overall_diskspace' : {'value':'128', 'unit':'GB'}
  'overall_computing_power' : {'value' ; '1.4', 'unit' : 'GHz'},
  'protocols' : [{'s3': []}, {'webdav':['sub1','sub2']}]
  }
}
```

The example above describes a storage system represented by one of our laptops located at CERN, Switzerland. It has 128GB of overall disk space and 1.4 GHz overall computing power. The supported protocols are S3 (default implementation) and WebDAV (only subtypes sub1 and sub2).

Dynamic Properties

In contrast to static properties, dynamic properties vary from request to request. Examples for such properties are current work load, available disk space, current connection bandwidth, . . . Because of their dynamic nature, the values of this properties are not kept inside the repository. For this scenario, it saves resources if the values are requested from the storage on demand instead of automatically update the repository information each time a certain value changes, like static properties.

To enable the client to query these dynamic properties, each property is represented by a method which must be defined inside the class of the according protocol. If a storage systems requires different operations to the one defined inside the default one to provided the requested information, a new subtype protocol must be defined. This way, the flexibility for each storage system to implement its individual way to provide the requested data is provided.

```
'dynamic' : {
  's3' : {
    'available_disk_space' : {
      'method' : 'get_diskspace()',
      '[some additional protocol specific information]'
    },
    'current_workload' : {
      'method' : 'get_workload()',
      '[some additional protocol specific information]'
    },
  ...
  }
  'webdav_sub1' : {
    'available_disk_space' : {
      'method' : 'get_diskspace()',
      '[some additional protocol specific information]'
    },
    'current_workload' : {
      'method' : 'get_workload()',
      '[some additional protocol specific information]'
    },
  ...
  },
  'webdav_sub2' : {
    ...
  },
...
}
```

The example above specifies for the according storage system and its supported protocols, how clients are able to request data using the according protocol object.

JSON Schema

To guarantee the information provided for each storage matches the common set, a JSON schema [1] is defined. Using this schema allows clients to verify if the responded data is valid, and therefore helps to write less complex code during implementation. Further supports this schema developers when defining the RSEP for a storage system by acting as a guideline. Again, by validating the RSEP against the schema, mistakes and errors can be prevented. Inside the central repository the validation of the data is performed automatically each time a information about a storage is created or updated.

Implementation Details

If there is something that needs to be explained further it will be written here.

Rucio Storage Protocol Object

VERY FLUFFY - more thinking needs to be done

As already mentioned above, in Rucio a pool with various protocols is provided. These protocols represent the actual interaction with the storage systems e.g. using SRM or S3. Because of the heterogeneity of the different storage

[1] Link to JSON Schema: http://json-schema.org/

systems, also sub types of protocols are supported. This way each storage system is able to describe its own implementation specialities (see also Rucio Storage Element Properties for details).

Methods

The Rucio Storage Protocol class is used whenever an RSE object interacts with a storage system. To enable this in a transparent way, each protocol or sub type must implement at least the methods defined in the generic protocol class, representing a common set of operations provided by each storage system. To ensure this, each protocol class must be inherited from the generic class.

CREATE_SESSION

Creates an authenticated user session at the specified storage system. TODO: Rucio Authentication

CLOSE_SESSION

Closes the existing user authenticated session at the according storage system.

READ

Returns the content of the requested file from the storage system.

WRITE

Writes the provided data into the referred file at the storage system.

REGISTER_FILE

Registers the uploaded file with its LFN and PFN at the according storage system.

DELETE

Deletes the referred file from the storage system.

LFN2PFN

Converts the logical file name into the physical file name of the according storage system.

Rucio Storage Exception Object

Like Rucio itself, RSE objects use RucioExceptions to escalate errors. For easier coding it is sub-classed as RSEException.

An RSEException consists of three attributes: the ID representing an unique integer identifier for each exception, the message text which is printed along side the ID if the exception is transformed to string and a data field for additional information to the exception.

Exception Codes

In the following a comprehensive list of all exceptions is given. TODO: Discuss if this list comprehensive, what is not needed

ID	Message Text	Description
101	Switching Protocols	The storage indicates the client to use a different protocol to fulfil its request.
202	Requested Accepted	Indicates that the request successfully transmitted to storage and that it will be executed later.
204	No Response	The storage has completed the request, but no content is provided to the client.
300	Multiple Endpoints	The storage provides this file multiple times (with different protocol) and the client has to select one.
301	Moved Permanently	The requested file has been permanently moved to a different location.
302	Found (but at a different endpoint)	The requested resource resides temporarily under a different URI.
304	Not modified	The requested file can be found in the client cache.
400	Bad Request	The requested was rejected by the server due to malformed syntax.
402	Payment Required	Nothing more to say. :)
403	Forbidden	The client has not necessary privileges to access this resource at this storage.
404	Resource Not Found	The requested resource was not found at the specified storage.
405	Method Not Allowed	The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.
409	Conflict	The clients request is in conflict with the rules defined for the storage.
410	Gone	The requested resource is no longer available at this storage.
413	Requested Entity To Large	The storage is refusing to process a request because the request entity is larger than the storage is willing or able to process.
500	Something Embarrassing Happened	Should not happen.
503	Service Unavailable	The requested service temporary not accessible for the client.
504	Gateway Timeout	The storage received a timeout while interacting with other storages.

Note: Because in Rucio Exception are always related to some unsolicited behaviour, they are not used to confirm an expected state.

Methods

GET_ID

Returns the ID of the exception according to the list above.

GET_MESSAGE

Returns the description of the message as defined in the list above.

GET_DATA

Returns a dictionary including additional information to the exception. For example the ID of the protocol if the error with the ID 101 is thrown or the URI of the storage where the requested file can be found if the error 302 is thrown. Details about this additional information can be found in the list above.

TO_STR

Returns a string consisting of the ID and the message of the exception.

Example Implementations

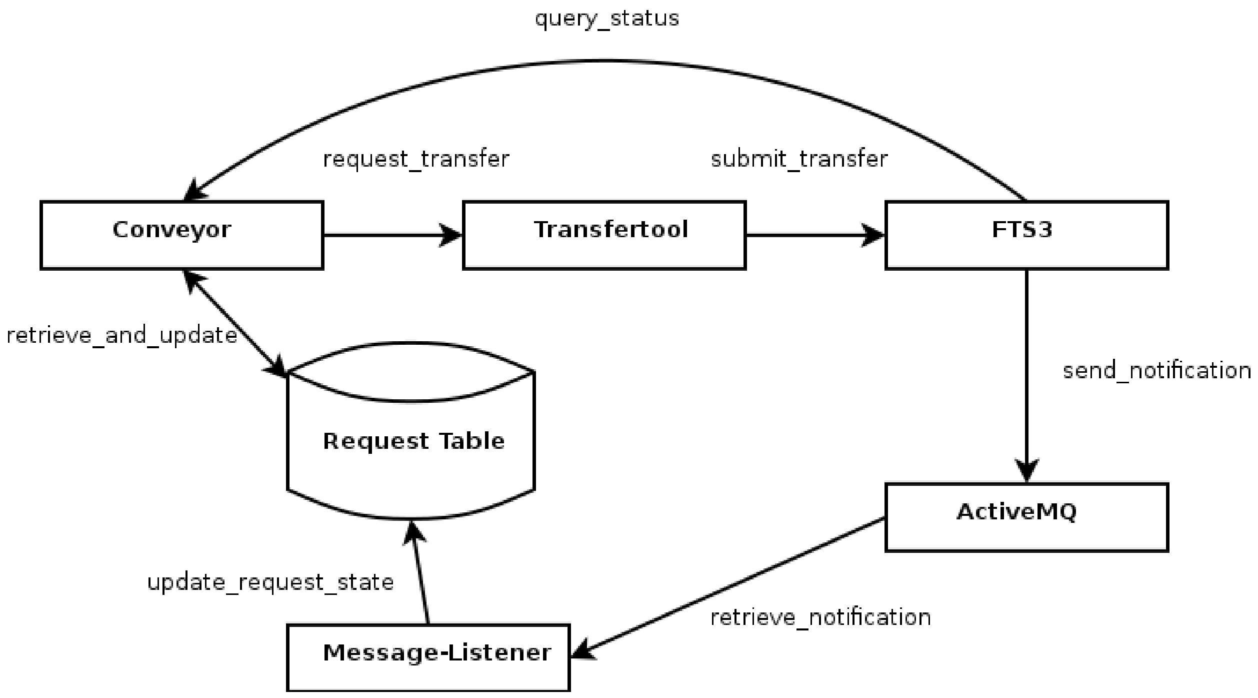
File System

WebDAV

S3

Scheduled Transfers

There are three components involved with scheduled transfer requests: the conveyor daemon, the transfertool, and the message-listener. The conveyor daemon selects which request should be queued for transfer, and then hands them over to the transfertool. The transfertool implements the binding to a, possibly third-party, service that actually executes the transfer. The message-listener is an optional component that updates the status of the actual transfer within Rucio, if a third-party system sends status update events. If the external system does not send events, then the Conveyor polls instead periodically.



Conveyor

The conveyor consists of two independent threads, the submitter and the poller. Multiple instances of the conveyor can be started as well.

The submitter retrieves a request from the database, in status QUEUED and with type TRANSFER. It then resolves the currently available source replicas for the request and submits it to a given implementation of the transfertool interface.

Currently, only FTS3 as a transfertool is implemented. After submission the submitter continues immediately with the next available request.

The poller retrieves a request of type TRANSFER from the database, which has been in state SUBMITTED since more than, configurable, one hour. Single file transfer should not take longer than this, which most likely means that the notification from the external transfer system was lost. The poller then explicitly queries the external transfer system for the status of the transfer, and resets the appropriate status (SUBMITTED, DONE, FAILED) within Rucio.

Transfertool

The transfertool implements three actions, submitting a transfer, querying the status of a transfer, and cancelling a transfer. There is currently only one implementation of the transfertool, for the WLCG File Transfer Service 3 (FTS3). It uses the REST API of FTS3 to submit or cancel JSON-encoded transfers, and parses the JSON responses of job status queries.

Message-Listener

The Message-Listener updates the state of the request within Rucio, after receiving ActiveMQ notifications from FTS3. The events are consumed based on an “vo=atlas” selector, so it actually has to consume all transfer state changes, but it will only act when an external state change would result in a Rucio request state change. The Message-Listener is also not a requirement, as the poller in the conveyor will ensure that every request will be processed eventually.

Data deletion

Replica data model:

- rse_id: The rse identifier
- scope: The scope name of the file
- name: Filename
- state: Replica state (AVAILABLE, UNAVAILABLE, COPYING, BEING_DELETED, BAD)
- lock_cnt: Counter of the nr of locks
- created_at: Creation date of the replica1
- accessed_at: Last access time of the replica (null for the moment)
- tombstone: Date of the last lock removal (null otherwise) (Maintained in the rule core part.)

Reaper daemon

Parameters:

- MinFreeSpace: Minimun free space(in bytes) which should be available at a RSE.
- MaxBeingDeletedFiles: Maximum number of files beeing deleted for a RSE.

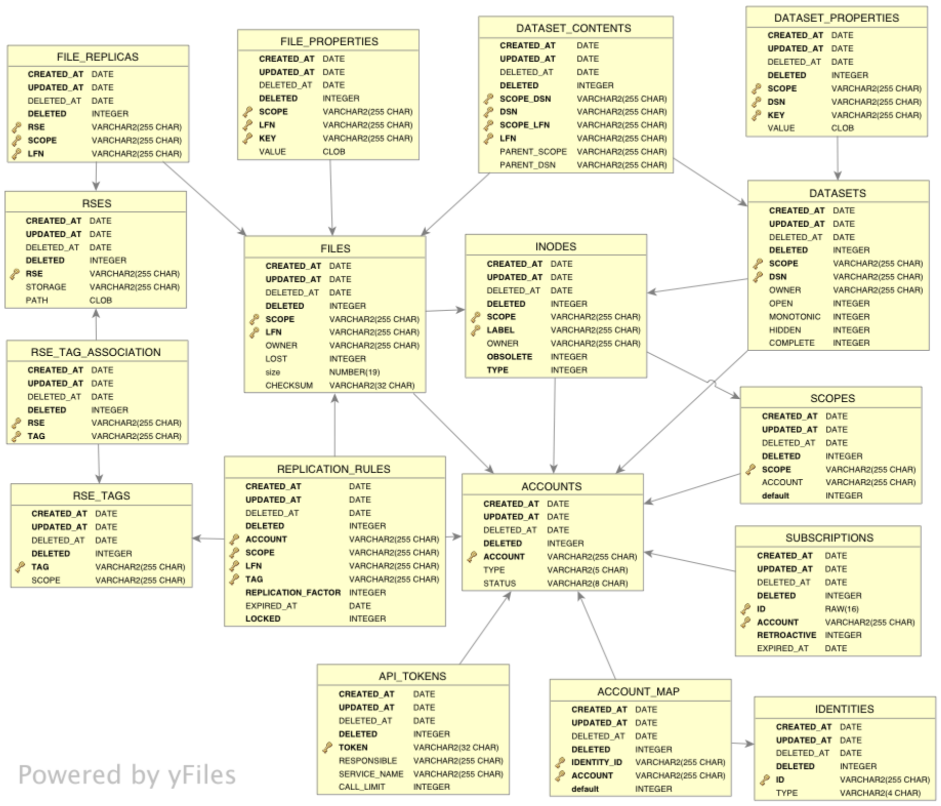
For each RSE, The reaper checks the available free space. If the available free space is less than a limit defined per RSE(MinFreeSpace), a cleanup cycle is triggered. The reaper selects (MinFreeSpace - ActualFreeSpace) bytes of files and then deletes them. Replicas of any file which is declared obsolete are immediately deleted. Those files are marked with a tombstone set to the epoch value (January 1, 1970).

Pseudo-code:

```
For each RSE in RSEs:
    Check rse free space # Explanation below
    replicas = list_unlocked_and_obsolete_replicas (MinFreeSpace - ActualFreeSpace)
    # Gives n replicas corresponding to the amount of bytes to free up or all obsolete ones
    # The list is ordered by created_at and tombstone.
    for each replica in replicas:
        status = Mark the replica state as BEING_DELETED
            # Update conditions: where lock_cnt=0 and tombstone is not null
            # Rules should not select replicas in the state BEING_DELETED
        if status:
                status = Delete file from RSE
                if status:
            Remove file from replica table
freed_space += filesize
```

Rucio Database Schema

This figure describes the database schema including the relationships between the tables, the available indexes and primary keys, and the identifiers tracked by the database.



Replication Rules Architecture

Architecture Overview

The replication rules are managed by two components in Rucio.

- Rules Core module

  The rules core module offers functions which are directly called by the API or other core components. The main functionality offered are creating, listing and deleting replication rules.

- Rules daemon: Rucio-Judge

  The rules daemon is responsible for making sure that replication rules are correct when datasets/containers are changed. The judge also takes care of the deletion of expired rules and re-evaluates replication rules which are in the STUCK state.

Database Schema

There are two tables which hold the information concerning all replication rules and locks.

Rules

Locks

Also relevant are the **DataIdentifier** table, holding all data identifiers, the **DataIdentifierAssociation** table, expressing the relation between child and parent dids as well as the **RSEFileAssociation** table, which is the catalog of all physical replicas.

Relevant System interactions

Besides listing and searching replication rules, there are 6 Rucio interactions which are relevant for the rule component.

- Creating a replication rule

  A replication rule is always created for a specific did (file, dataset, container). When the rule is created it is evaluated immediately, thus creating all replica locks and, if necessary, file transfers. This action is directly linked to the core method:

  `rucio.core.rule.`**`add_replication_rule`**`(`*dids*, *account*, *...*`)`

- Deleting a replication rule

  Replication rules can be deleted by their owner (or an privileged account). The removal of the rule and it's associated locks is done by the core function:

  `rucio.core.rule.`**`delete_replication_rule`**`(`*rule_id*, *...*`)`

- Adding a did to a parent did

  Attaching a data identifier to a dataset or container has to trigger a rule evaluation, as all parent rules have to be applied to the new children as well. The method flags the did for re-evaluation in the DID table. This re-evaluation is done asynchronously by the Rucio-Judge. The action is directly linked to the core method:

  `rucio.core.did.`**`attach_identifier`**`(`*scope*, *name*, *dids*, *...*`)`

- Removing a did from a parent did

When removing dids from a dataset or container, the previously matching rules may not match anymore. Thus the respective locks have to be removed from the files. The method flags the did for re-evaluation in the DID table. This is done asynchronously by the Rucio-Judge. This action is linked to the core method:

`rucio.core.did.`**`detach_identifier`**(*scope*, *name*, *dids*, *...*)

- Successfully finishing a transfer

  When a transfer finishes the state of all affected locks (one new replica can affect many locks) and rules have to be updated. This action is linked to the core method:

  `rucio.core.rule.`**`successful_transfer`**(*scope*, *name*, *rse_id*)

- Failing a transfer

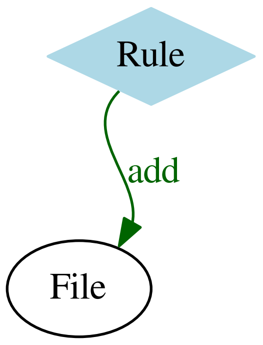
  When a transfer fails the state of all affected locks and rules has to be updated, so that the Rucio-Judge can make new decisions to repair the rule. This action is linked to the core method:

  `rucio.core.rule.`**`failed_transfer`**(*scope*, *name*, *rse_id*)

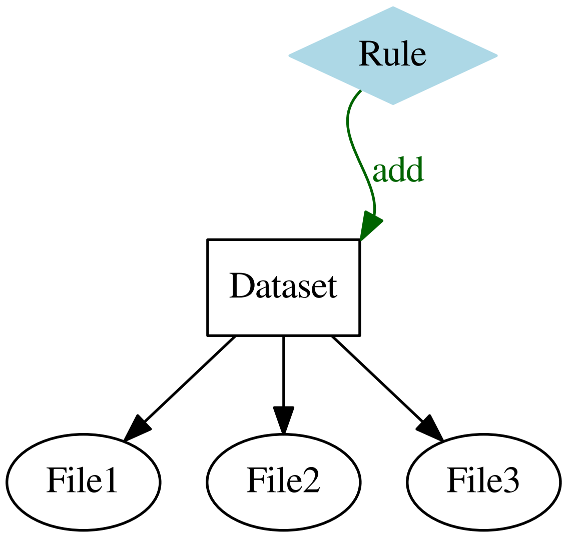
General Workflow

Adding replication rules

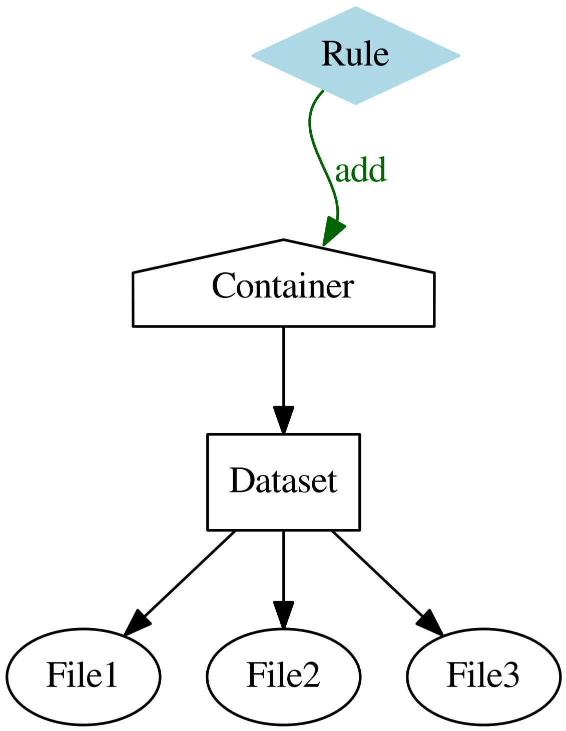
This section describes the general workflow when adding replication rules. There are basically three different cases that have to be considered. Rules added to a file, dataset or container.



Rule on file



Rule on dataset



Rule on container

The general workflow when adding a rule is described as follows: (Ignore lines 2, 6, 7 and 9 for the moment)

```
1  Create a DB transaction
2  Row-Lock the did in the did table
3  Resolve the RSE expression to a list of potential RSEs
4  Get the current quota/usage values of the account for each RSE
5  Create the replication rule in the ReplicationRule table
6  if did.type==CONTAINER:
7         Row-Lock all child datasets and containers in the did table
8  Resolve the did to it's files and get all associated ReplicaLocks
9  Row-Lock all these ReplicaLocks (Actually done in the same query as 8.)
10 if grouping==NONE:
11     for each file:
12         Pick N rses for the file considering filesize, quota, weights and existing locks of the file
13 if grouping==ALL:
14     Calculate size of all files
15     Calculate the current coverage (in bytes) of the files on the rses
16     Pick n rses considering the sum-size, quota, weights and rse coverage
17 if grouping==DATASET:
18     for each dataset:
19         Calculate the size of all files in the dataset
20         Calculate the current coverage (in bytes) of the files in the dataset on rses
21         Pick n rses considering the dataset-size, quota, weights and rse coverage
22 Create the locks in the database
```

```
23 Create the necessary transfers
24 Commit the DB transaction
```

Right now, the decisions in line 12, 16 and 21 where to create the new replica locks are done as follows:

1. Exclude all potential RSEs which do not have enough quota to hold the file/dataset/container.
2. If the RSEs already hold replica locks of the concerned replicas, sort these RSEs by number of replica locks.
3. Pick the first N RSEs out of the list.
4. If RSEs hold the same amount of replica locks (or no locks at all) pick N RSEs according to the RSE weights.

Example:

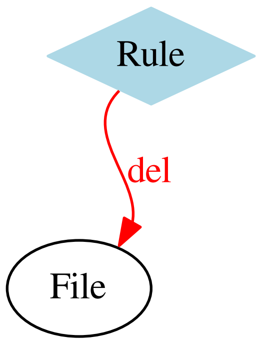
```
DatasetA = [File0, ..., File9] with 100 MB per file
potential RSEs:
    RSEA ... 1300 MB quota, 3 replicas of DatasetA, weight=0.1
    RSEB ... 400 MB quota, 5 replicas of DatasetA, weight=10
    RSEC ... 4000 MB quota, 0 replicas of DatasetA, weight=100
    RSED ... 3000 MB quota, 0 replicas of DatasetA, weight=50
Rule: 2 replicas, DATASET grouping
```

1. Exclude all potential RSEs which do not have enough quota (RSE B gets excluded); potential RSEs = [RSEA, RSEC, RSED].
2. RSEA already holds replicas, put RSEA to priority List; priorityRSEs = [RSEA].
3. Pick the first 2 RSEs out of the priorityRSEs list; (There is only 1 entry); RSEA is picked, 1 remaining RSE to pick.
4. RSEC and RSED left in potential RSE list; Pick according to weights; (Random pick according to weights) RSEC gets picked.

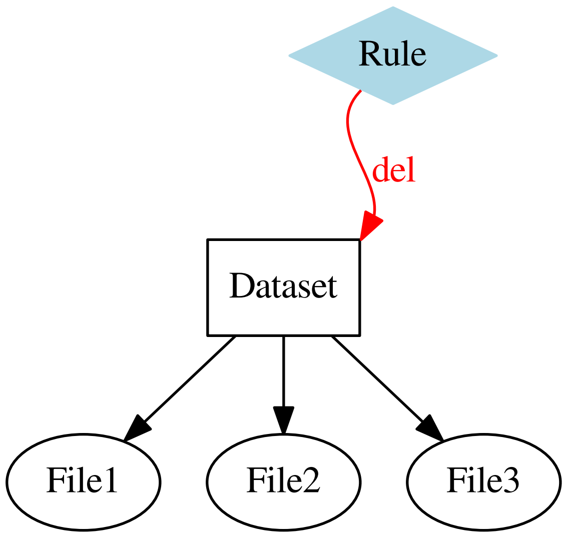
Result: Put replica locks on RSEA and RSEC.

Deleting replication rules

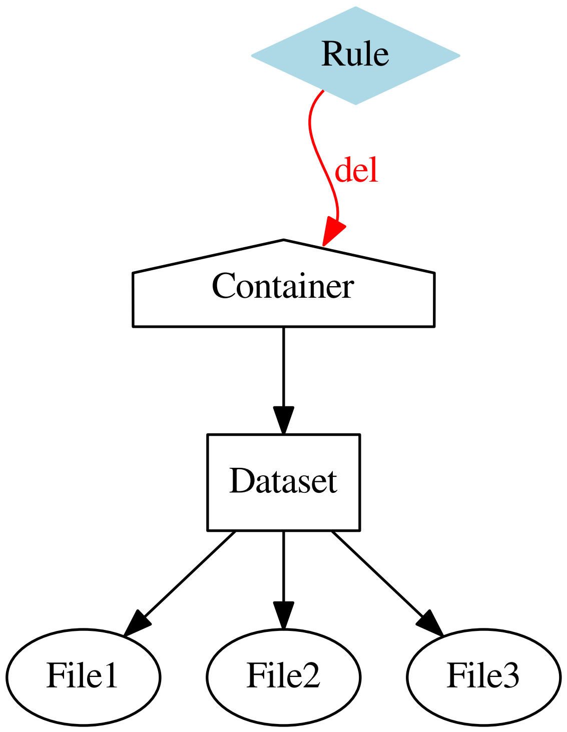
This section describes the general workflow when deleting replication rules.



Rule on file



Rule on dataset



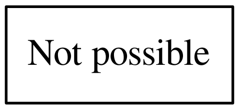
Rule on container

The general workflow when deleting a rule is described as follows: (Ignore lines 2 and 11 for the moment)

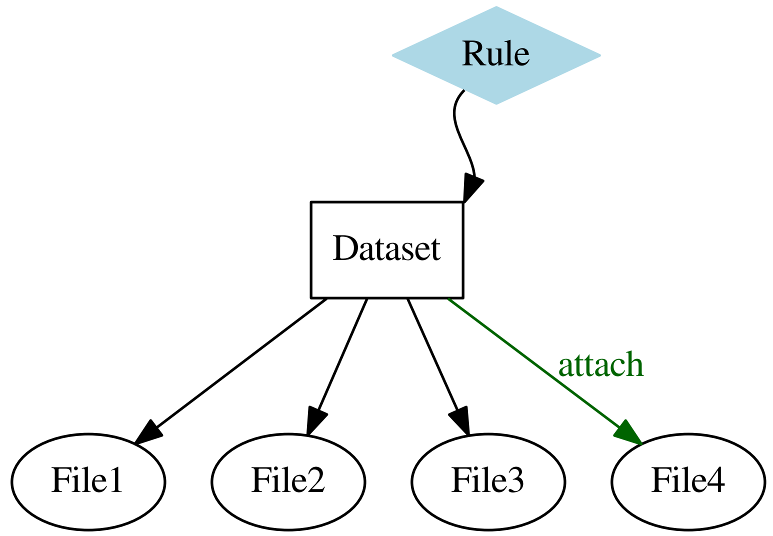
```
1  Create a DB transaction
2  Row-Lock the replication rule in the ReplicationRule table
3  if state==OK:  # There are no running transfers!
4      Delete the replication rule from the ReplicationRule table (Locks will be deleted cascading)
5  if state==SUSPENDED:  # There are no running transfers!
6      Delete the replication rule from the ReplicationRule table (Locks will be deleted cascading)
7  if state==STUCK:  # There are no running transfers!
8      Delete the replication rule from the ReplicationRule table (Locks will be deleted cascading)
9  if state==REPLICATING:  # There are running transfers which may have to be cancelled
10     Get all ReplicaLocks for all files affected by this rule
11     Row-Lock the Locks in the ReplicaLocks table (Will be done in the same query)
12     for each file:
13         If the lock is REPLICATING and there are no other Locks, cancel the transfer
14 Commit transaction
```

Re-Evaluating a DID (Something has been added)

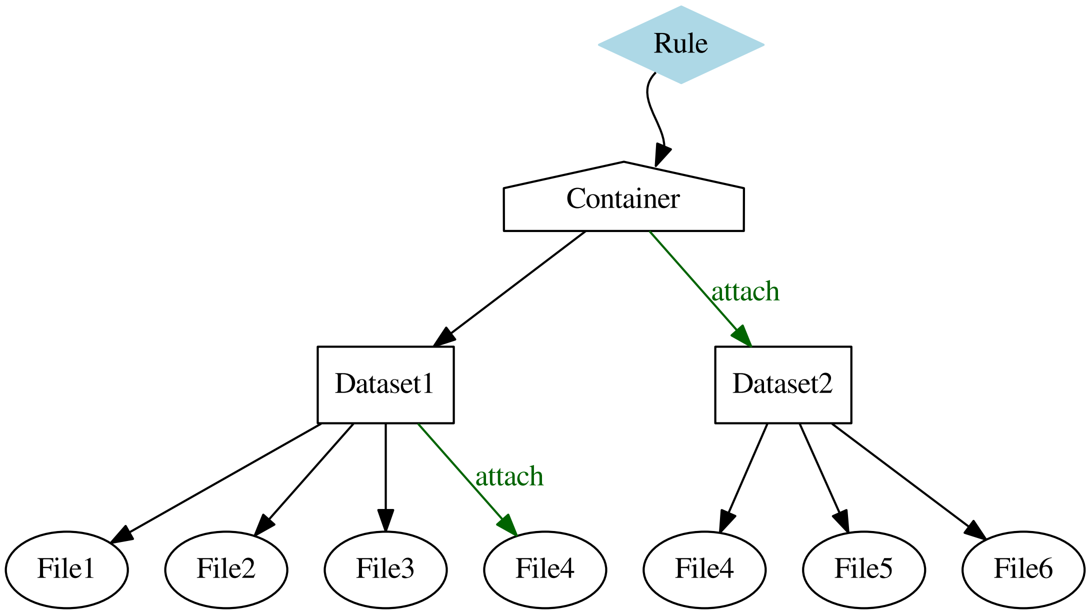
When files are added to datasets or datasets/containers to containers, the affecting rules have to be re-evaluated and Locks have to be set on the new children.



Rule on file



Rule on dataset



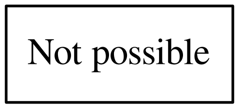
Rule on container

The general workflow when re-evaluating a rule is described as follows: (Ignore lines 3, 5, 7, 9, 12 and 14 for the moment)

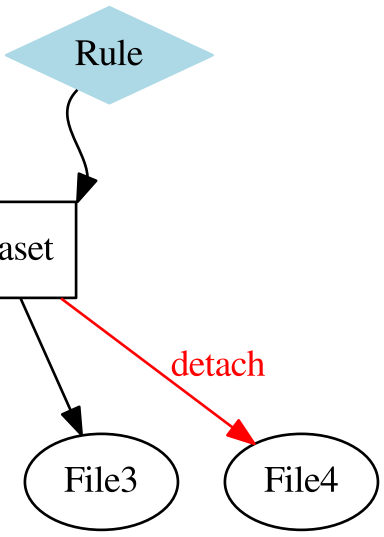
```
1  Create a DB transaction
2  Pickup the next did which needs re-evaluation
3  Row-Lock this did in the DID table
4  Get all parent dids (Go up the tree)
5  Row-Lock these parent dids in the DID table
6  Get all replication rules from the parent dids and the did itself
7  Row-Lock these rules in the Rule table
8  Get all the newly attached children of the DID
9  Row-Lock these dids in the did table
10 if these new child dids are DATASETS or CONTAINERS:
11     Follow the tree down the path to get all new_files
12     Row-Lock all intermediate datasets and containers in the DID table
13 Get the ReplicaLocks for all new files
14 Row-Lock these ReplicaLocks in the Lock Table
15 for all rules found in the parents:
16     Resolve RSE Expression and fetch Quotas
17     if rule.grouping == DATASET or CONTAINER:
18         Based on the existing locks get the grouping decision which has been made before
19     For the new files make the lock decisions and create transfers (Same algorithm as in Add Replicat
20     If same grouping is not possible due to quota, pick another RSE
21     If anything of the above fails, mark the rule as STUCK
22 Mark the did as re_evaluated
23 Commit the transaction
```

Re-Evaluating a DID (Something has been removed)

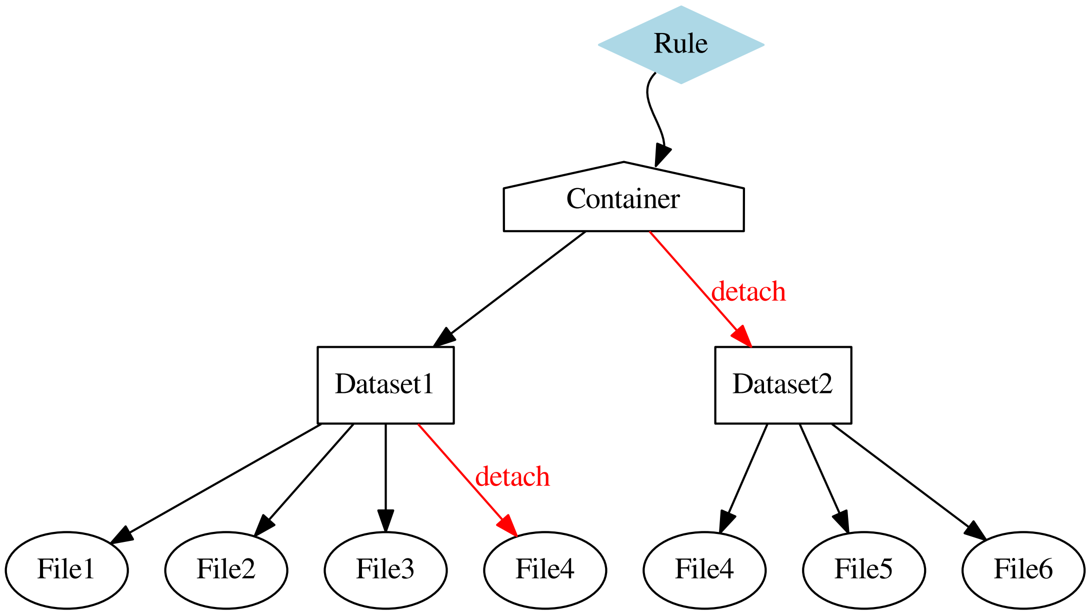
When files are removed from datasets or datasets/containers from containers, the affecting rules have to be re-evaluated and Locks have to be removed.



Rule on file



Rule on dataset



Rule on container

The general workflow when re-evaluating a rule is described as follows: (Ignore lines 3, 5, 7, and 11 for the moment)

```
1  Create a DB transaction
2  Pickup the next did which needs re-evaluation
3  Row-Lock this did in the DID table
4  Get all parent dids (Go up the tree)
5  Row-Lock these parent dids in the DID table
6  Get all replication rules from the parent dids and the did itself
7  Row-Lock these rules in the Rule table
8  Get all the files of the did (Does not consider the removed ones)
9  for each rule:
10     Get all locks of the rule
11     Row-Lock these locks
12     if there is no file for the lock, the lock can be deleted
13 Mark the did as re_evaluated
14 Commit the transaction
```

Updating locks on successful/failed transfer

If a transfer is successful or fails, all the locks for the file on this RSE have to be updated.

On successful transfer (Ignore line 3 for now)

```
1  Create a DB transaction
2  Get all the locks of the transferred file on the rse
3  Row-Lock these locks
4  for each lock:
5      Update Lock state to OK
6      Check if the replication rule of the lock has any REPLICATING locks left, if not mark the rule a
```

```
7  Commit the transaction
```

On failed transfer (Ignore line 3 for now)

```
1  Create a DB transaction
2  Get all the locks of the transferred file on the rse
3  Row-Lock these locks
4  for each lock:
5      Update Lock state to STUCK
6      Check if the replication rule of the lock has any REPLICATING locks lefts, if not mark the rule a
7  Commit the transaction
```

Race-Conditions and concurrency problems

This section specifically describes the race-conditions and concurrency issues that could bring the rules out of sync with the actual files. As this is very critical and cannot be detected easily, it is very important to prevent this issues in the first place.

1. A rule is applied to a did while some did in the structure (higher or lower) is being changed concurrently

   - *Adding replication rules* line 8 would read a did listing which could be invalid at t+1 and thus not apply the rule to every file. We therefore make the following requirement: **Whenever a did is changed or a rule is applied/evaluated on a did the session needs to acquire a row-lock of the did in the did table!** Thus we add line 2:

```
Row-Lock the did in the did table
```

     As every action needs to acquire a lock in the did table, only a single session can change the did. However, also dids lower in the structure could be changed concurrently. To prevent this we add line 6 and 7:

```
if did.type==CONTAINER:
    Row-Lock all child datasets and containers in the did table
```

   - *Re-Evaluating a DID (Something has been added)* To prevent changes of the did itself we add line 3:

```
Row-Lock this did in the DID table
```

     Line 4 would read a did listing of higher-level dids which could be invalid at t+1 and thus not apply the rules correctly. We add line 5:

```
Row-Lock these parent dids in the DID table
```

     Line 8 would read a did listing of lower-level dids which could be invalid at t+1 and thus not apply the rules correctly. We add line 9:

```
Row-Lock these dids in the did table
```

     Also line 12 is added for the same purpose:

```
Row-Lock all intermediate datasets and containers in the DID table
```

   - *Re-Evaluating a DID (Something has been removed)* Line 8 could end up with a wrong did listing at t+1. To prevent changes of the did itself we add line 3:

```
Row-Lock this did in the DID table
```

     Line 4 could end up with an inconsistent listing at t+1. As parent dids have to be prevented from changing as well, line 5 is added:

```
        Row-Lock these parent dids in the DID table
```

- When a did is attached to a parent-did, this parent-did has to be row-locked as well.
- When a did is detached from a parent-did, this parent-did has to be row-locked as well.

2. While a did is re-evaluated, other rules (applying to the did) are changed concurrently

   - *Adding replication rules* Line 8 would maybe get the right ReplicaLocks, but when creating the new locks (under the assumption that another lock is already there) this creates problems when these locks are deleted concurrently; Thus, when these locks get deleted while they are used as assumption for re-evaluation, it could happen that a lock is being created without a file replica (and without a transfer to create one). The solution is in line 9:

```
        Row-Lock all these ReplicaLocks (Actually done in the same query as 8.)
```

   By row-locking all the ReplicaLocks, it is not possible that a lock is being deleted while it is used as an assumption for re-evaluation.

   - *Deleting replication rules* As the adding part is requesting row-locks for the rule and locks, it is important the the deletion part does the same. Line 2:

```
        Row-Lock the replication rule in the ReplicationRule table
```

   Line 11:

```
        Row-Lock the Locks in the ReplicaLocks table (Will be done in the same query)
```

   - *Re-Evaluating a DID (Something has been added)* These rule and replica-lock locks have also be requested in the re-evaluation part. Line 7:

```
        Row-Lock these rules in the Rule table
```

   and line 14:

```
        Row-Lock these ReplicaLocks in the Lock Table
```

   - *Re-Evaluating a DID (Something has been removed)* The same thing also applies for the deletion part. Rules and Locks have to be row-locked as they cannot be used by another session concurrently. Line 7:

```
        Row-Lock these rules in the Rule table
```

   Line 11:

```
        Row-Lock these locks
```

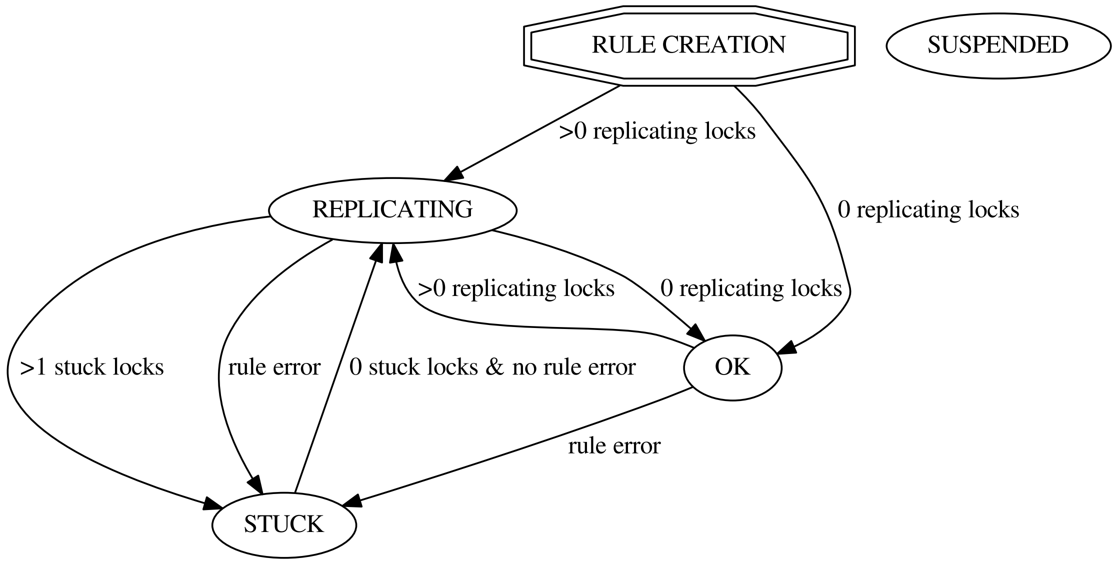
   - *Updating locks on successful/failed transfer* also when updating locks on successful/failed transfers, these locks have to be row-locked. Line 3:

```
        Row-Lock these locks
```

Rules Workflow

When a rule is created, at least one replica lock is created for each of the associated files of the did. These replica locks can be in 3 states: OK, if the replica already exists at the site; REPLICATING, if the replica is in the process of beeing transfered to the RSE and STUCK, if several transfer attempts have been made unsuccessfully. Usually, new rules will only have replica locks in the state OK or REPLICATING. After rule creation, depending if there are any REPLICATING locks (or all have already been satisfied) the rule is created in the states OK or REPLICATING.

However, as new files can be added to the did later, which results in additional replica locks, the state of the rule can change according to the following state diagram:



The SUSPENDED state is set and unset manually.

Exception and error handling

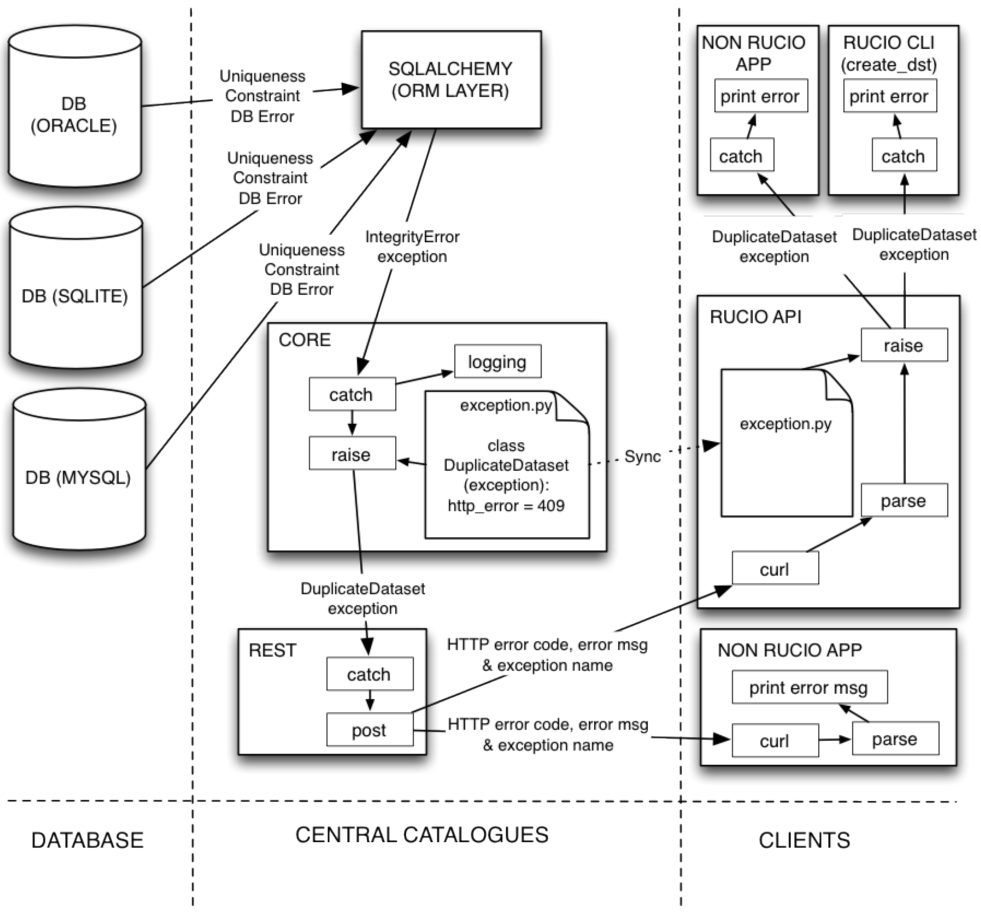
Exception Handling in Rucio

In Rucio, state is kept at the database level. Consequently many exceptions originate from the ORM layer. These exceptions will be caught at the CORE layer, and translated into meaningful Rucio exceptions.

In the case where REST processes catch a Rucio exception, they will communicate to the client the HTTP error code, the name of the exception, and the corresponding error string. The reason for sending the name of the exception is that HTTP error codes can map to multiple exceptions, and consequently client code cannot use the HTTP error code to ascertain the type of exception that occurred.

To make the error response both readable by humans querying the system directly and also parsable for the Rucio client API the error information will be sent both as a string and also as a dictionary in the response header. The error string contains the exception class and a human readable error message separated by a colon. The HTTP response header has to mandatory fields “ExceptionClass” and “ExceptionMessage”. The will be used by the Rucio client API to raise the correct exception on the client side. If needed the headers can be extended to give further details about the error.

This process is summarised in the following diagram.


DB (ORACLE)
DB (SQLITE)
DB (MYSQL)
Uniqueness Constraint DB Error
Uniqueness Constraint DB Error
Uniqueness Constraint DB Error
SQLALCHEMY (ORM LAYER)
IntegrityError exception
CORE
catch
logging
raise
exception.py
class DuplicateDataset (exception): http_error = 409
Sync
DuplicateDataset exception
REST
catch
post
HTTP error code, error msg & exception name
HTTP error code, error msg & exception name
NON RUCIO APP
print error
catch
RUCIO CLI (create_dst)
print error
catch
DuplicateDataset exception
DuplicateDataset exception
RUCIO API
raise
exception.py
parse
curl
NON RUCIO APP
print error msg
curl
parse
DATABASE
CENTRAL CATALOGUES
CLIENTS

Usecases

Usecases covered by Rucio

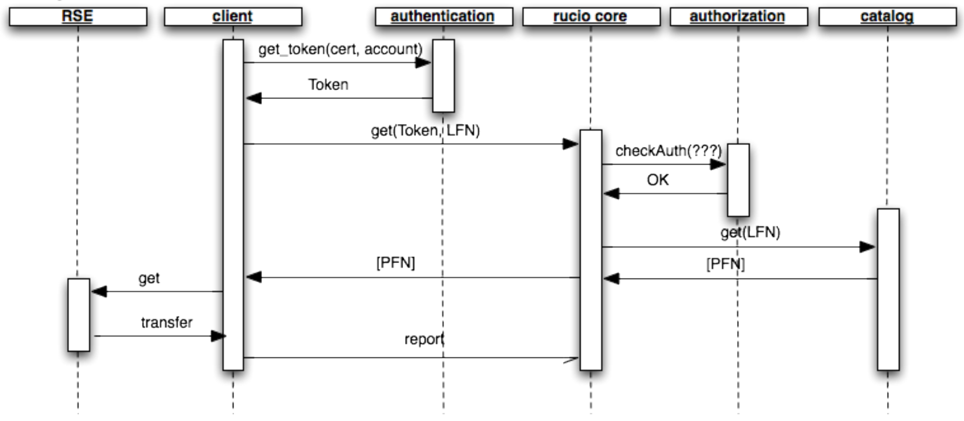
The following usecases are handled by Rucio and should be described with sequence diagrams.

- `usecases/authentication`
- `usecases/search`
- `usecases/usecase_upload_file_into_rucio` (Ralph)
- `usecases/add_account_identity`
- `usecases/add_metadata_dataset`
- `usecases/add_metadata_file`
- `usecases/add_scope_to_account`
- An user A should not be able to register a dataset in the scope of an user B
- Close a dataset (Angelos)
- `usecases/consistency_file_between_storage_and_rucio`
- Crosscheck that all files are still on disk and in the rucio catalog

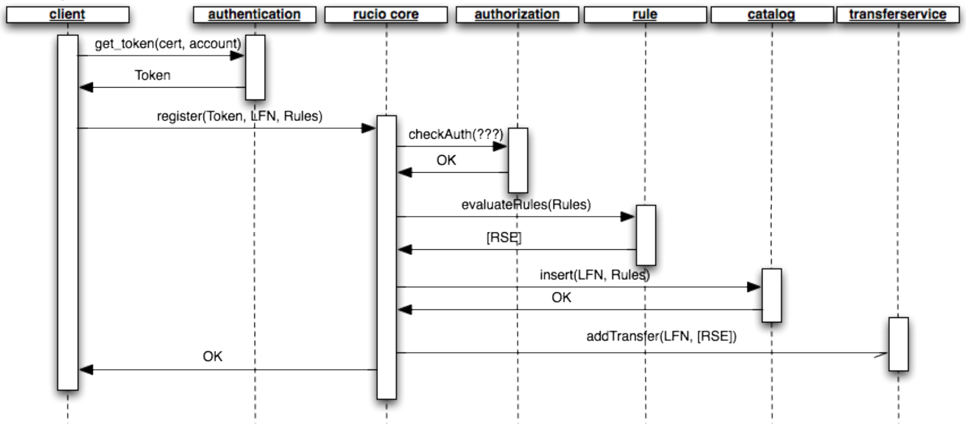
- Declare dataset unwanted (Angelos)
- Declare file as lost (Angelos)
- Declare file unwanted (Angelos)
- `usecases/delete_file_replica_from_storage`
- `usecases/detect_site_reach_watermark`
- `usecases/download_all_files_from_a_given_list_of_file_replicas_from_rucio` (Ralph)
- `usecases/download_all_files_from_a_given_list_of_files_from_rucio` (Ralph)
- `usecases/download_all_files_in_a_dataset_from_rucio` (Ralph)
- `usecases/download_files_from_rucio` (Ralph)
- Generate the list of files at a site
- Give how much data has an account
- List dataset parents (Angelos)
- `usecases/obsolete_dataset`
- `usecases/reupload_after_failure` (Ralph)
- Register a dataset with files (Angelos)
- `usecases/register_account` (Thomas)
- `usecases/register_transfer_request_file_fts`
- `usecases/register_file_already_on_storage_system` (Ralph)
- `usecases/remove_replication_rules_from_file` (Martin)
- `usecases/select_unwanted_files_for_deletion` (Martin)
- Send notifications when a transfer is done
- Set a quota on an account
- `usecases/set_replication_rule_to_file` (Martin)
- `usecases/add_subscription` (Martin)
- Tell how many files/how much space is used at a site
- `usecases/upload_file_with_replication_rule` (Martin/Ralph)
- `usecases/where_are_the_replicas_for_a_file` (Thomas)
- Where are the replicas for all files in dataset (Angelos)
- etc.

Sequence/flow diagrams

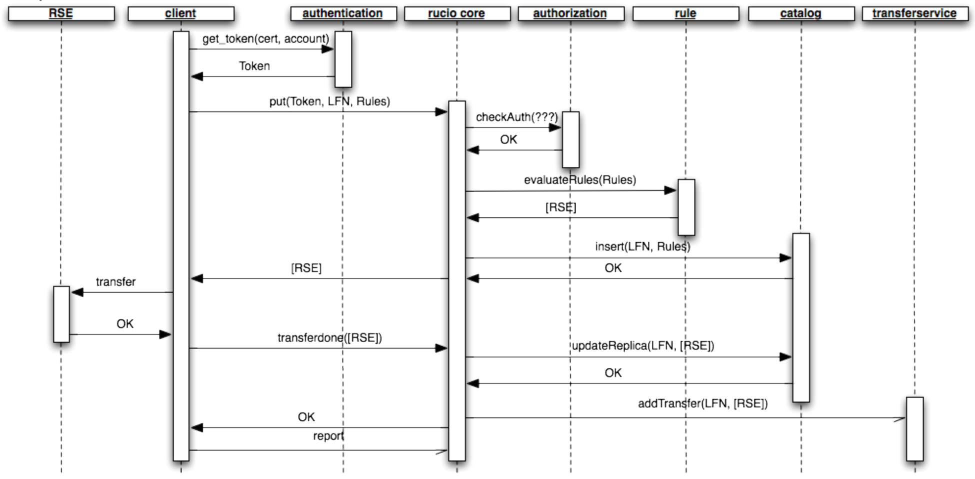
Rucio-get



rucio-register



rucio-put



CHAPTER 3

Developer Documentation

Development Guidelines

Coding Guidelines

For the most part we try to follow PEP 8 guidelines which can be viewed here: http://www.python.org/dev/peps/pep-0008/

There is a useful pep8 command line tool for checking files for pep8 compliance which can be installed with `easy_install pep8`. (It is also included in the virtual environment)

You can then run it manually with:

```
pep8 --repeat --ignore=E501 lib
```

(Yes, we ignore the "E501 - line too long" warning.)

To run the unit/integrations tests including the test-coverage:

```
nosetests -v --with-coverage --cover-package=rucio
```

Documentation Guidelines

The documentation in docstrings should follow the PEP 257 conventions (as mentioned in the PEP 8 guidelines).

More specifically:

1. Triple quotes should be used for all docstrings.
2. If the docstring is simple and fits on one line, then just use one line.
3. For docstrings that take multiple lines, there should be a newline after the opening quotes, and before the closing quotes.
4. Sphinx is used to build documentation, so use the restructured text markup to designate parameters, return values, etc. Documentation on the sphinx specific markup can be found here: http://sphinx.pocoo.org/markup/index.html

License and Copyright

Every source file must have the following copyright and license statement at the top:

```
# Copyright European Organization for Nuclear Research (CERN)
#
# Licensed under the Apache License, Version 2.0 (the "License");
# You may not use this file except in compliance with the License.
# You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0
#
# Authors:
# - XXXX XXXXX, <xxxx.xxxx@cern.ch>, 2012
```

All __init__.py files must have the same header, excluding the authors declaration. e.g.:

```
# Copyright European Organization for Nuclear Research (CERN)
#
# Licensed under the Apache License, Version 2.0 (the "License");
# You may not use this file except in compliance with the License.
# You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0
```

How to write test cases

The most important rule for API calls: ALWAYS WRITE YOUR TESTCASE AGAINST THE WEB-INTERFACE, NOT THE CORE API ITSELF! IF POSSIBLE, WRITE IT AGAINST THE CLIENT IF ONE EXISTS! This is to make sure that the full call chain works.

1. Test cases go into either lib/rucio/tests/
2. Filename must start with `test_`
3. Classname must start with Test
4. Test function names must start with `test_`
5. Do not import unittest
6. Do not subclass from unittest.TestCase
7. Remove the whole __name__ == '__main__' thing
8. Run all testcases with nosetests twice.

You can selectively run test cases by giving directories or files as parameters to the nosetests executable.

Executing unit tests the correct way

1. Run `tools/run_tests -1qa`

Setting up a Rucio development environment

Contributing patches & code review

Rucio follows a next/master development scheme: two protected branches called "next" and "master" track all features and patches. Contributors create their own private development branches to do their work, and once finished and code reviewed, these branches are merged into next and/or master.

The hosting is at GitLab, and the upstream is at:

https://gitlab.cern.ch/rucio01/rucio

First setup

Go to the GitLab web-interface mentioned above, and login with your CERN account. It will give you the option to fork the rucio01/rucio repository into your private account (upper left corner). Do this.

Afterwards, switch to your private account and clone it, for example:

```
$ git clone https://gitlab.cern.ch/<cern_username>/rucio.git
```

Setup your git environment. You must provide a valid e-mail address and ask Developer access to the project Rucio in Gitlab:

```
$ git config ---global user.name='Joaquin Bogado'
$ git config ---global user.name='joaquin.bogado@cern.ch'
```

The repository hooks and upstream are installed by executing the script tools/configure_git.sh:

```
$ cd rucio
$ ./tools/configure_git.sh
```

Verify that everything is alright. You should see both push/pull remotes for origin (your private account) and upstream (official rucio repository):

```
$ git remote -v
 origin  ssh://git@gitlab.cern.ch:7999/jbogadog/rucio.git (fetch)
 origin  ssh://git@gitlab.cern.ch:7999/jbogadog/rucio.git (push)
 upstream        https://gitlab.cern.ch/rucio01/rucio.git (fetch)
 upstream        xxx (push)
```

Also, it's necessary to create the file .gitlabkey with the development key provided by the GitLab interface (gitlabkey) in the local directory.

Installing dependencies

Rucio maintains three lists of dependencies:

```
$ tools/pip-requires
$ tools/pip-requires-client
$ tools/pip-requires-test
```

The first is the list of dependencies needed for running rucio, the second list includes dependencies used for the rucio python clients and CLIs and the third list is for active development and testing of rucio itself.

These depdendencies can be installed from PyPi using the python tool pip or by using the tools/install_venv.py script as described in the next section.

However, your system *may* need additional dependencies that *pip* (and by extension, PyPi) cannot satisfy. These dependencies should be installed prior to using *pip*, and the installation method may vary depending on your platform.

PyPi Packages and VirtualEnv

We recommend establishing a virtualenv to run rucio within. Virtualenv limits the python environment to just what you're installing as dependencies, useful to keep a clean environment for working on rucio. The tools directory in rucio has a script already created to make this very simple:

```
$ python tools/install_venv.py
```

This will create a local virtual environment in the directory `.venv`.

If you need to develop only the clients and have a default configuration:

```
$ python tools/install_venv.py --atlas-clients
```

Once created, you can activate this virtualenv for your current shell using:

```
$ source .venv/bin/activate
```

The virtual environment can be disabled using the command:

```
$ deactivate
```

You can also use `tools\with_venv.sh` to prefix commands so that they run within the virtual environment. For more information on virtual environments, see virtualenv.

Lastly you have to create a symbolic link from the virtual environments python directory to the rucio source directory:

```
$ cd .venv/lib/python2.7/site-packages/
$ ln PATH_TO_INSTALL_DIRECTORY/lib/rucio/ rucio -s
```

Developing a feature

Features are scheduled for the next Rucio release and are collected from the protected “next” branch. Create a new feature branch with:

```
$ tools/create-feature-branch <ticketnumber> <branch description>
```

and do your development there. When done, push the branch into origin (your private account) for code review:

```
$ tools/submit-merge
```

Developing a patch

A patch works exactly the same, but is branched off the “master”. Create a new patch branch with:

```
$ tools/create-patch-branch <ticketnumber> <branch description>
```

and do your development there. When done, push the branch into origin (your private account) for code review::

```
$ tools/submit-merge
```

Code review and merging a patch

Two rules must be obeyed:

1. Feature branches must be merged into “next”
2. Patch branches must be merged into “master” and “next”

(For now, step 2 is manual, we will automate it in the future.) Click the “Merge request” button in the web-interface and select the (potentially two) appropriate destination branches, e.g., from youraccount/featurebranch to rucio/next. Don’t forget that patch branches need two merge requests, both into “next” and “master”. (In the future, this will be automated. It is also possible to do this via CLI only, no web interface is actually needed.)

The merge request will enable the code review. After successful code review, the responsible can merge the patch on the web interface.

If something is weird, ask for help on rucio-dev@cern.ch :-D

Ticketing system

For Rucio we are using Jira to manage the development of the project:

> https://its.cern.ch/jira/browse/RUCIO

Tickets for new features should be submitted with a functional granularity, that is according to the API call being introduced and at which level it belongs. For example, “register_dataset API (CORE)”, “register_dataset (REST)”, and “register_dataset API (CLIENT)”, instead of big and vague new feature definitions like “new dataset functionality”. This level of granularity allows better tracking of the progress of the RUCIO project, informs developers when new interfaces become available, and leads to more meaningful changelogs when a release is made.

In order to avoid generating too many tickets and insuring the documentation of relevant work is placed in a single description, all minor schema changes and corresponding test cases should be included as part of the new feature ticket and seperate tickets should not be made. The exception to this is if additional functionality, a bug fix or a new test case is added to the task in a newer release of Rucio, this then should be documented as a new ticket, rather than modifying the existing ticket (as it is assigned to the previous Rucio release).

The ticket workflow in Jira is summarised here:

> https://confluence.atlassian.com/download/attachments/284367573/system-workflow.png

When one is finished working on a new feature or bug fix and this has been commited and submitted to Code Review for approval, the ticket status should be changed to ‘resolved’. Once the new code has been approved and commited to the GIT master the ticket status should be changed to ‘closed’.

GIT commits should include the relevant JIRA ticket number(s) in the beginning of the commit message. This is because Jira is integrated with GIT and will associate the tickets to the corresponding GIT commits.

Jira ticket headers and descriptions will be included on release changelogs. For this reason the titles and descriptions should be meaningful.

Naming convention for the Rucio database objects

All names must not be enclosed in quotes so that Oracle stores them in upper case in the data dictionary.

The Primary Key constraints (which would mean its index will have the same name)

name = TABLE_NAME || COLUMN_NAME(s) ||’_PK’

The Unique constraint name

name = TABLE_NAME || COLUMN_NAME(s) ||’_UQ’

Not Null constraints

name = TABLE_NAME || COLUMN_NAME || ‘_NN’

Note: This needs to be checked with sqlalchemy

Foreign Keys

name = TABLE_NAME || COLUMN_NAME(s) || '_FK'

Normal indexes

name = TABLE_NAME || COLUMN_NAME(s) || '_IDX'

Sequences

name = TABLE_NAME || COLUMN_NAME || '_SEQ'

Constraint types

name = TABLE_NAME || COLUMN_NAME || '_CHK'

Installing Rucio

Installing Rucio Clients

Prerequisites

Rucio clients runs on Python 2.6, 2.7.

Platforms: Rucio should run on any Unix-like platform.

Python Dependencies

Rucio clients need the following python modules:

```
argparse>=1.4.0           # Command-line parsing library
argcomplete>=1.8.2        # Bash tab completion for argparse
kerberos>=1.2.5           # Kerberos high-level interface
pykerberos>=1.1.14        # Kerberos high-level interface
requests==2.17.3          # Python HTTP for Humans.
requests-kerberos==0.11.0 # A Kerberos authentication handler for python-requests
wsgiref>=0.1.2            # WSGI (PEP 333) Reference Library
dogpile.cache>=0.6.2      # Caching API plugins
nose>=1.3.7               # For rucio test-server
boto>=2.46.1              # S3 boto protocol
python-swiftclient>=3.3.0 # OpenStack Object Storage API Client Library
tabulate>=0.7.7           # Pretty-print tabular data
progressbar>=2.3          # Text progress bar
bz2file>=0.98             # Read and write bzip2-compressed files.
python-magic>=0.4.13      # File type identification using libmagic
futures>=3.1.1            # Clean single-source support for Python 3 and 2
six>=1.10.0               # Python 2 and 3 compatibility utilities
```

All Dependencies are automatically installed with pip.

Install via pip

When `pip` is available, the distribution can be downloaded from the Rucio PyPI server and installed in one step:

```
$> pip install rucio-clients
```

This command will download the latest version of Rucio and install it to your system.

Upgrade via pip

To upgrade via pip:

```
$> pip install --upgrade rucio-clients
```

Install via pip and virtualenv

To install the Rucio clients in an isolated `virtualenv` environment:

```
$> wget --no-check-certificate https://raw.github.com/pypa/virtualenv/master/virtualenv.py
$> python virtualenv.py rucio
$> source rucio/bin/activate.csh
$> pip install rucio-clients
$> export RUCIO_HOME=`pwd`/rucio/
```

Installing using setup.py

Otherwise, you can install from the distribution using the `setup.py` script:

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

Installing ATLAS Rucio Clients

Prerequisites

Rucio clients runs on Python 2.6, 2.7.

Platforms: Rucio should run on any Unix-like platform.

Python Dependencies

Rucio clients need the following python modules:

```
argparse>=1.4.0            # Command-line parsing library
argcomplete>=1.8.2         # Bash tab completion for argparse
kerberos>=1.2.5            # Kerberos high-level interface
pykerberos>=1.1.14         # Kerberos high-level interface
requests==2.17.3           # Python HTTP for Humans.
requests-kerberos==0.11.0 # A Kerberos authentication handler for python-requests
wsgiref>=0.1.2             # WSGI (PEP 333) Reference Library
dogpile.cache>=0.6.2       # Caching API plugins
nose>=1.3.7                # For rucio test-server
boto>=2.46.1               # S3 boto protocol
python-swiftclient>=3.3.0 # OpenStack Object Storage API Client Library
tabulate>=0.7.7            # Pretty-print tabular data
progressbar>=2.3           # Text progress bar
bz2file>=0.98              # Read and write bzip2-compressed files.
python-magic>=0.4.13       # File type identification using libmagic
futures>=3.1.1             # Clean single-source support for Python 3 and 2
six>=1.10.0                # Python 2 and 3 compatibility utilities
```

All Dependencies are automatically installed with pip.

Setup CERN AFS client on lxplus

CERN Quickstart:

- bash:

```
    $> source /afs/cern.ch/atlas/offline/external/GRID/ddm/rucio/testing/bin/activate
```

- csh:

```
    $> source /afs/cern.ch/atlas/offline/external/GRID/ddm/rucio/testing/bin/activate.csh
```

By default the RUCIO_ACCOUNT variable is set to the AFS username.

- To test rucio:

```
    $> rucio ping
    $> rucio whoami
    $> rucio-admin account list-identities $RUCIO_ACCOUNT
```

Install via pip

When `pip` is available, the distribution can be downloaded from the Rucio PyPI server and installed in one step:

```
$> pip install atlas-rucio-clients
```

This command will download the latest version of Rucio and install it to your system.

Upgrade via pip

To upgrade via pip:

```
$> pip install --upgrade rucio-clients
```

Install via pip and virtualenv

To install the Rucio clients in an isolated `virtualenv` environment:

```
$> cd /tmp
$> curl -s https://raw.github.com/pypa/virtualenv/master/virtualenv.py | python2.6 - rucio # install
$> source rucio/bin/activate
$> pip install atlas-rucio-clients # install rucio-clients + atlas config
$> export RUCIO_ACCOUNT=$USER
$> source rucio/bin/activate
$> rucio ping
```

Installing using setup.py

Otherwise, you can install from the distribution using the `setup.py` script:

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

Installing Rucio server

Prerequisites

Rucio server runs on Python 2.6, 2.7.

Platforms: Rucio should run on any Unix-like platform.

Python Dependencies

Rucio server needs the following python modules:

```
# pip==9.0.1        # PyPA recommended tool for installing Python packages
SQLAlchemy==1.1.9 # db backend
alembic==0.9.1 # Lightweight database migration tool for SQLAlchemy
Mako==1.0.6 # Templating language
python-editor==1.0.3 # Programmatically open an editor, capture the result
flup==1.0.2 # Needed to deploy web.py in lighthttpd
```

All Dependencies are automatically installed with pip.

Install via puppet

puppet is an open and widely configuration management and automation system, for managing the infrastructure.

1. On the target node: start with a clean slate:

```
$> rm -rf /opt/rucio
```

2. On the puppet master: install with puppet

   do something like this in /etc/puppet/manifests/nodes.pp:

```
node '<hostname>' inherits basenode
{
  include 'rucio::lighttpd'
  include 'rucio::server-dev'
}
```

   then execute and wait:

```
$> puppet kick <hostname>
```

3. back to the target node: configure:

```
$> cd /opt/rucio/etc
$> cp rucio.cfg.template rucio.cfg
$> cd web
$> cp lighttpd.conf.template lighttpd.conf
```

4. startup lighttpd:

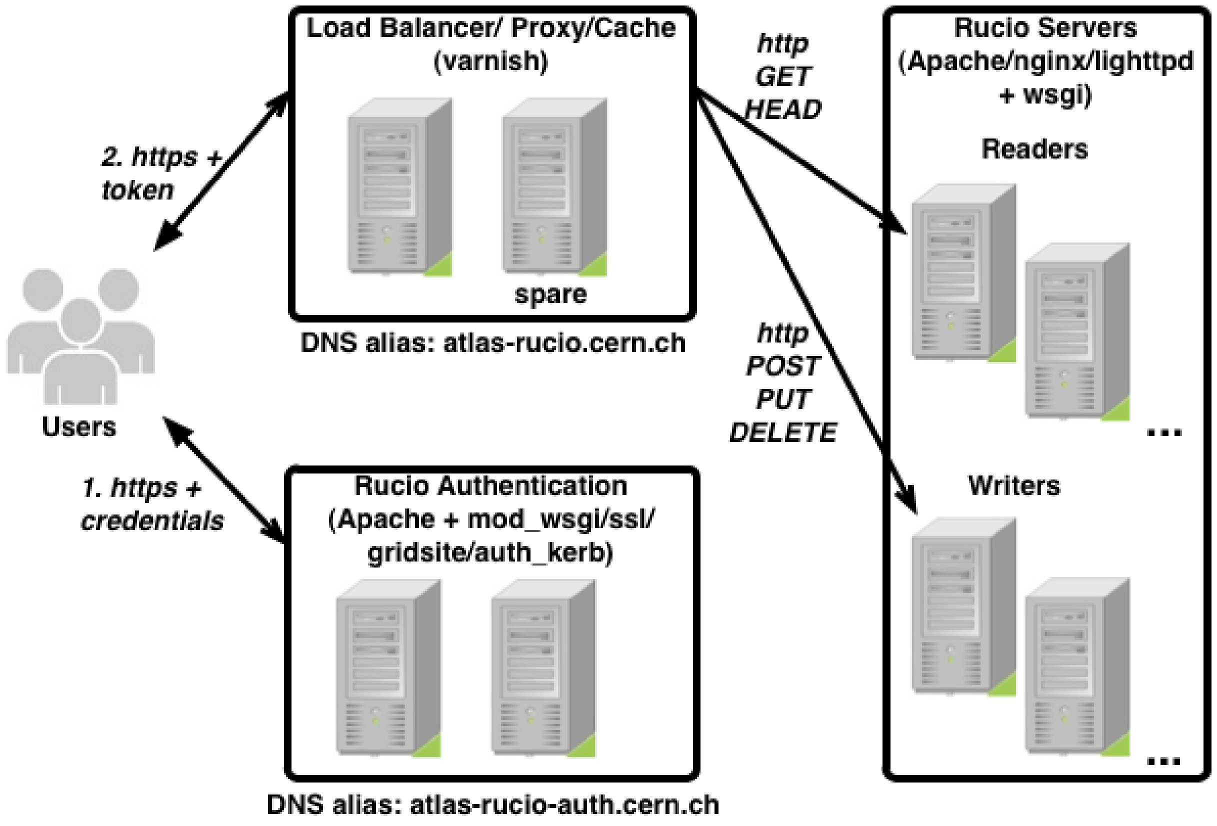
```
    $> cd /opt/rucio
    $> bin/venv_lighttpd.sh
```

5. test:

```
    $> curl -vvv -X GET -H "Rucio-Account: ddmlab" -H "Rucio-Username: xxxxx" -H "Rucio-Password: xx
```

you should get back an HTTP OK with a X-Rucio-Auth-Token in the HTTP header

Server Deployment Model


Load Balancer/ Proxy/Cache (varnish)
spare
DNS alias: atlas-rucio.cern.ch
http GET HEAD
Rucio Servers (Apache/nginx/lighttpd + wsgi)
Readers
...
2. https + token
Users
http POST PUT DELETE
1. https + credentials
Rucio Authentication (Apache + mod_wsgi/ssl/ gridsite/auth_kerb)
Writers
...
DNS alias: atlas-rucio-auth.cern.ch

ATLAS Rucio Integration Test-bed

Authentication

DNS: http://atlas-rucio-auth.cern.ch/

- voatlas298, AWStats
- voatlas299, AWStats

Load Balancer/Proxy/Cache

DNS: http://atlas-rucio.cern.ch/

- voatlas304, Munin monitoring
- voatlas305, Munin monitoring

Reader Backend

- voatlas300, AWStats
- voatlas301, AWStats

Writer Backend

- voatlas302, AWStats
- voatlas303, AWStats

Daemons

- voatlas63

Graphite

- voatlas70, Monitoring

Nagios

- voatlas143, Nagios monitoring

CHAPTER 5

Rucio RESTful API

General notes

Each resource can be accessed or modified using specially formed URLs and the standard HTTP methods:

- GET to read
- POST to create
- PUT to update
- DELETE to remove

We require that all requests are done over SSL. The API supports JSON formats. Rucio uses OAuth to authenticate all API requests. The method is to get an authentication token, and use it for the rest of the requests. Descriptions of the actions you may perform on each resource can be found below.

Date format

All dates returned are in UTC and are strings in the following format (RFC 1123, ex RFC 822):

```
Mon, 13 May 2013 10:23:03 UTC
```

In code format, which can be used in all programming languages that support strftime or strptime:

```
'%a, %d %b %Y %H:%M:%S UTC'
```

SSL only

We require that all requests(except for the ping) are done over SSL.

Response formats

The currently-available response format for all REST endpoints is the string-based format JavaScript Object Notation(JSON). The server answer can be one of the following content-type in the http Header:

```
Content-type: application/json
Content-Type: application/x-json-stream
```

In the last case, it corresponds to JSON objects delimited by newlines(streaming JSON for large answer), e.g.:

```
{ "id": 1, "foo": "bar" }
{ "id": 2, "foo": "baz" }
...
```

Error handling

Errors are returned using standard HTTP error code syntax. Any additional info is included in the header of the return call, JSON-formatted with the parameters:

```
ExceptionClass
ExceptionMessage
```

Where ExceptionClass refers to *Rucio Exceptions*.

Service

- *GET /PING*: Discover server version information
  - Command: rucio ping, method: ping

Authentication

- *GET /auth/userpass*: Retrieve an auth token with an username and password
- *GET /auth/x509*: Retrieve an auth token with a x509 certificate
- *GET /auth/x509_proxy*: Retrieve an auth token with a Globus proxy
- *GET /auth/gss*: Retrieve an auth token with a gss token
- *GET /auth/validate*: Retrieve an auth token with a gss token
- DELETE /auth/{token}: Revoke a token

Rucio account

- POST /accounts/{account_name}: Create account
  - Command: rucio-admin account add, method: add_account
- GET /accounts/{account_name}: Get account information
- PUT /accounts/{account_name}: Update account information
- GET /accounts/{account_name}/usage: Get account usage information
- GET /accounts/{account_name}/limits: Get limits
- PUT /accounts/{account_name}/limits: Set limits for a account and a value
- GET /accounts/{account_name}/rules: Get all rules of the account
- GET /accounts/whoami: Get information about account whose token is used
- GET /accounts/: List available accounts
- DELETE /accounts/{account_name}: Disable account name

RSE (Rucio Storage Element)

- POST /rses/(rse_name): Create a RSE - Command: rucio-admin rse add - Method: add_rse
- GET /rses/{rse_name}: Get RSE information
- GET /rses/: List available RSEs
- DELETE /rses/{rse_name}: Disable a RSE

- GET /rses/{rse_name}/usage: Get RSE usage information
- GET /rses/{rse_name}/usage/history: Get RSE usage information history

RSE attributes

- GET /rses/{rse_name}/attr/: List all keys of the RSE with their respective values
- GET /rses/{rse_name}/attr/{key}: Get the value of the RSE attribute/key
- POST /rses/{rse_name}/attr/{key}: Create an RSE key
- PUT /rses/{rse_name}/attr/{key}: Update the value of a key
- DELETE /rses/{rse_name}/attr/{key}: Remove a key from a RSE

Identity

- POST /accounts/{account_name}/identities/{userpass|x509|gss|proxy}/{identityString}: Grant a {userpass|x509|gss|proxy} identity access to an account
- GET /accounts/{account_name}/identities: List all identities on an account
- GET /identities/{userpass|x509|gss|proxy}/{identityString}/accounts/: List all account memberships of an identity
- DELETE /accounts/{account_name}/identities/{userpass|x509|gss|proxy}/{identityString}: Revoke a {userpass|x509|gss|proxy} identity's access to an account

Scope

- POST /accounts/{account_name}/scopes/{scope_name}: Create a scope
- GET /accounts/{account_name}/scopes/: List available scopes for an account
- GET /scopes/: List/query all scopes with filter parameter lists
- DELETE /accounts/{account_name}/scopes/{scope_name}: Delete a scope from an account

Data identifiers

- GET /dids/: Search data identifiers over all scopes with filter parameters
- POST /dids/{scope_name}/{did}: Create a new data identifier
- GET /dids/{scope_name}/: List all data identifiers in a scope
- DELETE /dids/{scope_name}/{did}: Obsolete a data identifier
- GET /dids/{scope_name}/{did}/rses/: List replicas for a data identifier
- GET /dids/{scope_name}/{did}/: List content of data identifier
- PUT /dids/{scope_name}/{did}/status: Update data identifier status
- GET /dids/{scope_name}/{did}/status: Get data identifier status
- GET /dids/{scope_name}/{did}/rules: List all rules of this did
- GET /dids/{scope_name}/{did}/meta/: List all keys of the data identifier with their respective values

- GET /dids/{scope_name}/{did}/meta/{key}: Retrieve the selected key value pair for the given data identifier
- PUT /dids/{scope_name}/{did}/meta/{key}: Set the value of the key to NULL ?
- DELETE /dids/{scope_name}/{did}/meta/{key}: Remove a key from a data identifier
- PUT /dids/{scope_name}/{did}/meta/{key}: Set the value of the key of a data identifier
- POST /dids/{scope_name}/{did_super}/{did_sub}: Add "sub" data identifier into "super" data identifier

Metadata

- POST /meta/{key}: Create a new allowed key (value is NULL)
- GET /meta/: List all allowed keys with their default values
- POST /meta/{key}: Create a new allowed key with a default value
- DELETE /meta/{key}: Delete an allowed key
- DELETE /meta/{key}/{defaultvalue}: Delete the default value of a key (change the value to NULL)

Replication rule

- POST /rules/: Create a rule on a data identifier
- GET /rules/{rule_id}: Get all the rules associated to a data identifier
- DELETE /rules/{rule_id}: Delete a rule

Subscriptions

- POST /subscriptions/{account_name}/: Register a subscription
- DELETE /subscriptions/{subscription_id}: Delete a subscription
- GET /subscriptions/{subscription_id}: Get subscription info
- GET /subscriptions/: List all subscriptions
- GET /subscriptions/{subscription_id}/rules: Get all rules of this subscription

REST API Examples Using Curl

Below are some examples of The Rucio REST API with Curl. We assume that there is a Rucio server running on the localhost on port 80/443.

Service

GET /PING

Discover server version information.

Responses

- `200 OK`

Example Request

```
$> curl -s  -X GET http://localhost/ping
{"version": "0-37-g7213ca2-dev1350298880"}
```

Authentication

GET /auth/userpass

Requesting a X-Rucio-Auth-Token with curl via username and password.

Responses

- `200 OK`
- `401 Unauthorized`

Example Request

```
$> curl -s -i --cacert /opt/rucio/etc/web/ca.crt  -X GET -H "X-Rucio-Account: root" -H "Rucio-Usernam
HTTP/1.1 200 OK
Date: Mon, 15 Oct 2012 11:37:33 GMT
Server: Apache/2.2.22 (Unix) mod_ssl/2.2.22 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Python/2.7.3 PHP/5.3.15
X-X-X-Rucio-Auth-Token: bad32ab79a1648128b5343b29580d96c
Content-Length: 0
Content-Type: application/octet-stream
```

GET /auth/x509

Requesting a X-Rucio-Auth-Token with curl via a X509 certificate.

Responses

- `200 OK`
- `401 Unauthorized`

Example Request

```
$> curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Account: root" -E /opt/rucio/etc/web/cli
HTTP/1.1 200 OK
Date: Mon, 15 Oct 2012 11:37:33 GMT
Server: Apache/2.2.22 (Unix) mod_ssl/2.2.22 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Python/2.7.3 PHP/5.3.15
X-X-X-X-X-Rucio-Auth-Token: 928c0a4747d346999cfaceac0b4a171d
Content-Length: 0
Content-Type: application/octet-stream
```

GET /auth/gss

Requesting a X-Rucio-Auth-Token with curl via kerberos.

Responses

- `200 OK`
- `401 Unauthorized`

Example Request

GET /auth/x509_proxy

Requesting a X-Rucio-Auth-Token with curl via a Globus proxy.

Responses

- `200 OK`
- `401 Unauthorized`

Example Request

```
$> curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Account: vgaronne" --cert $X509_USER_PRO
HTTP/1.1 200 OK
Date: Mon, 15 Oct 2012 10:58:37 GMT
Server: Apache/2.2.22 (Unix) mod_ssl/2.2.22 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Python/2.7.3 PHP/5.3.15
X-X-X-Rucio-Auth-Token: 10520defe5314ef68677be7a479152ae
Content-Length: 0
Content-Type: application/octet-stream
```

GET /auth/validate

Check the validity of a authentication token. Checking the validity of a token will extend its lifetime by one hour.

Responses

- `200 OK`: the token is valid
- `401 Unauthorized`: The token is not valid

Example Request

```
$> curl -s -i --cacert /opt/rucio/etc/web/ca.crt  -H "X-X-X-Rucio-Auth-Token: $RUCIO_TOKEN" -X GET ht
HTTP/1.1 200 OK
Date: Mon, 15 Oct 2012 11:37:33 GMT
Server: Apache/2.2.22 (Unix) mod_ssl/2.2.22 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Python/2.7.3 PHP/5.3.15
Content-Length: 85
Content-Type: application/octet-stream

{'lifetime': datetime.datetime(2012, 10, 15, 11, 58, 35, 832646), 'account': u'root'}
```

GET /auth/register_api_token

Responses

Example Request

Example Response

Account

POST /accounts/{accountName}

Create account.

Parameters

Name	Type	Description
`accountType`	String	The type of the account (user, group, atlas)

Responses

- `201 Created`: Account created
- `409 Conflict`: Account already exists
- `401 Unauthorized`

Example Request

```
HTTP/1.1 201 Created
```

GET /accounts/{accountName}

Get account information.

Responses

- `200 OK`
- `404 Not Found`

Example Request

```
curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Auth-Token: $TOKEN" -X GET https://localhos
```

Example Response

```
HTTP/1.1 200 OK
Date: Wed, 04 Jul 2012 13:37:04 GMT
Server: Apache/2.2.21 (Unix) mod_fastcgi/2.4.2 mod_ssl/2.2.21 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Pytho
Transfer-Encoding: chunked
Content-Type: application/json

{"status": "active", "account": "jdoe", "deleted": false, "created_at": "2012-07-04T13:37:04", "updat
```

PUT accounts/{accountName}

Update account information

Responses

- `200 OK`
- `404 Not Found`

Example Request

Example Response

```
HTTP/1.1 201 Created
Date: Wed, 04 Jul 2012 13:37:04 GMT
Server: Apache/2.2.21 (Unix) mod_fastcgi/2.4.2 mod_ssl/2.2.21 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Pytho
Transfer-Encoding: chunked
Content-Type: application/octet-stream

Created
```

GET accounts/whoami

Get information about account whose token is used to sign the request.

Responses

- `303 See Other`

Example Request

```
curl -s -i -L --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Auth-Token: $TOKEN" -X GET https://loca
```

Example Response

```
HTTP/1.1 303 See Other
Date: Wed, 04 Jul 2012 13:37:05 GMT
Server: Apache/2.2.21 (Unix) mod_fastcgi/2.4.2 mod_ssl/2.2.21 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Pytho
Location: https://localhost/account/root
Content-Length: 0
Content-Type: text/html

HTTP/1.1 200 OK
Date: Wed, 04 Jul 2012 13:37:05 GMT
Server: Apache/2.2.21 (Unix) mod_fastcgi/2.4.2 mod_ssl/2.2.21 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Pytho
Transfer-Encoding: chunked
Content-Type: application/json

{"status": "active", "account": "root", "deleted": false, "created_at": "2012-07-04T13:36:58", "updat
```

GET accounts/

List available accounts.

Responses

- `200 OK`

Example Request

```
curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Auth-Token: $TOKEN" -X GET https://localhos
```

Example Response

```
HTTP/1.1 200 OK
Date: Wed, 04 Jul 2012 13:37:05 GMT
Server: Apache/2.2.21 (Unix) mod_fastcgi/2.4.2 mod_ssl/2.2.21 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Pytho
Transfer-Encoding: chunked
Content-Type: application/json

["jdoe", "root"]
```

DELETE accounts/{accountName}

Disable an account.

Responses

- `200 OK`

Example Request

```
curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Auth-Token: $TOKEN" -X DELETE https://loca
```

Example Response

```
HTTP/1.1 200 OK
Date: Wed, 04 Jul 2012 13:37:05 GMT
Server: Apache/2.2.21 (Unix) mod_fastcgi/2.4.2 mod_ssl/2.2.21 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Pytho
Content-Length: 0
Content-Type: application/octet-stream
```

Location

POST locations/

Create a location

Parameters

Name	Type	Description
`locationName`	String	The name of the location

Responses

- `201 Created`: Location created
- `409 Conflict`: Location already exists
- `401 Unauthorized`

Example Request

```
curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Auth-Token: $TOKEN" -d '{"location":"MOCK"}
```

Example Response

GET locations/{locationName}

Get location information.

Responses

- `200 OK`

Example Request

```
curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Auth-Token: $TOKEN" -X GET https://localhos
```

Example Response

```
HTTP/1.1 405 Method Not Allowed
Date: Wed, 04 Jul 2012 13:37:05 GMT
Server: Apache/2.2.21 (Unix) mod_fastcgi/2.4.2 mod_ssl/2.2.21 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Pytho
Allow: POST
Transfer-Encoding: chunked
Content-Type: text/html
X-Pad: avoid browser bug

None
```

GET locations/

List available locations.

Responses

- `200 OK`

Example Request

```
curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Auth-Token: $TOKEN" -X GET https://localhos
```

Example Response

```
HTTP/1.1 200 OK
Date: Wed, 04 Jul 2012 13:37:05 GMT
Server: Apache/2.2.21 (Unix) mod_fastcgi/2.4.2 mod_ssl/2.2.21 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Pytho
Transfer-Encoding: chunked
Content-Type: application/json

[]
```

DELETE locations/{locationName}

Disable a location.

Responses

- `200 OK`

Example Request

```
curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Auth-Token: $TOKEN" -X DELETE https://local
```

Example Response

```
HTTP/1.1 405 Method Not Allowed
Date: Wed, 04 Jul 2012 13:37:05 GMT
Server: Apache/2.2.21 (Unix) mod_fastcgi/2.4.2 mod_ssl/2.2.21 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Pytho
Allow: POST
Transfer-Encoding: chunked
Content-Type: text/html
X-Pad: avoid browser bug

None
```

Rucio Storage Element

POST /locations/{locationName}/rses/

Tag a location with a RSE.

Parameters

Name	Type	Description
`rseName`	String	RSE name
`description` (optional)	String	Description of the RSE

Responses

- `201 Created`: Location-RSE created
- `409 Conflict`: Location-RSE already exists
- `401 Unauthorized`

Example Request

```
curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Auth-Token: $TOKEN" -d '{"rseName":"CLOUD_N
```

Example Response

GET locations/{locationName}/rses/

List all RSEs associated to a location.

Responses

- `200 OK`

Example Request

Example Response

GET rses/

List all RSEs.

Parameters

Name	Type	Description
`description` (optional)	String	Description of the RSE

Responses

- `200 OK`
- `404 Not Found`

Example Request

Example Response

DELETE locations/{locationName}/rses/{rseName}

Remove a location from a RSE.

Responses

- `200 OK`

Example Request

Example Response

Identity

POST accounts/{accountName}/identities/

Grant an x509|gss|userpass identity access to an account.

Parameters

Name	Type	Description
`type`	String	x509|gss|userpass
`identity`	String	DN|username|gss user

Responses

- `201 Created`: Account-identity created
- `409 Conflict`: Account-identity already exists
- `401 Unauthorized`

Example Request

Example Response

GET accounts/{accountName}/identities/

List all identities on an account.

Parameters

Name	Type	Description
‘‘type‘ ‘ (optional)	String	x509|gss|userpass

Responses

- `200 OK`

Example Request

Example Response

GET identities/{x509|gss|userpass}/{id}/accounts/

List all accounts an identity is member of.

Responses

- `200 OK`

Example Request

Example Response

DELETE accounts/{accountName}/identities/{x509|gss|userpass}/{id}

Revoke an x509|gss|userpass identity’s access to an account.

Responses

- `200 OK`

Example Request

Example Response

Scope

POST accounts/{accountName}/scopes/

Create a scope within an account.

Parameters

Name	Type	Description
`scopeName`	String	Scope name

Responses

- `201 Created`: Account-scope created
- `409 Conflict`: Account-scope already exists
- `401 Unauthorized`

Example Request

Example Response

GET accounts/{accountName}/scopes/

Get the scopes for an account.

Responses

- `200 OK`

Example Request

Example Response

GET scopes/

List all scopes.

Responses

- `200 OK`

Example Request

```
curl -s -i --cacert /opt/rucio/etc/web/ca.crt -H "X-Rucio-Auth-Token: $TOKEN" -X GET https://localhos
```

Example Response

```
HTTP/1.1 500 Internal Server Error
Date: Wed, 04 Jul 2012 13:37:06 GMT
Server: Apache/2.2.21 (Unix) mod_fastcgi/2.4.2 mod_ssl/2.2.21 OpenSSL/0.9.8r DAV/2 mod_wsgi/3.3 Pytho
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html

AccountNotFound: Account does not exist.
```

```
Details: Account ID 'ddmlab/' does not exist
{}
```

DELETE accounts/{accountName}/scopes/{scopeName}

Delete a scope from an account.

Responses

- `200 OK`

Example Request

Example Response

Dataset

POST datasets/{scopeName}/

Register a dataset.

Parameters

Name	Type	Description
`datasetName`	String	dataset name
`...`	...	...

Responses

- `201 Created`: Dataset created
- `409 Conflict`: Dataset already exists
- `401 Unauthorized`

Example Request

Example Response

GET datasets/{scopeName}/{datasetName}/

List dataset content.

Responses

- `200 OK`

Example Request

Example Response

GET datasets/{scopeName}/{datasetName}

List dataset meta-data.

Responses

- `200 OK`

Example Request

Example Response

PUT datasets/{datasetName}

Update dataset meta-data.

POST datasets/{scopeName}/{datasetName}/

Add file(s) to a dataset.

GET datasets/{scopeName}/{datasetName}/{fileName}

Get file meta-data.

GET datasets/

File

POST /locations/{locationName}/files/

Register a file.

Parameters

Name	Type	Description
`fileName`	String	file name
`...`	...	...

Responses

- `201 Created`: File created
- `409 Conflict`: File already exists
- `401 Unauthorized`

Example Request

Example Response

GET /files/{scopeName}/locations/

List file replicas.

PUT /files/{scopeName}/{fileName}/

Update file meta-data.

GET /files/{scopeName}/{fileName}

GET files/

Search files.

REST API Automatic Generated Documentation

WARNING

The following documentation was generated automatically from a script in tools/generateRESTDocs.py.

This script extract useful information directly from source code, but with a best effort approach.

In any case, this documentation can be helpful, but in some cases will be **PLAIN WRONG**.

In some cases the URLs reported are incomplete, the raised exceptions will contain strange strings and in all the cases the correct return value (which usually is **200 OK** or **203 CREATED**) in case of no error is omitted.

The documentation is organized by source code, so REST_account.html will contain the documentation for the rucio/web/rest/acccount.py source file.

If you found an inconsistency with the behavior of Rucio REST API and this documentation, please don't hesitate in submit a JIRA ticket.

Automatic Generated Docs

account.py

GET /accounts/<account>/attr/
list all attributes for an account.

Example request:

```
GET /accounts/<account>/attr/ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'AccountNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

POST /accounts/<account>/attr/<key>
Add attributes to an account.

Example request:

```
POST /accounts/<account>/attr/<key> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'cannot decode json parameter dictionary'
- 400 Bad Request – 'KeyError': '%s not defined' % str(e
- 400 Bad Request – 'TypeError': 'body must be a json dictionary'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 409 Conflict – 'Duplicate': e.args[0][0]
- 404 Not Found – 'AccountNotFound': e.args[0][0]

DELETE /accounts/<account>/attr/<key>
disable account with given account name.

Example request:

```
DELETE /accounts/<account>/attr/<key> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 404 Not Found – 'AccountNotFound': e.args[0][0]

GET /accounts/<account>/scopes/
list all scopes for an account.

Example request:

```
GET /accounts/<account>/scopes/ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'AccountNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]
- 404 Not Found – 'ScopeNotFound': 'no scopes found for account ID '%s'' % account

POST /accounts/<account>/scopes/<scope>
create scope with given scope name.

Example request:

```
POST /accounts/<account>/scopes/<scope> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 409 Conflict – 'Duplicate': e.args[0][0]
- 404 Not Found – 'AccountNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /accounts/<account>

get account information for given account name.

Example request:

```
GET /accounts/<account> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'AccountNotFound': e.args[0][0]
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

POST /accounts/<account>

create account with given account name.

Example request:

```
POST /accounts/<account> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'cannot decode json parameter dictionary'
- 400 Bad Request – 'KeyError': '%s not defined' % str(e
- 400 Bad Request – 'TypeError': 'body must be a json dictionary'
- 409 Conflict – 'Duplicate': e.args[0][0]
- 401 Unauthorized – 'AccessDenied': e.args[0][0]

- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

DELETE /accounts/<account>

disable account with given account name.

Example request:

```
DELETE /accounts/<account> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 404 Not Found – 'AccountNotFound': e.args[0][0]

GET /accounts/?$/?$

list all rucio accounts.

Example request:

```
GET /accounts/?$/?$ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

GET /accounts/<account>/limits/<rse=None>

get the current limits for an account on a specific RSE

Example request:

```
GET /accounts/<account>/limits/<rse=None> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'RSENotFound': e.args[0][0]

POST /accounts/<account>/identities

Grant an identity access to an account.

Example request:

```
POST /accounts/<account>/identities HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'cannot decode json parameter dictionary'
- 400 Bad Request – 'KeyError': '%s not defined' % str(e
- 400 Bad Request – 'TypeError': 'body must be a json dictionary'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 409 Conflict – 'Duplicate': e.args[0][0]
- 404 Not Found – 'AccountNotFound': e.args[0][0]

GET /accounts/<account>/identities

No doc string

Example request:

```
GET /accounts/<account>/identities HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

DELETE /accounts/<account>/identities

Delete an account's identity mapping.

Example request:

```
DELETE /accounts/<account>/identities HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'cannot decode json parameter dictionary'
- 400 Bad Request – 'KeyError': '%s not defined' % str(e
- 400 Bad Request – 'TypeError': 'body must be a json dictionary'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 404 Not Found – 'AccountNotFound': e.args[0][0]
- 404 Not Found – 'IdentityError': e.args[0][0]

GET /accounts/<account>/rules

Return all rules of a given account.

Example request:

```
GET /accounts/<account>/rules HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'RuleNotFound': e.args[0][0]

GET /accounts/<account>/usage/

Return the account usage of the account.

Example request:

```
GET /accounts/<account>/usage/ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'AccountNotFound': e.args[0][0]
- 401 Unauthorized – 'AccessDenied': e.args[0][0]

GET /accounts/<account>/usage/<rse>

Return the account usage of the account.

Example request:

```
GET /accounts/<account>/usage/<rse> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'AccountNotFound': e.args[0][0]
- 404 Not Found – 'RSENotFound': e.args[0][0]
- 401 Unauthorized – 'AccessDenied': e.args[0][0]

authentication.py

OPTIONS /auth/userpass/userpass

HTTP Success:

Example request:

```
OPTIONS /auth/userpass/userpass HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

GET /auth/userpass/userpass

HTTP Success:

Example request:

```
GET /auth/userpass/userpass HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 401 Unauthorized – 'CannotAuthenticate': 'Cannot authenticate to account %(account
- 500 Internal Server Error – e.__class__.__name__: e.args[0]
- 401 Unauthorized – 'CannotAuthenticate': 'Cannot authenticate to account %(account

OPTIONS /auth/gss/gss

HTTP Success:

Example request:

```
OPTIONS /auth/gss/gss HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

OPTIONS /auth/x509/x509_proxy/x509/x509_proxy

HTTP Success:

Example request:

```
OPTIONS /auth/x509/x509_proxy/x509/x509_proxy HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

GET /auth/x509/x509_proxy/x509/x509_proxy

HTTP Success:

Example request:

```
GET /auth/x509/x509_proxy/x509/x509_proxy HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 401 Unauthorized – 'CannotAuthenticate': 'Cannot get DN'
- 401 Unauthorized – 'CannotAuthenticate': 'Cannot authenticate to account %(account
- 401 Unauthorized – 'CannotAuthenticate': 'No default account set for %(dn
- 401 Unauthorized – 'CannotAuthenticate': 'Cannot authenticate to account %(account

OPTIONS /auth/validate/validate

HTTP Success:

Example request:

```
OPTIONS /auth/validate/validate HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

GET /auth/validate/validate

HTTP Success:

Example request:

```
GET /auth/validate/validate HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 401 Unauthorized – 'CannotAuthenticate': 'Cannot authenticate to account %(account

did.py

GET /dids/<scope>/$

Return all data identifiers in the given scope.

Example request:

```
GET /dids/<scope>/$ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]

GET /dids/<scope>/dids/search

List all data identifiers in a scope which match a given metadata.

Example request:

```
GET /dids/<scope>/dids/search HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 409 Conflict – 'UnsupportedOperation': e.args[0][0]
- 404 Not Found – 'KeyNotFound': e.args[0][0]

POST /dids

No doc string

Example request:

```
POST /dids HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

POST /dids/attachments/attachments

No doc string

Example request:

```
POST /dids/attachments/attachments HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

GET /dids/<scope>/<name>/status/(.+)/(.+)

Retrieve a single data identifier.

Example request:

```
GET /dids/<scope>/<name>/status/(.+)/(.+) HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'ScopeNotFound': e.args[0][0]
- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

POST /dids/<scope>/<name>/status/(.+)/(.+)

Create a new data identifier.

Example request:

```
POST /dids/<scope>/<name>/status/(.+)/(.+) HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 400 Bad Request – 'ValueError': str(e
- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 409 Conflict – 'DuplicateContent': e.args[0][0]
- 409 Conflict – 'DataIdentifierAlreadyExists': e.args[0][0]
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 409 Conflict – 'UnsupportedOperation': e.args[0][0]
- 500 Internal Server Error – 'DatabaseException': e.args
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

PUT /dids/<scope>/<name>/status/(.+)/(.+)

Update data identifier status.

Example request:

```
PUT /dids/<scope>/<name>/status/(.+)/(.+) HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json data parameter'
- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 409 Conflict – 'UnsupportedStatus': e.args[0][0]
- 409 Conflict – 'UnsupportedOperation': e.args[0][0]
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /dids/<scope>/<name>/dids

Returns the contents of a data identifier.

Example request:

```
GET /dids/<scope>/<name>/dids HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

POST /dids/<scope>/<name>/dids

Append data identifiers to data identifiers.

Example request:

```
POST /dids/<scope>/<name>/dids HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 409 Conflict – 'DuplicateContent': e.args[0][0]
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 409 Conflict – 'UnsupportedOperation': e.args[0][0]
- 404 Not Found – 'RSENotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0]

DELETE /dids/<scope>/<name>/dids

Detach data identifiers from data identifiers.

Example request:

```
DELETE /dids/<scope>/<name>/dids HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 409 Conflict – 'UnsupportedOperation': e.args[0][0]
- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 401 Unauthorized – 'AccessDenied': e.args[0][0]

GET /dids

No doc string

Example request:

```
GET /dids HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

GET /dids/<scope>/<name>/files

List all replicas of a data identifier.

Example request:

```
GET /dids/<scope>/<name>/files HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /dids/<scope>/<name>/parents

List all parents of a data identifier.

Example request:

```
GET /dids/<scope>/<name>/parents HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /dids/<scope>/<name>/meta

List all meta of a data identifier.

Example request:

```
GET /dids/<scope>/<name>/meta HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

POST /dids/<scope>/<name>/meta/<key>

Add metadata to a data identifier.

Example request:

```
POST /dids/<scope>/<name>/meta/<key> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 409 Conflict – 'Duplicate': e[0][0]
- 400 Bad Request – 'KeyNotFound': e[0][0]
- 400 Bad Request – 'InvalidMetadata': e[0][0]
- 400 Bad Request – 'InvalidValueForKey': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /dids/<scope>/<name>/rules

Return all rules of a given DID.

Example request:

```
GET /dids/<scope>/<name>/rules HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'RuleNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0]

GET /dids/<scope>/<name>/associated_rules

Return all associated rules of a file.

Example request:

```
GET /dids/<scope>/<name>/associated_rules HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 500 Internal Server Error – e.__class__.__name__: e.args[0]

GET /dids/<guid>/guid

Return the file associated to a GUID.

Example request:

```
GET /dids/<guid>/guid HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0]

identity.py

PUT /identities/<account>/x509

Create a new identity and map it to an account.

Example request:

```
PUT /identities/<account>/x509 HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

PUT /identities/<account>/gss

Create a new identity and map it to an account.

Example request:

```
PUT /identities/<account>/gss HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

lock.py

GET /locks

get locks for a given rse.

Example request:

```
GET /locks HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 500 Internal Server Error – e.__class__.__name__: e.args[0]

GET /locks

get locks for a given scope, name.

Example request:

```
GET /locks HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 500 Internal Server Error – e.__class__.__name__: e.args[0]

meta.py

GET /meta

List all keys.

Example request:

```
GET /meta HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

POST /meta

Create a new allowed key (value is NULL).

Example request:

```
POST /meta HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 409 Conflict – 'Duplicate': e[0][0]
- 400 Bad Request – 'UnsupportedValueType': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /meta

List all values for a key.

Example request:

```
GET /meta HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

POST /meta

Create a new value for a key.

Example request:

```
POST /meta HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 409 Conflict – 'Duplicate': e[0][0]
- 400 Bad Request – 'InvalidValueForKey': e[0][0]
- 400 Bad Request – 'KeyNotFound': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

redirect.py

GET /redirect/<scope>/<name>/?$

Redirect download

Example request:

```
GET /redirect/<scope>/<name>/?$ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

replica.py

GET /replicas/<scope>/<name>/?$

List all replicas for data identifiers.

Example request:

```
GET /replicas/<scope>/<name>/?$ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

POST /replicas/?$/?$/(.+)/(.+)/?$

Create file replicas at a given RSE.

Example request:

```
POST /replicas/?$/?$/(.+)/(.+)/?$ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 409 Conflict – 'Duplicate': e[0][0]
- 409 Conflict – 'DataIdentifierAlreadyExists': e[0][0]
- 404 Not Found – 'RSENotFound': e[0][0]
- 503 Service Unavailable – 'ResourceTemporaryUnavailable': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

PUT /replicas/?$/?$/(.+)/(.+)/?$

Update a file replicas state at a given RSE.

Example request:

```
PUT /replicas/?$/?$/(.+)/(.+)/?$ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 500 Internal Server Error – 'UnsupportedOperation': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

DELETE /replicas/?$/?$/(.+)/(.+)/?$

Delete file replicas at a given RSE.

Example request:

```
DELETE /replicas/?$/?$/(.+)/(.+)/?$ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 404 Not Found – 'RSENotFound': e[0][0]
- 503 Service Unavailable – 'ResourceTemporaryUnavailable': e[0][0]
- 404 Not Found – 'ReplicaNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

POST /replicas/list/?$/list/?$

List all replicas for data identifiers.

Example request:

```
POST /replicas/list/?$/list/?$ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

POST /replicas/getdidsfromreplicas/?$/getdidsfromreplicas/?$

List the DIDs associated to a list of replicas.

Example request:

```
POST /replicas/getdidsfromreplicas/?$/getdidsfromreplicas/?$ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

POST /replicas/badreplicas/?$/badreplicas/?$

Declare a list of bad replicas.

Example request:

```
POST /replicas/badreplicas/?$/badreplicas/?$ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 404 Not Found – 'ReplicaNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

rse.py

GET /rses//

List all RSEs.

Example request:

```
GET /rses// HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 400 Bad Request – 'InvalidRSEExpression': e
- 400 Bad Request – 'InvalidObject': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

POST /rses/<rse>

Create RSE with given name.

Example request:

```
POST /rses/<rse> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter dictionary'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 409 Conflict – 'Duplicate': e[0][0]
- 400 Bad Request – 'InvalidObject': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

PUT /rses/<rse>

Update RSE properties (e.g. name, availability).

Example request:

```
PUT /rses/<rse> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter dictionary'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 409 Conflict – 'Duplicate': e[0][0]
- 400 Bad Request – 'InvalidObject': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /rses/<rse>

Details about a specific RSE.

Example request:

```
GET /rses/<rse> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'RSENotFound': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

DELETE /rses/<rse>

Disable RSE with given account name.

Example request:

```
DELETE /rses/<rse> HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 404 Not Found – 'RSENotFound': e.args[0][0]
- 401 Unauthorized – 'AccessDenied': e.args[0][0]

POST /rses
create rse with given RSE name.

Example request:

```
POST /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter dictionary'
- 400 Bad Request – 'KeyError': '%s not defined' % str(e
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 409 Conflict – 'Duplicate': e[0][0]

GET /rses
list all RSE attributes for a RSE.

Example request:

```
GET /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

DELETE /rses

No doc string

Example request:

```
DELETE /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

GET /rses
List all supported protocols of the given RSE.

Example request:

```
GET /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'RSEOperationNotSupported': e[0][0]
- 404 Not Found – 'RSENotFound': e[0][0]
- 404 Not Found – 'RSEProtocolNotSupported': e[0][0]
- 404 Not Found – 'RSEProtocolDomainNotSupported': e[0][0]
- 404 Not Found – 'RSEProtocolNotSupported': 'No prptocols found for this RSE'

`POST /rses`

Create a protocol for a given RSE.

Example request:

```
POST /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter dictionary'
- 404 Not Found – 'RSENotFound': e[0][0]
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 409 Conflict – 'Duplicate': e[0][0]
- 400 Bad Request – 'InvalidObject': e[0][0]
- 404 Not Found – 'RSEProtocolDomainNotSupported': e[0][0]
- 409 Conflict – 'RSEProtocolPriorityError': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

`GET /rses`

List all references of the provided RSE for the given protocol.

Example request:

```
GET /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'RSENotFound': e[0][0]
- 404 Not Found – 'RSEProtocolNotSupported': e[0][0]
- 404 Not Found – 'RSEProtocolDomainNotSupported': e[0][0]

PUT /rses

Updates attributes of an existing protocol entry. Because protocol identifier, hostname,

Example request:

```
PUT /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter dictionary'
- 400 Bad Request – 'InvalidObject': e[0][0]
- 404 Not Found – 'RSEProtocolNotSupported': e[0][0]
- 404 Not Found – 'RSENotFound': e[0][0]
- 404 Not Found – 'RSEProtocolDomainNotSupported': e[0][0]
- 409 Conflict – 'RSEProtocolPriorityError': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

DELETE /rses

Deletes a protocol entry for the provided RSE.

Example request:

```
DELETE /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 404 Not Found – 'RSEProtocolNotSupported': e[0][0]
- 404 Not Found – 'RSENotFound': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /rses

Get RSE usage information.

Example request:

```
GET /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'RSENotFound': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

PUT /rses

Update RSE usage information.

Example request:

```
PUT /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter dictionary'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /rses

Get RSE usage information.

Example request:

```
GET /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'RSENotFound': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /rses

Get RSE limits.

Example request:

```
GET /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'RSENotFound': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

PUT /rses

Update RSE limits.

Example request:

```
PUT /rses HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter dictionary'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

rule.py

GET /rules

get rule information for given rule id.

Example request:

```
GET /rules HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'RuleNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0]

PUT /rules

Update the replication rules locked flag .

Example request:

```
PUT /rules HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 404 Not Found – 'RuleNotFound': e.args[0][0]
- 404 Not Found – 'AccountNotFound': e.args[0][0]
- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

`POST /rules`

Create a new replication rule.

Example request:

```
POST /rules HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 409 Conflict – 'InvalidReplicationRule': e.args[0][0]
- 409 Conflict – 'DuplicateRule': e.args[0]
- 409 Conflict – 'InsufficientTargetRSEs': e.args[0][0]
- 409 Conflict – 'InsufficientAccountLimit': e.args[0][0]
- 409 Conflict – 'InvalidRSEExpression': e.args[0][0]
- 404 Not Found – 'DataIdentifierNotFound': e.args[0][0]
- 409 Conflict – 'ReplicationRuleCreationTemporaryFailed': e.args[0][0]
- 409 Conflict – 'InvalidRuleWeight': e.args[0][0]
- 409 Conflict – 'StagingAreaRuleRequiresLifetime': e.args[0]
- 409 Conflict – 'InvalidObject': e.args[0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

`DELETE /rules`

Delete a new replication rule.

Example request:

```
DELETE /rules HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 401 Unauthorized – 'AccessDenied': e.args[0][0]
- 404 Not Found – 'RuleNotFound': e.args[0][0]

GET /rules

get locks for a given rule_id.

Example request:

```
GET /rules HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 500 Internal Server Error – e.__class__.__name__: e.args[0]

scope.py

GET /scopes

List all scopes.

Example request:

```
GET /scopes HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

POST /scopes

Creates scope with given scope name.

Example request:

```
POST /scopes HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 409 Conflict – 'Duplicate': e.args[0][0]
- 404 Not Found – 'AccountNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]

GET /scopes

List all scopes for an account.

Example request:

```
GET /scopes HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'AccountNotFound': e.args[0][0]
- 404 Not Found – 'ScopeNotFound': 'no scopes found for account ID '%s'' % account

subscription.py

GET /subscriptions

Retrieve a subscription.

Example request:

```
GET /subscriptions HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'SubscriptionNotFound': e[0][0]

PUT /subscriptions

Update an existing subscription.

Example request:

```
PUT /subscriptions HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 404 Not Found – 'SubscriptionNotFound': e[0][0]
- 400 Bad Request – 'InvalidObject': e[0][0]

POST /subscriptions

Create a new subscription.

Example request:

```
POST /subscriptions HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type:
```

Status Codes

- 400 Bad Request – 'ValueError': 'Cannot decode json parameter list'
- 409 Conflict – 'SubscriptionDuplicate': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0][0]
- 400 Bad Request – 'InvalidObject': e[0][0]

GET /subscriptions

Return all rules of a given subscription id.

Example request:

```
GET /subscriptions HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 404 Not Found – 'RuleNotFound': e.args[0][0]
- 404 Not Found – 'SubscriptionNotFound': e[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0]

GET /subscriptions/<account>/<name=None>/Rules/States

Return a summary of the states of all rules of a given subscription id.

Example request:

```
GET /subscriptions/<account>/<name=None>/Rules/States HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/x-json-stream'
```

Status Codes

- 500 Internal Server Error – e.__class__.__name__: e.args[0]

GET /subscriptions

Retrieve a subscription matching the given subscription id

Example request:

```
GET /subscriptions HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: 'application/json'
```

Status Codes

- 404 Not Found – 'SubscriptionNotFound': e.args[0][0]
- 500 Internal Server Error – e.__class__.__name__: e.args[0]

CHAPTER 6

Rucio Python Client APIs

Rucio Clients

Rucio includes a client class to removes some of the complexity of dealing with raw HTTP requests against the RESTful API.

In the example below, which shows how to instanciate a Rucio Client, we assume that there is a Rucio server running on the localhost on port 80/443.

Some other examples of using Rucio's Client class can be found here: Rucio Clients Examples.

Rucio Clients Examples

Below are some examples of using Rucio's Client class. We assume that there is a Rucio server running on the localhost on port 80/443.

Service

ping

Discover server version information.

```
        __init__
        '''
        self.cacert = config_get('test', 'cacert')
        self.usercert = config_get('test', 'usercert')
        try:
```

Account

create_account

Add an account.

Account Methods

AccountLimit Methods

Scope Methods

Identity Methods

Data Identifier Methods

Meta-data Methods

RSE Classes

Client

Manager

Storage

Replica Methods

Replication Rule Methods

Lock Methods

Subscription Methods

Constants

Rucio Exceptions

Exceptions used with Rucio.

The base exception class is `RucioException`. Exceptions which are raised are all subclasses of it.

exception `rucio.common.exception.`**`AccessDenied`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`AccountNotFound`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`CannotAuthenticate`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ClientParameterMismatch`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ClientProtocolNotSupported`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ConfigNotFound`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ConfigurationError`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`CounterNotFound`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`DataIdentifierAlreadyExists`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`DataIdentifierNotFound`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`DatabaseException`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`DestinationNotAccessible`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`Duplicate`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`DuplicateContent`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`DuplicateRule`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ErrorLoadingCredentials`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`FileAlreadyExists`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`FileConsistencyMismatch`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`FileReplicaAlreadyExists`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`FullStorage`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`IdentityError`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`IdentityNotFound`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InputValidationError`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InsufficientAccountLimit`**`(`**args*, ***kwargs*`)`
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InsufficientTargetRSEs`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InvalidMetadata`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InvalidObject`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InvalidPath`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InvalidRSEExpression`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InvalidReplicationRule`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InvalidRequest`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InvalidRuleWeight`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InvalidType`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`InvalidValueForKey`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`KeyNotFound`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`LifetimeExceptionDuplicate`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`LifetimeExceptionNotFound`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ManualRuleApprovalBlocked`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`MissingClientParameter`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`MissingDependency`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`MissingSourceReplica`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`NameTypeError`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`NoAuthInformation`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RSEAccessDenied`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RSEBlacklisted`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RSEFileNameNotSupported`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RSENotConnected`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RSENotFound`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RSEOperationNotSupported`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RSEOverQuota`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RSEProtocolDomainNotSupported`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RSEProtocolNotSupported`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RSEProtocolPriorityError`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ReplicaNotFound`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ReplicaUnAvailable`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ReplicationRuleCreationTemporaryFailed`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RequestNotFound`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ResourceTemporaryUnavailable`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RucioException`**(**args*, ***kwargs*)
Bases: `exceptions.Exception`

To correctly use this class, inherit from it and define a 'message' property. That message will get printf'd with the keyword arguments provided to the constructor.

exception `rucio.common.exception.`**`RuleNotFound`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`RuleReplaceFailed`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ScopeAccessDenied`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ScopeNotFound`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ScratchDiskLifetimeConflict`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`ServiceUnavailable`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`SourceAccessDenied`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`SourceNotFound`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`StagingAreaRuleRequiresLifetime`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`SubscriptionDuplicate`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`SubscriptionNotFound`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`UnsupportedOperation`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`UnsupportedStatus`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

exception `rucio.common.exception.`**`UnsupportedValueType`**(**args*, ***kwargs*)
Bases: *`rucio.common.exception.RucioException`*

CHAPTER 7

Rucio CLI

Rucio CLI Tools

Rucio provides two command-line tools:

- `man/rucio-admin`: Utility for managing and configuring Rucio.
- `man/rucio`: Utility for interacting with Rucio.

Examples with rucio-admin command-line tools can be found here: Rucio Admin CLI Examples.

Examples with rucio command-line tools can be found here: Rucio CLI Examples.

Rucio CLI Examples

`rucio --version`

Print version information:

```
$> rucio --version
```

`rucio ping`

Discover server version information:

```
$> rucio ping
0-42-gd374efe-dev1350397766
```

`rucio add-replicas`

Add file replicas:

```
$> rucio add-replicas  --lfns vgaronne:Myfile vgaronne:Myfile1 vgaronne:Myfile2 --rses MOCK1 MOCK1 MO
Added file replica vgaronne:Myfile at MOCK1
Added file replica vgaronne:Myfile1 at MOCK1
Added file replica vgaronne:Myfile2 at MOCK1
```

`rucio add`

Add dataset/container and content:

```
$> rucio add --dest vgaronne:MyDataset1 --srcs vgaronne:Myfile vgaronne:Myfile1
Added: vgaronne:MyDataset1

$> rucio add --dest vgaronne:MyDataset2 --srcs vgaronne:Myfile2
Added: vgaronne:MyDataset2

$> rucio add --dest vgaronne:MyContainer1 --srcs vgaronne:MyDataset1 vgaronne:MyDataset2
Added: vgaronne:MyContainer1

$> rucio add --dest vgaronne:MyBigContainer1 --srcs vgaronne:MyContainer1
Added: vgaronne:MyBigContainer1
```

`rucio del`

delete container/dataset or contents:

```
$> rucio del vgaronne:MyDataset1 --from vgaronne:MyContainer1
```

`rucio list`

List container/dataset contents:

```
$> rucio list vgaronne:MyDataset
vgaronne:Myfile
vgaronne:Myfile1
vgaronne:Myfile2

$> rucio list vgaronne:MyBigContainer1
vgaronne:MyContainer1

$> rucio list vgaronne:MyContainer1
vgaronne:MyDataset1
vgaronne:MyDataset2

$> rucio list vgaronne:MyDataset1
vgaronne:Myfile
vgaronne:Myfile1
```

`rucio list-files`

List file contents:

```
$> rucio list-files vgaronne:MyDataset1
vgaronne:Myfile
vgaronne:Myfile1

$> rucio list-files vgaronne:MyBigContainer1
vgaronne:Myfile2
vgaronne:Myfile
vgaronne:Myfile1
```

rucio list-replicas

List file replicas:

```
$> rucio list-replicas vgaronne:Myfile1
vgaronne:Myfile1: MOCK1

$> rucio list-replicas vgaronne:MyDataset
vgaronne:Myfile2: MOCK1
vgaronne:Myfile: MOCK1
vgaronne:Myfile1: MOCK1

$> rucio list-replicas vgaronne:MyBigContainer1
vgaronne:Myfile2: MOCK1
vgaronne:Myfile: MOCK1
vgaronne:Myfile1: MOCK1
```

rucio upload

Upload data into rucio:

```
$> rucio upload --rse MOCK --scope vgaronne --files Myfile4 --dsn Mydataset4
Loading credentials from /Users/garonne/Lab/rucio/etc/rse-accounts.cfg
Loading repository data from /Users/garonne/Lab/rucio/etc/rse_repository.json
Upload**Upload**Upload**Upload**Upload**Upload**Upload**Upload**Upload**Upload**
Sourcefile: ./Myfile4
Target: vgaronne:Myfile4
Trans: /tmp/rucio_rse/vgaronne/a3/44/39/Myfile4
Upload**Upload**Upload**Upload**Upload**Upload**Upload**Upload**Upload**Upload**
download operation for Myfile4 done
```

rucio download

download data from rucio:

```
$> rucio download --dir=/tmp/download  vgaronne:Myfile4
Loading credentials from /Users/garonne/Lab/rucio/etc/rse-accounts.cfg
Loading repository data from /Users/garonne/Lab/rucio/etc/rse_repository.json
Download**Download**Download**Download**Download**Download**Download**Download**Download**Download**
Sourcefile: /tmp/rucio_rse/vgaronne/a3/44/39/Myfile4
Target: /tmp/download/vgaronne/Myfile4
Download**Download**Download**Download**Download**Download**Download**Download**Download**Download**
download operation for vgaronne:Myfile4 done

$> ls /tmp/download/vgaronne/
Myfile4
```

rucio search

Search data identifiers:

To Do

`rucio get-metadata`

To Do

`rucio set-metadata`

To Do

`rucio del-metadata`

To Do

`rucio list-rse-usage`

To Do:

```
$> rucio list-rse-usage MOCK

$> rucio list-rse-usage --history MOCK
```

`rucio list-account-usage`

To Do:

```
$> rucio list-account-usage --history

$> rucio list-account-limits
```

Rucio Admin CLI Examples

The syntax of the Rucio admin command line interface is: rucio-admin <ressource> <command> [args], where ressource can be account,identity,rse,scope,meta.

The –help argument can be used to know the syntax of each commands.

Account

`rucio-admin account add`

Add an account:

```
$> rucio-admin account add vgaronne
Added new account: vgaronne
```

`rucio-admin account del`

Delete an account:

```
$> rucio-admin account del vgaronne
Deleted account: vgaronne
```

`rucio-admin account list`

List accounts:

```
$> rucio-admin account list
root
vgaronne
```

`rucio-admin account show`

List account details:

```
$> rucio-admin account show vgaronne
status     : active
account    : vgaronne
deleted    : False
created_at : 2012-10-16T14:30:04
updated_at : 2012-10-16T14:30:04
deleted_at : None
type       : user
```

`rucio-admin account set-limits`

Set account limits:

```
$> rucio-admin account set-limits --account vgaronne --rse_expr "GROUPDISK AND tier=1" --value 100000
Added new limits to account: vgaronne
```

`rucio-admin account get-limits`

Get account limits:

```
$> rucio-admin account get-limits vgaronne
```

`rucio-admin account del-limits`

Del account limits:

```
$> rucio-admin account del-limits --account vgaronne --rse_expr "GROUPDISK AND tier=1"
```

Identity

`rucio-admin identity add`

Grant a {userpass|x509|gss|proxy} identity access to an account:

```
$> rucio-admin identity add --account vgaronne --id vgaronne@CERN.CH --type gss
Added new identity to account: vgaronne@CERN.CH-vgaronne
```

`rucio-admin list-identities`

List all identities on an account:

```
$> rucio-admin account list-identities vgaronne
Identity: vgaronne@CERN.CH,  type: gss
```

Rucio Storage Element (RSE)

`rucio-admin rse add`

Add a RSE:

```
$> rucio-admin rse add MOCK
Added new RSE: MOCK
```

`rucio-admin rse list`

List RSEs:

```
$> rucio-admin rse list
MOCK
MOCK1
MOCK2
```

`rucio-admin rse set-attr`

Set RSE attribute:

```
$> rucio-admin rse set-attr --rse MOCK --key tier  --value 1
Added new RSE attribute for MOCK: tier-1
```

Set RSE a tag (attribute with value=True):

```
$> rucio-admin rse set-attr --rse MOCK2 --key GROUPDISK  --value True
Added new RSE attribute for MOCK2: GROUPDISK-True
```

`rucio-admin rse get-attr`

Get RSE attribute:

```
$> rucio-admin rse get-attr MOCK
tier: 1
```

`rucio-admin rse del-attr`

Delete RSE attribute:

```
$> rucio-admin rse del-attr --rse MOCK2 --key CLOUD --value CERN
Deleted RSE attribute for MOCK2: CLOUD-CERN
```

Scope

`rucio-admin scope add`

Add scope to an account:

```
$> rucio-admin scope add --account vgaronne --scope vgaronne
Added new scope to account: vgaronne-vgaronne
```

`rucio-admin scope list`

List scopes:

```
$> rucio-admin scope list
vgaronne
```

Meta-data

`rucio-admin metadata add`

Create a new allowed key(with default values if specified):

```
$> rucio-admin metadata add --key --value --type --DItypes
```

`rucio-admin metadata del`

Delete an allowed key or key/value:

```
$> rucio-admin metadata del --key --value --type --DItypes
```

`rucio-admin metadata list`

List all allowed keys with their default values:

```
$> rucio-admin metadata list
```

Rucio CLI tutorial

`Configuration check with 'rucio ping'`

This command connect to server to get the version running on the server. Actually, if there is a configuration problem, this command will fail. In order to everything else in this tutorial works, this command should run smoothly.:

```
$> rucio ping
0.1.33
```

You will probably need to check your etc/rucio.cfg file if the output is more like this:

```
$> rucio ping
Traceback (most recent call last):
  File "/usr/local/bin/rucio", line 1568, in <module>
    result = command(args)
  File "/usr/local/bin/rucio", line 108, in ping
    ca_cert=args.ca_certificate, timeout=args.timeout)
  File "/usr/local/lib/python2.7/dist-packages/rucio/client/pingclient.py", line 24, in __init__
    super(PingClient, self).__init__(rucio_host, auth_host, account, ca_cert, auth_type, creds, timeo
  File "/usr/local/lib/python2.7/dist-packages/rucio/client/baseclient.py", line 157, in __init__
    self.__authenticate()
  File "/usr/local/lib/python2.7/dist-packages/rucio/client/baseclient.py", line 484, in __authentica
    self.__get_token()
  File "/usr/local/lib/python2.7/dist-packages/rucio/client/baseclient.py", line 394, in __get_token
    if not self.__get_token_x509():
  File "/usr/local/lib/python2.7/dist-packages/rucio/client/baseclient.py", line 341, in __get_token_
    raise exc_cls(exc_msg)
rucio.common.exception.CannotAuthenticate: Cannot authenticate.
Details: Cannot authenticate to account testuser with given credentials
```

List available RSEs

Whenever you need to locate, upload, download or delete a replica, often you will need the name of the endpoint affected. To get the list of all the available RSEs, you can try:

```
$> rucio list-rses
AGLT2_CALIBDISK
AGLT2_DATADISK
AGLT2_LOCALGROUPDISK
AGLT2_PERF-MUONS
AGLT2_PHYS-HIGGS
... output omited ...
ZA-UJ_PRODDISK
ZA-UJ_SCRATCHDISK
ZA-WITS-CORE_LOCALGROUPDISK
ZA-WITS-CORE_PRODDISK
ZA-WITS-CORE_SCRATCHDISK
```

You can filter the type of the endpoint just greping the output. For example, to the see all the available scratch disks, this should work:

```
$> rucio list-rses | grep SCRATCHDISK
AGLT2_SCRATCHDISK
AM-04-YERPHI_SCRATCHDISK
ANLASC_SCRATCHDISK
AUSTRALIA-ATLAS_SCRATCHDISK
... output omited ...
WEIZMANN-LCG2_SCRATCHDISK
WUPPERTALPROD_SCRATCHDISK
ZA-UJ_SCRATCHDISK
ZA-WITS-CORE_SCRATCHDISK
```

Add a replica for a file

To upload a replica for a file, you will need the name of the endpoint (RSE name) and the scope (usually, user.<your_login_name>) and the local name of the file.:

```
$> rucio upload --files halpha-spectre.dat --rse PRAGUELCG2-RUCIOTEST_SCRATCHDISK --scope user.jbogad
2014-09-10 15:40:29,714 DEBUG [Looping over the files]
2014-09-10 15:40:29,716 DEBUG [Extracting filesize (27562) and checksum (73cc55fe) for file user.jbog
2014-09-10 15:40:29,955 DEBUG [Using account jbogadog]
2014-09-10 15:40:29,956 INFO [Adding replicas in Rucio catalog]
2014-09-10 15:40:30,090 INFO [Replicas successfully added]
2014-09-10 15:40:30,090 INFO [Adding replication rule on RSE PRAGUELCG2-RUCIOTEST_SCRATCHDISK for the
2014-09-10 15:40:54,257 INFO [Upload operation for [{'adler32': '73cc55fe', 'scope': 'user.jbogadog',
2014-09-10 15:40:57,509 INFO [Will update the file replicas states]
2014-09-10 15:40:59,523 INFO [File replicas states successfully updated]
2014-09-10 15:40:59,523 INFO [Upload successfull]
```

The scope:name pair is called DID. This is a unique identifier for every file or dataset in Rucio Catalog.

Download a replica

To download a replica, you will need the scope and the name of the replica.:

```
$ rucio download user.jbogadog:halpha-spectre.dat
File downloaded. Will be validated
File validated
download operation for user.jbogadog:halpha-spectre.dat done
```

The downloaded file will be in $RUCIO_HOME/<your_scope>. In the example, *$RUCIO_HOME/user.jbogadog/halpha-spectre.dat*

Create a dataset and add files to it

In Rucio, you can create, upload and download Datasets. A Dataset is a container for several files. You can create a Dataset with the following command.:

```
$> rucio add-dataset user.jbogadog:mydataset
Added user.jbogadog:mydataset
```

Note that you always need to refer to the Dataset by scope:name, where 'scope' usually is user.<your_login_name> and 'name' is the name of the Dataset. The previous command creates an open empty Dataset in the Rucio Catalog. You now can add files to it in the following way:

```
$> rucio add-files-to-dataset --to user.jbogadog:mydataset user.jbogadog:hbeta-spectre.dat user.jboga
```

All the files you want to add to a dataset must be previously uploaded to Rucio Catalog.

Now you can see the content of a dataset with the command:

```
$> rucio list-dids user.jbogadog:mydataset
user.jbogadog:halpha-spectre.dat [FILE]
user.jbogadog:hbeta-spectre.dat [FILE]
user.jbogadog:na-spectre.dat [FILE]
```

List files belonging to a scope and it's properties

You can see all the files that belongs to your scope, invoking the command list-dids:

```
$> rucio list-dids
user.jbogadog:halpha-spectre.dat [FILE]
user.jbogadog:hbeta-spectre.dat [FILE]
user.jbogadog:na-spectre.dat [FILE]
user.jbogadog:mydataset [DATASET]
```

Also, you can see the properties of a files using get-metadata command:

```
$> rucio get-metadata user.jbogadog:halpha-spectre.dat
campaign: None
is_new: None
is_open: None
guid: None
availability: None
deleted_at: None
panda_id: None
version: None
scope: user.jbogadog
hidden: False
md5: None
events: None
adler32: 73cc55fe
complete: None
monotonic: False
updated_at: 2014-09-10 13:40:34
obsolete: False
did_type: FILE
suppressed: False
expired_at: None
stream_name: None
account: jbogadog
run_number: None
name: halpha-spectre.dat
task_id: None
datatype: None
created_at: 2014-09-10 13:40:30
bytes: 27562
project: None
length: None
prod_step: None
```

Adding rules for replication

In Rucio, you can add rules to automatically replicate files and datasets. In order to create a new rule for a file or dataset, you can try this:

```
$> rucio add-rule user.jbogadog:halpha-spectre.dat 2 'spacetoken=ATLASSCRATCHDISK'
```

This will add a rule that makes 2 copies of the file 'user.jbogadog:halpha-spectre.dat'. The expression between quotes is a boolean one, that returns a list of possible RSEs in which the files or datasets can be copied. Rucio will automatically select the best option that satisfy the criterion. Other possible expressions are *'tier=3'*, *'cloud=DE'*, *'country=Argentina'*, etc. To see what properties can you use to filter an endpoint, you can run:

```
$> rucio-admin rse get-attribute 'PRAGUELCG2-RUCIOTEST_SCRATCHDISK'
DETIER2S: True
ALL: True
DETIER2DS: True
physgroup: None
country: Czech Republic
spacetoken: ATLASSCRATCHDISK
site: praguelcg2
PRAGUELCG2-RUCIOTEST_SCRATCHDISK: True
cloud: DE
TIER2DS: True
tier: 2
FZKSITES: True
stresstestweight: 1.0
istape: False
```

For more information on rules and how to combine it, you can read the Replication Rules Syntax section.

You can also see all the rules for your files with:

```
$> rucio list-rules --account jbogadog
ID (account) SCOPE:NAME: STATE [LOCKS_OK/REPLICATING/STUCK], RSE_EXPRESSION, COPIES
=================================================================================
2d6472897cb4414786f66c80b7b857d5 (jbogadog) user.jbogadog:halpha-spectre.dat: REPLICATING[0/2/0], "ti
980fcfae20244f3ca147b0d368d800e5 (jbogadog) user.jbogadog:hbeta-spectre.dat: REPLICATING[0/1/0], "PRA
a86be72f7b5c4cfeb9bd700e7a7462cc (jbogadog) user.jbogadog:na-spectre.dat: REPLICATING[0/1/0], "PRAGUE
530e46584b5048b093b97f1d3007fc6b (jbogadog) user.jbogadog:halpha-spectre.dat: REPLICATING[0/1/0], "PR
c356af4fec964f9582ec2c3d6360eded (jbogadog) user.jbogadog:halpha-spectre.dat: REPLICATING[1/1/0], "sp
```

And you can see information about the rule status with:

```
$> rucio rule-info c356af4fec964f9582ec2c3d6360eded
Id:                          c356af4fec964f9582ec2c3d6360eded
Account:                     jbogadog
Scope:                       user.jbogadog
Name:                        halpha-spectre.dat
RSE Expression:              spacetoken=ATLASSCRATCHDISK
Copies:                      2
State:                       REPLICATING
Locks OK/REPLICATING/STUCK: 1/1/0
Grouping:                    DATASET
Expires at:                  None
Locked:                      False
Weight:                      None
Created at:                  2014-09-15 11:06:21
Updated at:                  2014-09-15 11:06:21
Error:                       None
Subscription Id:             None
```

Whenever you delete a rule, if is the only rule over a file, the file is marked to be deleted and eventually will. However, until the file is effectively deleted, will no longer appear in the list-rules nor in the list-dids outputs.:

```
$> rucio delete-rule 980fcfae20244f3ca147b0d368d800e5
Removed Rule
$> rucio list-rules --account jbogadog
ID (account) SCOPE:NAME: STATE [LOCKS_OK/REPLICATING/STUCK], RSE_EXPRESSION, COPIES
=================================================================================
2d6472897cb4414786f66c80b7b857d5 (jbogadog) user.jbogadog:halpha-spectre.dat: REPLICATING[0/2/0], "ti
a86be72f7b5c4cfeb9bd700e7a7462cc (jbogadog) user.jbogadog:na-spectre.dat: REPLICATING[0/1/0], "PRAGUE
```

```
530e46584b5048b093b97f1d3007fc6b (jbogadog) user.jbogadog:halpha-spectre.dat: REPLICATING[0/1/0], "PR
```

If there are other rules over a file, then only the rule is deleted but not the file itself, as you can see in the following example:

```
$> rucio delete-rule 530e46584b5048b093b97f1d3007fc6b
Removed Rule
$> rucio list-rules --account jbogadog
ID (account) SCOPE:NAME: STATE [LOCKS_OK/REPLICATING/STUCK], RSE_EXPRESSION, COPIES
================================================================================
2d6472897cb4414786f66c80b7b857d5 (jbogadog) user.jbogadog:halpha-spectre.dat: REPLICATING[0/2/0], "ti
a86be72f7b5c4cfeb9bd700e7a7462cc (jbogadog) user.jbogadog:na-spectre.dat: REPLICATING[0/1/0], "PRAGUE
```

Appendices

Acronyms and Abbreviations

LFN	Logical File Name.
LRU	Least Recently Used.
DSN	Dataset Name.
PFN	Physical File Name.
RSE	Rucio Storage Element.
UUID	Universal Unique Identifier.
DID	Data IDentifier.
REST	Representational state transfer.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index