

TOMUVOL DAQ based on DIM

L.Mirabito

1 Architecture

1.1 PC names and role

The acquisition is based on 15 RaspberryPI micro computers controlling and reading each the 12 DIFs connected on each of the 4 planes: **lyoilcrpiXX** , **withh XX in [03 -13,14,15,19]**. Data are collected and written to disk on the linux PC *lyosdhcal8*.

The 15 RPIs are responsible for chip configurations, DIF control and data readout. **lyoilcrpi17** is also handling the control of the SDCC. The central PC is in charged of the configuration database access, the data collection, the event building and writing and the human interface instance

Since DIM is the network framework used in this data acquisition, the DIM DNS must be stated once **lyosdhcal8** with

```
killall -9 dns
export DIM_DNS_NODE=lyosdhcal8
/opt/xdag/bin/dns &>/dev/null &
```

1.2 DIF control

On each RPI should run a DIM daemon that controls the DIF connected to it. It is started with

```
sudo /etc/init.d/dimdifd start
```

its status is obtained with

```
sudo /etc/init.d/dimdifd status
```

and it can be stoped with

```
sudo /etc/init.d/dimdifd stop
```

The log file is **/var/log/dimdif.log**

It is a DIM server acknowledging the following commands:

- **SCANDEVICES** : List all DIFs connected on USB and publish a DIM Service DEVICES of 255 integers with the identifiers of the found DIFs
- **INITIALISE**: requires the DIF identifier, it opens the DIF devices and tries to write and read the test register. It updates the Dim Services STATE and INFO and returns an INITIALISED state in case of success or INIT_FAILED

- **REGISTERSTATE**: requires the control register (integer) and the current Database state (string). It stores the control register to be used at the next configuration and subscribes to the database DIM server services publishing DIF parameters for each DIF separately (`/DB/STATE_NAME/DIFid`). By this way it obtains from the `dimdb` service the configuration parameter of each DIF.
- **CONFIGURE**: it configures all connected DIF and their associated ASICs. It updates the STATE and the INFO of each DIF. The STATE contains the whole decoding of the slow control status.
- **START**: it starts the acquisition of each DIF. The STATE of each DIF is updated to STARTED.
- **STOP**: it stops the acquisition of each DIF. The STATE of each DIF is updated to STOPPED.

It also publishes 3 services per DIF:

- **STATE**: a string containing the current DIF state (FOUND, INITIALISED, PRECONFIGURED, CONFIGURED, STARTED, STOPPED)
- **INFO**: a structure with *dif id*, *status*, *slow control status*, *bcid*, *gtc*, *bytes read* and *pc name*
- **DATA**: the last DIF data buffer read.

1.3 SDCC control

A DIM daemon is running on the RPI (*clrtorp2-prive*) where the SDCC is connected. It is started with

```
sudo /etc/init.d/dimcccd start
```

its status is obtained with

```
sudo /etc/init.d/dimcccd status
```

and it can be stopped with

```
sudo /etc/init.d/dimcccd stop
```

The log file is `/var/log/dimccc.log`

It acknowledges the following commands:

- **INITIALISE**: requires the CCC FTDI identifier ("DCCCCC01"), it opens the SDCC devices and tries to write and read the test register.
- **CONFIGURE**: it sends DIFRESET and CCCRESET before the start of the acquisition
- **DIFRESET**: it sends a DIFRESET command on the DCC bus
- **CCCRESET**: it send a CCCRESET command on the DCC bus
- **PAUSE**: it send a PAUSE command on the DCC bus

- **RESUME**: it send a RESUME command on the DCC bus
- **START**: it send a START_ACQUISITION command on the DCC bus
- **STOP**: it send a STOP_ACQUISITION command on the DCC bus
- **TESTREGISTERREAD** : read the test register and update the REGISTER DIM service
- **TESTREGISTERWRITE** : requires an integer value that is written in the test register

It publishes 2 DIM services, **REGISTER** that is filled when TESTREGISTERREAD is called and an integer **STATUS** which value is changed depending on the last command received.

1.3.1 Possible issue

At power on, the USB control may be faulty. It is recommended to INITIALISE and use the TESTREGISTERWRITE/READ to check the correct access of the FPGA.

1.4 Database control

The database control is the responsibility of one DIM daemon server running on the central PC (*clrtodaq0-prive*). It is started with

sudo /etc/init.d/dimdbd start

its status is obtained with

sudo /etc/init.d/dimdbd status

and it can be stoped with

sudo /etc/init.d/dimdbd stop

The log file is **/var/log/dimdb.log**

It acknowledges the following commands:

- **DOWNLOAD**: requires the name of the needed DB state ("LPCC_230"), it downloads the given state from the configuration databas and creates for each DIF a DIM service withe name **/DB/STATE_NAME/DIFid** containing the configuration parameters for all the ASICs of the given DIF
- **DELETE**: it deletes all the DIM services of the current state
- **NEWRUN**: it asks the database for a new run number and update **/DB/RUNFROMDB** service

1.5 Data writing

The data writing is achieved by one DIM daemon server running on the central PC (*clrtodaq0-prive*). It is started with

```
sudo /etc/init.d/dimwriterd start
```

its status is obtained with

```
sudo /etc/init.d/dimwriterd status
```

and it can be stopped with

```
sudo /etc/init.d/dimwriterd stop
```

The log file is **/var/log/dimwriter.log**

It acknowledges the following commands:

- **INITIALISE**: it subscribes to all possible DIF DATA and INFO services connected to the DIM DNS. It also subscribes to the RUNFROMDB service from the dimdb server. It creates an *ShmProxy* instance that is spying the */dev/shm* directory for new data collected. The *infoHandler* method of the server received data from the *dimdif* daemon DATA services and write them in */dev/shm*
- **START**: it opens an LCIO file in */data/online/Results* using the current value of run obtained by the RUNFROMDB service. It starts the */dev/shm* spying, collects data and writes them to disk when all publishing DIF have completed the same BCID buffer.
- **STOP**: it stops the spying and closes the data file.

1.5.1 Possible issue

This server is still partially in development. It has to be started after the other daemons in order to be able to collect run number and DIF data. In case of failure, please exit cleanly the daq control (see next section), restart this daemon and restart the daq control

2 DAQ control

The DAQ control is done on the main console login on *lyoilcbpi01*. There is a graphical interface and it will be describe later on but the underlying control is done by the *DimDaqControl* package.

2.1 The DimDaqControl package

The control of the DAQ is done via *DimDaqControl* a standalone DIM program interfaced to python. The python package *LSDHCALDimCtrl* is in */data/online/SDHCAL/DimCtrl*. The environment needs to be set properly with:

```
cd /data/online/SDHCAL/DimCtrl
export LD_LIBRARY_PATH=../lib:$LD_LIBRARY_PATH
python
```

Inside the python shell launched you can do the following:

1. **import LSDHCALDimCtrl** : imports the python package
2. **s=LSDHCALDimCtrl.DimDaqControl("Tomuvol")**: creates an instance of DimDaqControl
3. **s.scandns()**: lists all DIMServer, services and commands available on the DIM DNS
4. **s.initialiseWriter("/data/online/Results")**: Initialise the data writing on the given directory. It has to be done only once if the dimwriter daemon is not stopped.
5. **s.scan()**: sends a SCANDEVICES command to all dimdif servers and wait for DEVICES service update. It must be called once or after a destroy()
6. **s.initialise()**: sends an INITIALISE command to all DIF found previously
7. **s._print()**: prints the status of each DIF on each dimdif server from the INFO service of each given DIF. Like the scan() command it must be called only after a scan() command
8. **s.download("LPCC_230")**: triggers the DELETE of the previous state in the DB server and the DOWNLOAD of the specified state. Since no semaphore mechanism is used in this command a 20 s delay may be needed before the data are available. It has to be called only when a new DB state needs to be downloaded or when the dimdb daemon is restarted
9. **s.registerstate(0x815A1B00,"LPCC_230")**: send a REGISTERSTATE command to all dimdif server. The state specified should be identical to the last downloaded state in dimdb. It must be called before configure to also set the control register.
10. **s.configure()**: sends a CONFIGURE command to all dimdif servers
11. **s.start()**: starts a run, sends a START command to the dimccc, the dimdif and the dimwriter servers
12. **s.stop()**: stops a run,sends a STOP command to the dimccc, the dimdif and the dimwriter servers
13. **s.destroy()**: send a DESTROY command to the dimdif servers. All the DIF handling are cleared locally so the only possible next command is scan()

The red commands needs to be used only once per session. The blue ones should be used once or if any changes has occurred on dimdif servers (power cycle for example). The purple ones are used to reconfigure chips. The green ones are the normal one to start and stop a run. Eventually the black commands are optional, the **download** one must be called at least ones to access the DB but is not needed in a new session if the dimdb server is alive and already contains the state

2.2 The StartDaq package

The *StartDaq* package embedded the *DimDaqControl* and add hosts management, LV and HV control. It can be started directly in python on lyoilcbpi01. It uses a configuration file to handle the host management and set the DB and trigger parameter. An exemple is found in SDHCAL/config_m3.py:

```
host=['lyoilcrpi03','lyoilcrpi04','lyoilcrpi05','lyoilcrpi06','lyoilcrpi07','lyoilcrpi08',
'lyoilcrpi09','lyoilcrpi10','lyoilcrpi11','lyoilcrpi12','lyoilcrpi14','lyoilcrpi15',
'lyoilcrpi19','lyoilcrpi13']
ccc='lyoilcrpi17'
db='lyosdhcal8'
writer='lyosdhcal8'
register=0x89580000
# 115 + petit seup
state="Dome_42chambres_Reference_v4_145"
# Novermber 2012 state="Dome_42chambres_Reference_v4_115"
# November 2012 + masks state="Dome_42chambres_Reference_v4_144"
directory="/data/NAS/December2014"
```

Inside the python shell launched you can do the following:

1. **import StartDaq** : imports the python package
2. **s=StartDaq.StartDaq("config_m3")**: creates an instance of StartDaq withc the config_m3.py files
3. **s.host_start()**: starts DIM daemon on all hosts list in config.host,config.ccc,config.db and config.writer names
4. **s.host_stop()**: stops DIM daemon on all hosts list in config.host,config.ccc,config.db and config.writer names
5. **s.rpi_start()**: starts DIM daemon on all hosts list in config.host
6. **s.rpi_stop()**: stops DIM daemon on all hosts list in config.host
7. **s.synchronizeRPI()**: copy local version (lyoilcbpi01) /opt/dhcal/lib,bin contents to all hosts list in config.host. rpi_stop needs to be called before and rpi_start after

8. **s.initialiseWriter("/data/online/Results")**: Initialise the data writing on the given directory. It has to be done only once if the dimwriter daemon is not stopped.
9. **s.scan()**: sends a SCANDEVICES command to all dimdif servers and wait for DEVICES service update. It must be called once or after a destroy()
10. **s.initialise()**: sends an INITIALISE command to all DIF found previously
11. **s._print()**: prints the status of each DIF on each dimdif server from the INFO service of each given DIF. Like the scan() command it must be called only after a scan() command
12. **s.download("LPCC_230")**: triggers the DELETE of the previous state in the DB server and the DOWNLOAD of the specified state. Since no semaphore mechanism is used in this command a 20 s delay may be needed before the data are available. It has to be called only when a new DB state needs to be downloaded or when the dimdb daemon is restarted
13. **s.registerstate(0x815A1B00,"LPCC_230")**: send a REGISTERSTATE command to all dimdif server. The state specified should be identical to the last downloaded state in dimdb. It must be called before configure to also set the control register.
14. **s.configure()**: sends a CONFIGURE command to all dimdif servers
15. **s.start()**: starts a run, sends a START command to the dimccc, the dimdif and the dimwriter servers
16. **s.stop()**: stops a run, sends a STOP command to the dimccc, the dimdif and the dimwriter servers
17. **s.destroy()**: send a DESTROY command to the dimdif servers. All the DIF handling are cleared locally so the only possible next command is scan()

The red commands needs to be used only once per session. The blue ones should be used once or if any changes has occurred on dimdif servers (power cycle for example). The purple ones are used to reconfigure chips. The green ones are the normal one to start and stop a run. Eventually the black commands are optional, the **download** one must be called at least ones to access the DB but is not needed in a new session if the dimdb server is alive and already contains the state

2.3 Spying with webDid

The latest versions of DIM provide a web interface to all DIM servers and clients. The server is started with:

```
export DIM_DNS_NODE=lyosdhcal8
/opt/dhcal/dimWebDID/webDID
```

Then the interface can be accessed via a web browser at address <http://lyosdhcal8:2500> as seen on figure ??

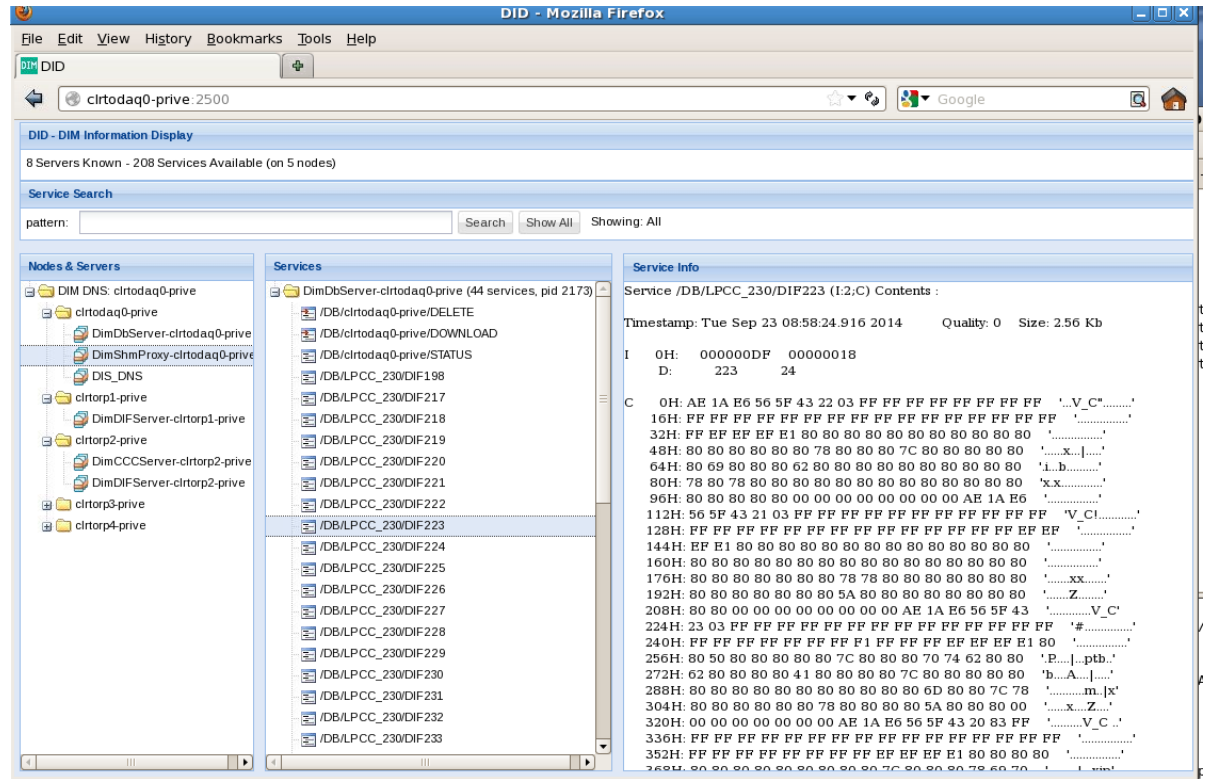


Figure 1: The web application webDID used to spy DIM services

Commands can be sent manually but spying of dynamically allocated infos (like the DID DATA service) should be avoided since it crashes the program.

2.4 Other tips

2.4.1 Restarting the whole daq

In case of problem, after exiting the python shell, one can restart the dns and launch

`~/startdimdaq`

It restarts all needed daemons