

Notes on the Monitoring and Information Service

Matt Leslie, Petr Vokac, Mariano Zimmler, Adam Lyon¹,
Sinisa Veseli

21 July 2004

1 Running the MIS server on d0cs005

Right now, the standard MIS server runs on d0cs005. Below are instructions for starting and stopping the server.

1.1 Starting the MIS server

- Log into d0cs005 as sam
- Check if the MIS server is running with `ps -fwwwu sam | grep -v mysql`
- `cd /home/sam/sam_mis/src/python`
- `source setups`
- `python misServer.py`

Note that the current configuration sends output to your window. You should not close this window without stopping the server first.

1.2 Stopping the MIS server

If you are at a window with the MIS server running in the foreground, put it in the background with,

- Type `control-z` to suspend the foreground server
- Type `bg` to put it in the background
- Proceed with the 5th step below

To stop the MIS server, do

- Log into d0cs005 as sam
- Check if the MIS server is running with `ps -fwwwu sam | grep -v mysql`
- `cd /home/sam/sam-mis/src/python`
- `source setups`

¹ Alert Adam at lyon@fnal.gov if changes are needed for this document.

- `python shutdownClient.py ServerName=SAMMIS@d0cs005 Password=a`

1.3 Clearing the MIS backend database

One can remove all data from the MIS backend database by running the `clearDB.py` script in the `/home/sam/sam-mis/src/python` directory.

2 Running projects on the Test Harness

Please coordinate with Adam if you want to run projects on the test harness.

The test harness runs on `d0srv002`. You can only log in as SAM on that machine.

Whenever you log into `d0srv002`, you **must** do the following steps right away,

- Log into `d0srv002` as `sam`
- `cd /home/sam/cabsrv2th`
- `source sourceMe # This step is very important`

Check that the station is running before starting projects with,
`sam dump station -projects`

If the station is not running, ask Adam to give it a kick.

2.1 Running a project on the test harness

The following set of commands will run one project on the test harness. The project will ask for 10 files, and sleep for 60 seconds after receiving each file. The test harness will automatically make a dataset for you.

- `cd new_test_harness/mist1`
- `python TestHarness.py launchOne.xml`

The `mist1` directory can get filled with lots of stuff. You can safely delete the following files when no projects are running:

- `thlog*.txt`
- `@*.py*`
- `test*.html`
- `sam_batch*`

3 Analyzing the MIS backend DB

The MIS installation is configured to save events to a backend MySQL database.

3.1 Events DB Schema

Event information are spread among three tables.

The *EVENT* table contains basic information about the event:

- *DBID* is the numerical ID of the event. This is assigned by the database in ascending order.
- *ID* is the full text ID of the event as assigned by the event producing application.
- *TYPE* is the category of the event. (e.g. `FileLocationSelectionEvent`)
- *TIME* is the time that the event occurred in “seconds since epoch”. You will need some function to convert it into a real time.
- *PARENTID* is the full text ID of the parent event. Events may have a hierarchical structure (e.g. the station send a `FileTransferEvent` and then spawns an *eworker* process. The events from the *eworker* have the `FileTransferEvent` as their parent).
- *PRODUCERID* is a text ID of the application that sent the event. Only *eworker* seems to be filling this in right now.
- *DICTIONARYID* is a link to the *EVENTDICTIONARY* table for more information about this event.

The *EVENTDICTIONARY* table contains information that was sent with the event.

- *DICTIONARYID* is the ID# used to associate dictionary items with an event. See the *EVENT* table. All dictionary items that come from the same event will have the same *DICTIONARYID*.
- *KEYID* is the link to the *DICTIONARYKEYS* table. Use it to determine the key name for the value you are interested in.
- *VALUE* is the value of the dictionary item.
- *DICTIONARYVALUEID* is used if a “subdictionary” was passed in with the main dictionary. The *DICTIONARYVALUEID* corresponds to more *DICTIONARYIDS* with the sub dictionary information.

The *DICTIONARYKEYS* table contains the key names for dictionary values.

- *KEYID* is the ID# for a particular key. See *EVENTDICTIONARY*.
- *DICTIONARYKEY* is the name of the particular key

3.2 Sample SQL

Below is some SQL that can be used to obtain information from the MIS backend DB.

3.2.1 Information about a particular event

Given an event DBID, return the main EVENT information.

```
select * from EVENT where dbid=19
```

```
DBID          19
ID            SPFTE__EWORKER@d0srv002.fnal.gov:12849__1090278018__2
TYPE         StartPhysicalFileTransferEvent
TIME         1090278018
PARENTID     FTE__EWORKER@d0srv002.fnal.gov:12849__1090278018__1
PRODUCERID   EWORKER@d0srv002.fnal.gov:12849
DICTIONARYID 31
```

To get the dictionary information (printing out the dictionary keys and values) for this event, do

```
select  e.dbid, k.DICTIONARYKEY, v.value,
        v.DICTIONARYVALUEID
from    EVENT e, EVENTDICTIONARY v, DICTIONARYKEYS k
where   e.dbid=19 and
        v.dictionaryid = e.DICTIONARYID and
        v.KEYID=k.keyid
```

#	dbid	DICTIONARYKEY	value	DICTIONARYVALUEID
1	19	EventType	StartPhysicalFileTransferEvent	{null}
2	19	FileName	cosmics_0000140346_002.raw	{null}
3	19	SourceLocation	{null}	32
4	19	TargetLocation	{null}	33
5	19	TargetLocationType	Station	{null}
6	19	SourceLocationType	MSS	{null}

Note that some of the values are null, but these have entries in DICTIONARYVALUEID filled in. That means these entries are sub-dictionaries. To view them, do

```
select e.dbid, k.DICTIONARYKEY, k1.DICTIONARYKEY,
       v1.VALUE
from   EVENT e, EVENTDICTIONARY v, EVENTDICTIONARY v1,
       DICTIONARYKEYS k,
       DICTIONARYKEYS k1
where  e.dbid=19 and
       v.dictionaryid = e.DICTIONARYID and
       k.KEYID = v.KEYID and
```

```
v.dictionaryvalueid is not null and
v1.dictionaryid = v.dictionaryvalueid and
k1.KEYID = v1.KEYID
```

#	dbid	DICTIONARYKEY	DICTIONARYKEY	VALUE
1	19	SourceLocation	SourceVolumeLabel	prj029
2	19	SourceLocation	SourceMSSName	enstore
3	19	SourceLocation	SourceFileOffset	10
4	19	SourceLocation	SourcePath	/pnfs/sam/beagle/copy1/datalogger/initial_runs/datalogger/all/all
5	19	TargetLocation	TargetPath	/sam/cache1/boo
6	19	TargetLocation	TargetNode	d0srv002.fnal.gov
7	19	TargetLocation	TargetStation	fnal-cabsrv2thmis

The first DICTIONARYKEY corresponds to the name of the sub-dictionary. The second is the key for the particular value.

3.3 Doing SQL queries from python

The MySQLdb python module allows you to query a *mysql* database from within python.

If you are logged into d0cs005, in your python script do the following:

```
import MySQLdb

# Make connection to the database
db = MySQLdb.connect(user='sam', passwd='samandeggs', db='SAM')
```

If you are logged into another machine (ask Adam to install MySQLdb if it is not there), then you need to first set up an ssh tunnel to *d0cs005*. Do the following from your shell prompt (you will first need to get a Kerberos root ticket)...

```
ssh -N -L 3307:d0cs005.fnal.gov:3306 sam@d0cs005.fnal.gov &
```

Note that we use port 3307 instead of the default port of 3306 in case your machine has its own mysql server running. Then you set up python with

```
import MySQLdb

# Make connection to the database (note you only specify the port, not
# the host name)
db = MySQLdb.connect(user='sam', passwd='samandeggs', db='SAM',
                    port=3307)
```

In either case, now you use a database *cursor* to perform SQL commands. For example,

```
cu = db.cursor()
```

```
sql = "select * from EVENT where dbid=19"
nrows = cu.execute(sql)
print nrows # Returns 1
```

The `nrows` variable is filled with the number of rows that satisfy the query. To get the result itself, use the `cu.fetchone()` to get one row at a time (each successive call gets the next row). To get all of the rows as a list, do `cu.fetchmany()`.

For example, if you did the calls above and then did,

```
results = cu.fetchmany()
print results
```

The output would be

```
((19L, 'SPFTE__EWORKER@d0srv002.fnal.gov:12849__1090278018__2',
'StartPhysicalFileTransferEvent', 1090278018L,
'FTE__EWORKER@d0srv002.fnal.gov:12849__1090278018__1',
'EWORKER@d0srv002.fnal.gov:12849', 31L),)
```

Note this is a list of a list (if there were more than one row returned, each row would be an entry in the main list).

If you are doing an operation that changes the database (INSERT, UPDATE, DROP, CREATE, DELETE), then you must call `db.commit()` to have those changes take effect. If you haven't committed and want to undo the changes, do `db.rollback()`. Once you commit, you cannot undo the changes.

When you are done working with the database, you should always do `db.close()`

4 Notes on Test Harness configuration

This info is more to remind Adam about the complicated configuration. To run the MIS enabled station on the test harness, the following were configured:

- The station sam configuration is `mis-int`.
- The stager sam configuration is `mis-int-worker`.
- Note that the new style DB server is used, so to set it up, you must do `sam -t -q mis-int` (see the `sourceMe` file).
- Note that special versions of `sam_batch_adapter_pyapi`, `sam_common_pylib`, `sam_idl_pylib`, and `omniORB` are needed. Again, see the `sourceMe` file.
- Note that the station code was placed in `/fnal/ups/prd/sam_station/Linux-2-4/sam_station_mis` (though the ups

db is within /home/sam/cabsrv2th/upsdb). It was put there because the sam_station executables are needed on all machines but there was no room in the home area. It can't stay like that forever.