# B2Safe module administrator guide

## Table of Contents



Updated to the version 2.3 of the module.

## Architecture

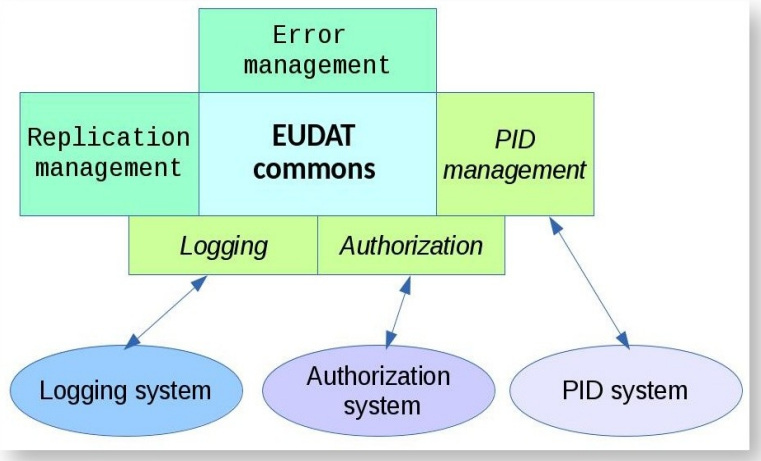


*Figure 1: module architecture*

The architecture of the module is organized into small sub-modules. It includes the following major components:

1. EUDAT commons

It is a rule set containing the basic functions for processing replication and control-files of each single data object.

2. Logging

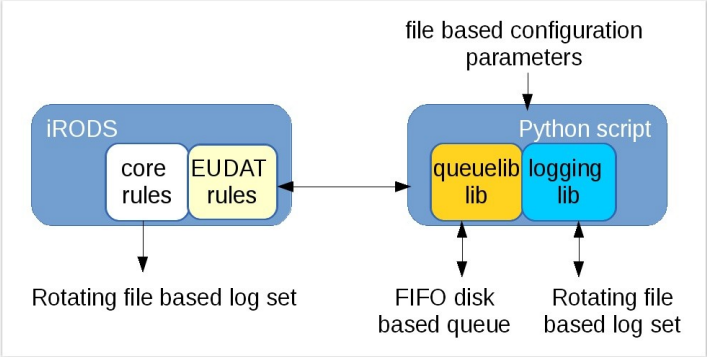


*Figure 2: logging*

It is built on two different data structures: “queue” and “log”, and based on the python-script `/<irods_home>/modules/B2SAFE/cmd/log.manager.py` to do different operations (push, pop, queuesize, log) on the aforementioned data structures. This logging mechanism is used for sorting out only the important information, supporting the monitoring of the data transfers and of the pid management operations. It works as a replication tracking system, which allows to manage the failed transfers in an asynchronous way.

3. Authorization

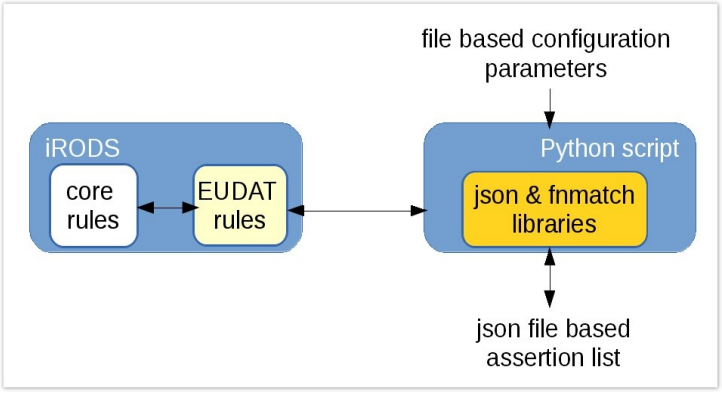


*Figure 3: authorization*

The authorization decision is based on the file "*authz.map.json*", which contains triplets (subject, action, target) called assertions: they represent who are allowed to do what.

4. PID management:

This rule set contains the rules to perform the main operations related to the PID service. It is based on the script `<irods_home>/modules/B2SAFE/cmd/epicclient.py`; which relies on the configuration file `<irods_home>/modules/B2SAFE/conf/credentials`.

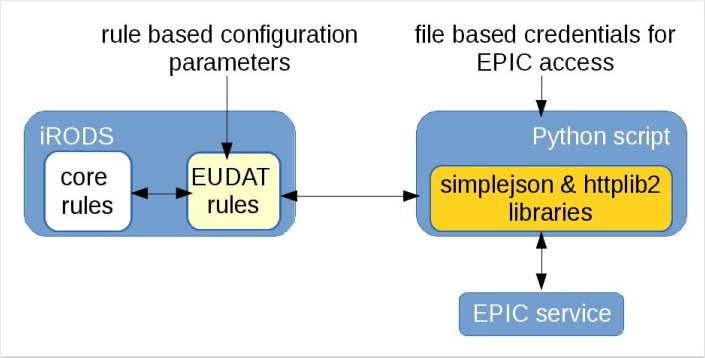


*Figure 4: PID management*

5. Replication Management

This rule set includes functions used to transfer a single data object or a whole collection, together with integrity check mechanisms.

6. Error Management

The last rule set includes functions able to process errors during data transfer. It can detect errors about checksum and size inconsistencies, PID processing and data object ownernership.

### Replica integrity verification

By the integrity verification we mean computing and verifying checksums of replicas of the data objects. (Note: term digest would be more appropriate than checksum, but we follow term used in iRODS). The checksum is computed initially while data ingest (e.g. just after iput) and replication (e.g. in B2SAFE replication is done automatically using iRODS rules) and stored in the iCAT database. The first computed checksum is the referential one and it is also stored in PID service.

The checksums may be also re-computed and verified afterwards at least in two cases:

1. periodically (e.g. every 2 years in order to detect any storage media corruption)
2. on demand (e.g. when the data is migrated to another storage)

In both cases the suitable policy will depend on physical storage (type, configuration, etc.) and availability of computational resources required for re-computing potentially large amount of data, thus it must be decided by data center administrators. This policy is called "local verification policy" later on. The "global" part of replica integrity verification workflow includes exchange of checksum related metadata between PID service and iCATs.

The mechanism of the local verification policy is implemented as shown on the below figure.

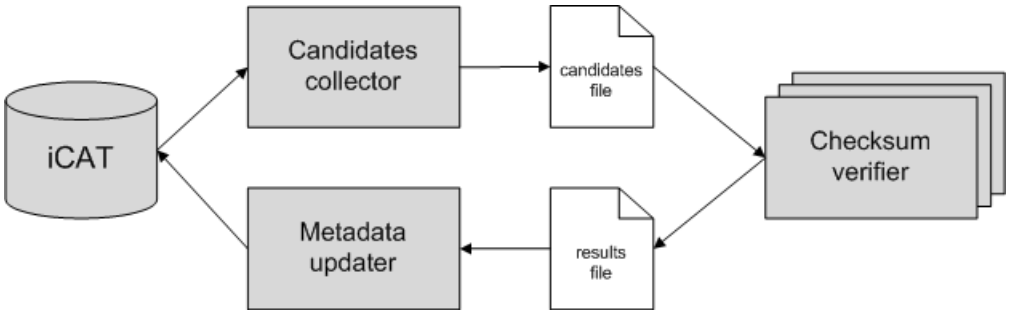

*Figure 5: the local verification flow*

The mechanism consists of the following functional blocks:

- **iCAT** database contains metadata describing location of the physical replicas, checksums and time of the last checksum verification (format of checksum related metadata is described in a following paragraph).
- **Candidates collector** selects candidate data replicas from the iCAT. The selection is based on resource and time of the last verification. This module is implemented as `eudat-get-checksum-verify-candidates` script. The output of the module is **candidates file**.
- **Checksum verifier** performs the verification of replicas taken from candidates file and outputs to the **results file**. The candidates file is removed. This is a storage specific module and thus will have many implementations. It may be run on iRods server or on underlying storage server or partially on both. Two reference implementations will be provided: `eudat-verify-checksums-disc` (to be run on iRods server and local discs) and `eudat-verify-checksums-hsm` (to be run on HSM server on a NFS share provided for iRods server).
- **Metadata updater** reads results file and updates the iCAT metadata (time of verification) of verified objects or reports the problem if the verification failed. The results file is removed. This module is implemented as `eudat-update-checksums` script. The script also calls the rule updatePidChecksum.r (see below), which updates the checksum in the PID record using iRODS rules implemented in the rulebase of the B2SAFE module.

### Authentication

The module in the directory `<irods_home>/modules/B2SAFE/scripts/` includes a set of python scripts which allow to get user information from a remote source and import them in a cache, local to the EUDAT node executing the scripts and eventually into B2Safe.
At present the two steps depicted in the following pictures have been implemented.
It is possible to test them, using the script `remote_users_sync.py`, `user_sync.py` and the related configuration files inside the directory "conf". In the directory "test" there is an example of json file, implementing the local cache. The remote source considered so far is Unity (http://www.unity-idm.eu/), since it is part of the EUDAT AA infrastructure.

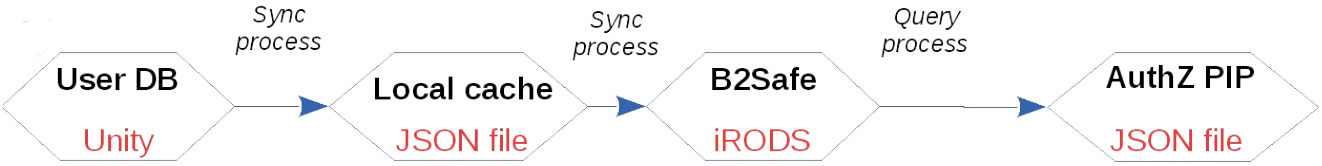

*Figure 6: user account synchronization process*

# Install

See <module_home>/install.txt .

# Configuration

### Authentication

The current authentication system relies on the EUDAT AA infrastructure. In particular it assumes that it possible for the user to get a X.509 (short living) credential and use it to authenticate to the B2SAFE service via GSI mechanism. The aforementioned AuthN scripts allow to accomplish the two preliminary conditions to make it working:

1. import the user account into B2SAFE so that the global (at EUDAT infrastructure level) identity of the user is mapped into a local (at B2SAFE instance level) user identity.
2. import the distinguished name of the user X.509 (short living) credential in order to map such credential to the local B2SAFE user identity.

Therefore the set of scripts inside the directory `<irods_home>/modules/B2SAFE/scripts/authN_and_authZ` as mentioned in the previous paragraph should be configured to perform these two tasks.

- Synchronization with the EUDAT central user DB:

  ```
  $ remote_users_sync.py conf/remote.users.sync.conf syncto EUDAT
  ```

  the configuration file appears like this:

  ```
  # section containing the common options
  [Common]
  logfile=/path/to/remote.sync.log
  # possible values: INFO, DEBUG, ERROR, WARNING
  loglevel=INFO
  usersfile=/path/to/irods.external
  dnsfile=/path/to/irods.DNs.map


  [EUDAT]
  host=https://unity.eudat-aai.fz-juelich.de:8443/rest-admin/v1/
  username=xxxxxxx
  password=xxxxxxx
  carootdn=/CN=test.ca/OU=JSC/O=FZJ/L=Juelich/C=DE
  ns_prefix=eudat_
  ```

  **usersfile**: the file which contains the user information imported from the EUDAT user DB, it is in JSON format.

  **dnsfile**: the file which contains the distinguished names assoicated to the users.

**host**: the URL of the Unity REST admin interface

**username**: Unity administrator username

**password**: Unity administrator password

**carootdn**: DN of the EUDAT CA

**ns_prefix**: optional prefix for local usernames and groups to mitigate the risk of name clashing between names imported from multiple sources.

- Synchronization between the local cache and the B2SAFE service:

```
$ user_sync.py conf/user.sync.conf sync
```

the configuration file appears like this:

```
# section containing the logging options
[Logging]
# possible values: INFO, DEBUG, ERROR, WARNING, CRITICAL
log_level=INFO
log_file=path/to/user.sync.log


# section containing the sources for users and
projects/groups'information
[Sources]
project_file=irods.local
external_file=irods.external
dn_map_file=irods.DNs.map
# set to "False" to define it false, set to "True" to set it to
true
local_authoritative=True
# condition to filter the projects/groups to be added.
# Only triplets are allowed: (attribute, operator, value).
# Operators allowed are <,>,==,!=,<=,>=
# Only numeric attributes and values are supported
# Example: DiskQuota > 0
condition=DiskQuota > 0


# section related to the management of the quota per root
(project) collection inside irods
[Quota]
```

```
# if "False" disable quota management
quota_active=True
# file where are stored the quota info in json format
quota_file=quota.stats.json
# attribute provided by the local system (Sources->project_file)
# which contain the quota limit per project (iRODS root
collections)
quota_attribute=DiskQuota
# allowed: B, KB, MB, GB, TB.
quota_unity=GB


# section containing options for the notification system
[Notification]
# set to "False" to define it false, set to "True" to set it to
true
notification_active=True
notification_sender=xxxxxxxx
notification_receiver=xxxxx@yyyyyy


# section containing options for the GridFTP integrated with
iRODS via DSI library
[GridFTP]
gridftp_active=True
gridmapfile=grid-mapfile
# this is the DN of the X.509 certificate associated to the
GridFTP server
gridftp_server_dn=/C=ZZ/O=YYYY/OU=Host/L=XXXXX/CN=KKKKKK


# section containing options for iRODS operations
[iRODS]
internal_project_list=public,rodsadmin
irods_home_dir="/ZONE_NAME/home/"
# if "False" the home directories for the sub-groups are not
created.
irods_subgroup_home=False
```

```
irods_debug=False
```

In the following description, I will call “project” a group of users, which represents a whole community, a specific project inside a community or involving multiple community, but anyway with an internal administrative consistency. And sub-group a group of users which represents a sub-set of the user of the project and that can not be partitioned in further sub-groups. Thus at present only one level of sub-groups is managed. In the future it will be possible to extend the tool to manage a complex hierarchy of sub-groups if there is the need.

**project_file**: the JSON file which contains the list of projects derived from the local system.

**external_file**: the list of projects coming from external sources, see **usersfile** parameter in the `remote_users_sync.py` description.

**dn_map_file**: the list of distinguished names, see **dnsfile** paramter in the `remote_users_sync.py` description.

**local_authoritative**: if true, during the merging of the project_file list and external file list, only the projects contained in the first one, the local source, will be define into B2SAFE.

**condition**: if defined, only the projects which have an attribute satisfying such condition will be defined into B2SAFE

**quota_active**: if true, the mechanism to calculate and log the quota per project, which is mapped to a single root collection of data within B2SAFE (with, potentially, multiple sub-collections) is enabled. However this mechanism will just report the quota limit, not the quota current usage (see experimental feature description below).

**quota_file**: the JSON file where the quota limits per project will be reported.

**quota_attribute**: the attribute name containing the value of the quota limit in the local system.

**notification_active**: if false, make silent the notification mechanism.

**gridftp_active**: if true, it enables the mechanism to manage the GridFTP interface for the B2SAFE service

**gridmapfile**: the grid-mapfile related to the GridFTP service integrated into the B2SAFE service via the DSI library.

**gridftp_server_dn**: the distinguished name of the GridFTP server certificate

**internal_project_list**: the list of iRODS groups, which are used for administrative reasons and then should not be considered “projects”. They are excluded by the synchronization mechanism.

**irods_home_dir**: this is the iRODS root directory, where all the collections associated to the projects are created.

**irods_subgroup_home**: if false, the home directories of each sub-group is removed, just after creation. This is useful if you want that the sub-groups are meant only for access right management and not as further storage areas where to put data.

**irods_debug**: if true, it enables the debugging of each iRODS operation executed by the tool, which implies a bigger log file.

- Quota per collection management (experimental):

```
quota_stats.py conf/quota.stats.conf compute
```

the configuration file appears like this:

```
{
"stat_output_file": "quota.stats.json",
"storage_space_unity": "GB",
"log_level": "INFO",
"log_file": "quota.stats.log",
"notification_active": "True",
"notification_sender": "xxxxxx",
"notification_receiver": "xxxxxx@yyyyyy",
"internal_project_list": "public,rodsadmin",
"mirrored_projects": "xxxxxxx,zzzzzzz",
"irods_home_dir": "/ZONE_NAME/home/",
"irods_debug": "False"
}
```

Most part of the parameters has the same meaning described above for the other configurations, excepted for the following ones:

**stat_output_file**: the JSON file where are reported the quota usage statistics per project (collection). The file is created by the `user_sync.py` script, then enriched by `quota_stats.py`.

**mirrored_projects**: the projects whose data collection is duplicated inside the same B2SAFE instance for redundancy. It means that the quota limit is doubled too.

### Authorization

- iRODS 3.3.x is required.
- in case you are using the version 3.3.0 please apply the patch in "rsExecCmd.patch" placed the dir "patches", using the unix “patch” command:
  ```
  $ patch < rsExecCmd.patch
  ```
- in the rule file "*eudat.re*": there are two rules called "*EUDATAuthZ*" and "*getAuthZParameters*". The "*EUDATAuthZ*" calls an external python script placed in `<irods_home>/server/bin/cmd` and called "*authZ.manager.py*". Which requires a configuration file placed in `<irods_home>/modules/B2SAFE/conf` and called "*authz.map.json*". The script provides just a couple of methods: "test" and "check", which returns a boolean value of True if the authorization is granted, False otherwise. The authorization decision is based on the file "*authz.map.json*", which contains triplets (subject, action, target) called assertions. So, for example, passing to the script in input a request like:
  ```
  testuser#testzone,
  read,
  /<irods_home>/modules/B2SAFE/conf/credentials
  ```

It will be accepted if the json file contains:

```
    "assertion 1":
      { "subject":
         [ "testuser#testzone" ],
        "action":
         [ "read" ],
        "target":
         [ "/<irods_home>/modules/B2SAFE/conf/credentials" ]
      }
```

Or even:

```
    "assertion 1":
       { "subject":
          [ "*#testzone" ],
         "action":
          [ "read" ],
         "target":
          [ "/<irods_home>/modules/B2SAFE/conf/*" ]
       }
```

Because it supports the wild characters in the same way a shell do.

- in the rulebase file "*core.re*" the hook should be configured using the patch "*corere.patch*" placed in the folder "patches" of the module, using the unix “patch” command:

```
$ patch <  corere.patch
```

- The entry point for rules specific for certain external executables should be called inside the "*EUDATAuthZ*" as fall back.
- The assertions are parsed in sequence till the conditions of the request are matched, hence those remainng after that one who matches are skipped. In other words, the first assertions (sorted top-down) have higher priority than the following ones. So for example, in this case:

```
    "assertion 1":
       { "subject":
          [ "red#testzone" ],
         "action":
          [ "/<irods_home>/server/bin/cmd/analyze" ],
         "target":
          [ "/testzone/projectA/collection1" ]
       }
    "assertion 2":
       { "subject":
          [ "blue#testzone" ],
         "action":
          [ "/<irods_home>/server/bin/cmd/analyze" ],
         "target":
          [ "/testzone/projectA/collection2" ]
       }
```

to the user “blue” will be forbidden to analyze the collection1, but he/she will be allowed to do it with collection2.

### Logging

We are considering here `/<irods_home>/modules/B2SAFE/conf/log.manager.conf`. Just configure the logging level (INFO, DEBUG, ERROR) and the path to the logging directory:

```
{
"log_level": "DEBUG",
"log_dir": "/<irods_home>/modules/B2SAFE/log",
}
```

### Public profile

In order to publish the basic local B2SAFE configuration we need to know for a site:

- whether this one provides a iRODS service.
- which is its local B2SAFE configuration:
  - its iRODS zone
  - the logical iRODS resource to use
  - and a kind of absolute iRODS path, where EUDAT data is stored (*/iRODSZone/home/*...) ..."

The EUDAT registry creg.eudat.eu can serve this purpose. In creg.eudat.eu, there is the "Services" view, where it is possible to select a specific B2SAFE instance page and update its "Service Extension Properties" section.

At the bottom of the page there are the following "Service Extension Properties":

| Name | Value |
|---|---|
| irods_path | e.g. /vzRZGE/eudat/ |
| irods_zone | e.g. vzRZGE |
| irods_resource | e.g. cacheResc |

The "+" sign "Add Property" allows to add properties to the service.

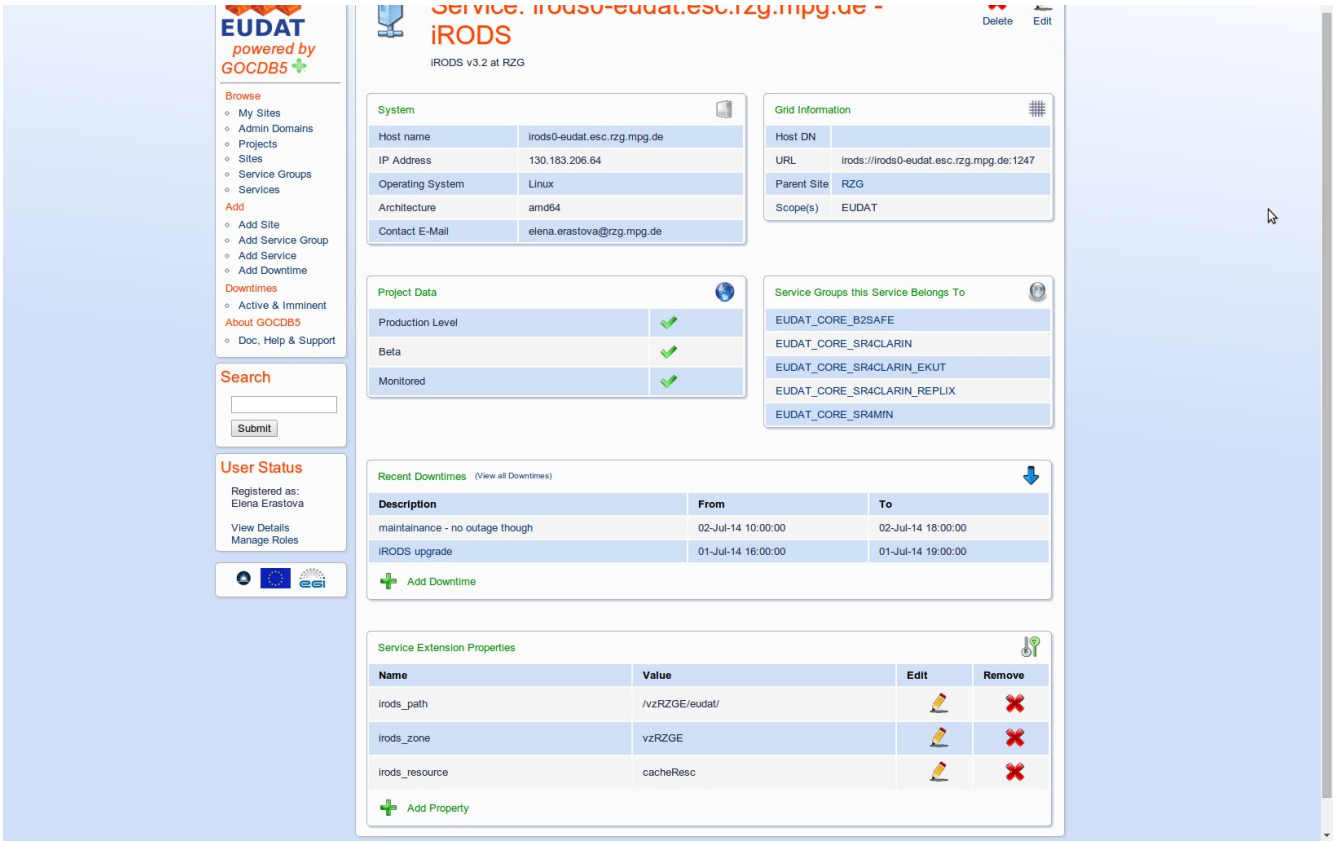


*Figure 7: EUDAT service registry (GOCDB)*

## Replica integrity verification

The scripts described below are place in:
`/<irods_home>/modules/B2SAFE/scripts/integrity` .

### Periodical check

All modules are run from cron daily or several times a day. Option `-d` of `eudat-get-checksum-verify-candidates` script is set to number of days of the period (e.g. 365 if the check is to be performed every year). The mechanism will verify replicas with time of last verification between 365 and 366 days ago.

### On demand check

If check of all replicas from a resource is needed, `eudat-get-checksum-verify-candidates` script must be run by the administrator with option `-d` set to 0. Then the rest of the modules must be run either by the administrator or by cron mechanism.

### Format of metadata

The mechanism requires information about the time of initial checksum computation or time of its last verification. While the reference checksum is an attribute of the data object (logical file), the real checksum (and time of its last computation/verification) is an attribute of each data replica (physical file on the resource). The metadata in iCAT is normally connected to data object, not to the replica, thus the name of the resource is part of the metadata attribute. The format is:

```
attribute: eudat_dpm_checksum_date:<Resource>
value: <date of computation>
units:
```

Where <Resource> is name of the resource where the replica is located and <date of computation> is date of its last computation/verification in seconds since Epoch. E.g.:

```
$ imeta ls -d /tempZone/home/rods/test2/testfile.z26952
AVUs defined for dataObj /tempZone/home/rods/test2/testfile.z26952:
attribute: eudat_dpm_checksum_date:demoResc
value: 1402231096
units:
----
attribute: eudat_dpm_checksum_date:hsmResc
value: 1402231333
units
```

#### Format of checksum verification candidates file

Each line of the file is a single record describing physical file. Each record is a triplet separated with spaces: <path relative to the root of resource> <reference checksum> <date of previous computation in sec since Epoch>. E.g:

```
home/rods/test2/testfile.z26460
sha2:Dw6dY0tp2lvuqMvh0OpfYLMicgGZfGmdzKzCK3G9Kxg= 1402744383
home/rods/test2/testfile.z26500
sha2:VHaQ8qqQEhPOmhbprBjOF/4cA2VSBcj2NScBWlSq3Ms= 1402744383
home/rods/test2/testfile.z26526
sha2:A5td6CeZkhVkBFjSHtifLm9PY7K0176vkkvQNVuyiuU= 1402744383
home/rods/test2/testfile.z26896
sha2:KdjKGXem0GQ8pPKEEvPxrePrc+tYTSBU5SQ0meAoYp8= 1402744383
home/rods/test2/testfile.z26952
sha2:jlVGfGkiPcFMNi5CnMnAjw6+IcPaQ+nxKIHD30vpmiU= 1402744383
```

#### Format of checksum verification results file

Each line of the file is a single record describing physical file. Each record is a triplet separated with spaces: <path relative to the root of resource> <result of verification> <date of current computation in sec since Epoch>. The result is OK if the actual checksum matches the referential one or ERR else. E.g:

```
home/rods/test2/testfile.z25660 OK 1403004246
home/rods/test2/testfile.z26500 OK 1403004246
home/rods/test2/testfile.z26526 OK 1403004246
home/rods/test2/testfile.z26896 ERR 1403004246
home/rods/test2/testfile.z26952 OK 1403004246
```

In the above example checksum of "home/rods/test2/testfile.z26896" doesn't match, the replica might be spoiled and the problem should be comunicated to the administrator.

**Usage of scripts**

**Usage of eudat-get-checksum-verify-candidates**

```
Usage: eudat-get-checksum-verify-candidates [-r <iRods_resource>] [-d
<valid_days>] [-l <log_level>] <candidates_file>
       eudat-get-checksum-verify-candidates -h | --help

The script queries iCAT and gets list of physisical data replicas
that are candidates to verify checksum.
The result is written to <candidates_file>
Only replicas located on <iRods_resource> and those having checksum
verified before <valid_days>.
Amount of logging info is controlled by <log_level>. Allowed values
are 0-6.

Defaults:
  <iRods_resource> = demoResc
  <valid_days> = 365
  <log_level> = 3
```

**Usage of eudat-update-checksums**

```
Usage: eudat-update-checksums [-r <iRods_resource>] [-i
<input_results_dir>] [-l <log_level>] <root_path_of_the resource>
       eudat-update-checksums -h | --help

The script reads files from <input_results_dir> in order of their
creation.
The files contain results of the checksum verification, including
verification time.
The script sets the new time in iCAT.
Amount of logging info is controlled by <log_level>. Allowed values
are 0-6.

Defaults:
  <iRods_resource> = demoResc
  <input_results_dir> = <root_path_of_the resource>/../checksum-
verify-results
  <log_level> = 3
```

**Usage of eudat-verify-checksums-disc**

```
Usage: eudat-verify-checksums-disc [-i <input_candidates_file>] [-o
<output_results_dir>] [-l <log_level>] <root_path_of_the resource>
       eudat-verify-checksums-disc -h | --help
```

```
The script reads input file with candidates to verify checksum
<input_candidates_file>,
recalculates the checksums and writes verification results to a file
in <output_results_dir> directory.
The paths of the candidate files are relative to <root_path_of_the
resource>.
Amount of logging info is controlled by <log_level>. Allowed values
are 0-6.

Defaults:
  <input_candidates_file> = <root_path_of_the resource>/../checksum-
verify-candidates
  <output_results_dir> = <root_path_of_the resource>/../checksum-
verify-results
```

**Usage of eudat-verify-checksums-hsm**

```
TO BE DONE
```

#### updatePidChecksum.r

**updatePidChecksum.r**

```
updatePidChecksum{
    EUDATSearchPID(*path, *pid);
    if ( str(*pid) == "empty" ) {
        EUDATePIDcreate(*path, *pid);
    }
    logInfo("DPM CHECKSUM update related to PID *pid");
    getEpicApiParameters(*credStoreType, *credStorePath, *epicApi,
                         *serverID, *epicDebug);
    msiExecCmd("epicclient.py","*credStoreType *credStorePath modify
               *pid CHECKSUM 'none'", "null", "null", "null", *out);
    EUDATiCHECKSUMget(*path, *checksum);
    msiExecCmd("epicclient.py","*credStoreType *credStorePath modify
               *pid CHECKSUM *checksum", "null", "null", "null",
               *out);
    msiGetStdoutInExecCmdOut(*out, *response);
    logInfo("EUDATeCHECKSUMupdate -> modify handle response =
            *response");
}

INPUT *path=""
OUTPUT ruleExecOut
```

# Changelog

See <module_home>/changelog.txt

# API (EUDAT rules)

**Commons**

EUDATiCHECKSUMretrieve(*path, *checksum)

Get an existent checksum from iCAT

EUDATiCHECKSUMget(*path, *checksum)

Get, if exist or create if not, a checksum from iCAT

EUDATgetObjectTimeDiff(*filePath, *age)

Calculate the difference between the creation time and the modification time of an object (in seconds).

EUDATfileInPath(*path,*subColl)

Check if a file is in a given path

EUDATCreateAVU(*Key,*Value,*Path)

Create a metadata triplet on iCAT

EUDATiDSSfileWrite(*DSSfile)

Create an object containing a list of PIDs related to a specific collection

EUDATiCHECKSUMdate(*coll, *name, *resc, *modTime)

Checks if date of the last computation of iCHECKSUM was set and set the date if not

**Logging**

EUDATLog(*message, *level)

Log an event

EUDATQueue(*action, *message, *number)

Log a failure to a FIFO queue

**Authorization**

EUDATAuthZ(*user, *action, *target, *response)

Authorization policy decision point

**PID management**

EUDATCreatePID(*parent_pid, *path, *ror, *iCATCache, *newPID)

Create PID

EUDATSearchPID(*path, *existing_pid)

Search PID

EUDATSearchPIDchecksum(*path, *existing_pid)

Search PID by checksum

EUDATUpdatePIDWithNewChild(*parentPID, *childPID)

Update PID record field 10320/LOC

EUDATGetRorPid(*pid, *ror)

Get PID record field RoR's value

EUDATeiPIDeiChecksumMgmt(*path, *PID, *ePIDcheck, *iCATuse, *minTime)

**Replication management**

EUDATUpdateLogging(*status_transfer_success, *path_of_transfered_file, *target_transfered_file, *cause)

Log a transfer event to the log file and, if it is a failure, to the FIFO queue

EUDATCheckError(*path_of_transfered_file,*target_of_transfered_file)

Perform error checks about the transfer

EUDATTransferSingleFile(*path_of_transfered_file,*target_of_transfered_file)

Transfer a single file

EUDATTransferUsingFailLog(*buffer_length)

Retry to perform a certain number of failed transfers queued in the FIFO queue

EUDATCheckReplicas(*source, *destination)

Check whether two files are available and identical and trigger replication if they are not

EUDATTransferCollection(*path_of_transfered_coll,*target_of_transfered_coll,*incremental,*recursive)

**Error management**

EUDATCatchErrorChecksum(*source,*destination)

Catch error with Checksum

EUDATCatchErrorSize(*source,*destination)

Catch error Size of file

EUDATProcessErrorUpdatePID(*updfile)

Process error update PID at Parent_PID. It will be processed during replication_workflow, called by updateMonitor.

EUDATCatchErrorDataOwner(*path,*status)

Catch error Data Owner if user is not owner of Data from *path

Create or update a PID, including checksum

EUDATiPIDcreate(*path, *PID)

Create a PID as iCAT metadata

EUDATiFieldVALUEretrieve(*path, *FNAME, *FVALUE)

Get a metadata value from iCAT

EUDATePIDcreate(*path, *PID)

Create a PID as EPIC service record

EUDATePIDsearch(*field, *value, *PID)

Search a PID on the EPIC service

EUDATeCHECKSUMupdate(*PID)

Update the PID record field checksum

EUDATeURLupdate(*PID, *newURL)

Update the PID record field URL

EUDATeURLsearch(*PID, *URL)

Searches about the URL field of the PID

EUDATePIDremove(*path)

Delete a PID

EUDATeiPIDeiChecksumMgmtColl(*sourceColl)

Walk through the collection. For each object, it creates a PID and stores its value and the object checksum in the iCAT.

EUDATiRORupdate(*source, *pid)

Add the ROR field of the PID of the object to iCAT

EUDATeParentUpdate(*PID, *PFName, *PFValue)

Update the EUDAT ROR or PPID field in the PID record

Transfer a whole collection

EUDATIntegrityCheck(*srcColl,*destColl)

Checks all errors between source-Collection and destination-Collection.

EUDATVerifyCollection(*srcColl)

Verify that the object is a collection

# Performance

In order to improve the performance in case of multiple processes requesting iRODS resources is advisable to change the following line in the rule set *core.re*:

```
# 17) acSetReServerNumProc - This rule set the policy for the number of processes
# to use when running jobs in the irodsReServer. The irodsReServer can now
# muli-task such that one or two long running jobs cannot block the execution
# of other jobs. One function can be called:
#    msiSetReServerNumProc(numProc) - numProc can be "default" or a number
#    in the range 1-4. numProc will be set to 1 if "default" is the input.
#
#acSetReServerNumProc {msiSetReServerNumProc("default"); }
```

With the next one:

```
acSetReServerNumProc {msiSetReServerNumProc("4"); }
```

But be aware that on irods 3.3 you need to apply a patch to the file `<irods_home>/server/core/src/reServerLib.c`. Otherwise it will not work properly. For the patches, see irods chat: https://groups.google.com/forum/#!msg/irod-chat/IGYRQdNk9QE/FqcMct5OY-UJ ("intermittend stat errors since the enabling of 4 ruleservers").