DECTRIS
detecting the future



User Manual

PPU – PILATUS Processing Unit

This user manual is valid for the following FURKA and GRIMSEL versions:

FURKA Version: *V.3.7.1*

Grimsel Version: *V.3.7.1*

Document Version: V4

DECTRIS Ltd.
5405 Baden- Daettwil
Switzerland
www.dectris.com

Table of Contents

1. Document History 3

1.1. Current document 3

1.2. Changes 3

2. How to use this Manual 4

2.1. Address and Support 4

2.2. Disclaimer 5

3. PILATUS Processing Unit 6

3.1. Hardware 6

3.2. Software 6

3.2.1. Reinstallation of the Graphics Driver* 6

3.3. Third-Party Software 7

3.4. Time Synchronization with the Detector Server 7

4. FURKA & GRIMSEL 9

4.1. FURKA 9

4.1.1. Installation 9
4.1.2. What's new 11
4.1.3. Running FURKA 12
4.1.4. Configuration 14
4.1.5. Possible Issues 14

4.2. GRIMSEL 15

4.2.1. Installation 15
4.2.2. What's new 16
4.2.3. Running GRIMSEL 17
4.2.4. Adding Files to the GRIMSEL Pipe 19
4.2.5. Interacting with GRIMSEL 19
4.2.6. Possible Issues 26

4.3. Replication of Directory Structures 26

1. Document History

1.1. Current Document

Version	Date	Status	prepared	checked	released
4	24.08.2021	Released	DieG	LisG	DieG

1.2. Changes

Version	Date	Changes
4	23.08.2021	Updated formatting
3	21.08.2019	Udpate of Corporate Design
2	12.10.2016	Network adapters and license key
1	17.12.2015	First version of the document

2. How to use this Manual

Before you start to operate the PPU, please read this User Manual carefully.

2.1. Address and Support

DECTRIS Ltd.
Taefernweg 1
5405 Baden- Daettwil
Switzerland
Phone: +41 56 500 21 02
Fax: + 41 56 500 21 01

Website:

- www.dectris.com → support → Technical Notes → PPU
- www.dectris.com → support → FAQ
- www.dectris.com → support → Problem Report

Email:

- support@dectris.com

Should you have questions concerning the system or its use, please contact us via phone, email or fax.



Do not ship the system back before you receive the necessary transport and shipping information!

2.2. Explanation of Symbols

Symbol	Description
	Important or helpful notice.
	Caution. Please follow the instructions carefully to prevent equipment damage or personal injury.

2.3. Disclaimer

DECTRIS Ltd. has carefully compiled the contents on this manual according to the current state of knowledge. Damage and warranty claims arising from missing or incorrect data are excluded.

DECTRIS Ltd. bears no responsibility or liability for damage of any kind, also for indirect or consequential damage resulting from the use of this system.

DECTRIS Ltd. is the sole owner of all user rights related to the contents of the manual (in particular information, images or materials), unless otherwise indicated. Without the written permission of DECTRIS Ltd. it is prohibited to integrate the protected contents published in these applications into other programs or other Web sites or to use them by any other means.

DECTRIS Ltd. reserves the right, at its own discretion and without liability or prior notice, to modify and/or discontinue this application in whole or in part at any time, and is not obliged to update the contents of the manual.

3. PILATUS Processing Unit

3.1. Hardware

The PILATUS processing unit (PPU) efficiently complements PILATUS detector systems at synchrotron beamlines. It consists of a high-end server, which provides massive processing power, and dedicated software packages to avoid bottlenecks in beamline infrastructure. The PPU is delivered with a graphics card* and a high-speed network card.

PPU key advantages are:

- Highly reliable high-speed data transfer
- Real-time data visualization and manipulation
- Fast crystallographic data processing
- Fast local storage for several weeks of regular user operation

The PPU is available in two configurations PPU XL and a trimmed-down version, the PPU mini. The PPU mini focuses on stable and highly reliable high-speed data transfer but offers no additional computing capacities.

3.2. Software

The PPU comes with CentOS 7.x pre-installed.

The default password for the normal user det is set to Pilatus2.

In addition to the operating system the following software is preinstalled:

- FURKA (copies data from the ramdisk of the detector server to the ramdisk of the PPU)
- GRIMSEL (copies data from ramdisk to disk)
- ALBULA* (viewer for Pilatus images with basic statistic tools)
- XDS* (crystallographic data reduction package, see: http://xds.mpimf-heidelberg.mpg.de/)

3.2.1. Reinstallation of the Graphics Driver*

The driver for the graphics card depends on the current kernel. Do not update the kernel unless strictly necessary, and reinstall the driver after a kernel update. The graphics driver is best installed at runlevel 3. In order to boot the PPU at runlevel 3 follow the steps below:

- Reboot the PPU
- At the grub prompt select the recent kernel (presumably the uppermost)
- Press 'e' (Edit)
- Select the line starting with "Kernel"
- Press 'e'
- Add '3' to the line and press Enter
- Press 'b' (Boot)

* Does not apply to PPU mini

After booting log in as root, go to directory `/var/lib/nvidia-drivers-installer/installer` and run the file `./installer`. Reboot after the installation.

3.3. Third-Party Software*

The third-party software XDS is proposed for crystallographic data processing. The architecture of the PPU allows taking full advantage of the multi-threading capabilities of XDS. Keeping all data and computations in ramdisk eliminates the bottleneck of disk I/O and results in superior data processing performance.

XDS comes pre-installed and ready to use. The XDS binaries are located in the directory `/usr/local/bin`.

For full XDS documentation and literature please refer to http://xds.mpimf-heidelberg.mpg.de/.

Please note: XDS is free of charge for non-commercial applications. For industrial usage of XDS a license is required (e-mail enquiry: Wolfgang.Kabsch@mpimf-heidelberg.mpg.de).

3.4. Time Synchronization on the Detector Server

To synchronize time between the detector server and the PPU, a cronjob has to be set up on the detector server that synchronizes system time every hour with the PPU.

Change the /etc/ntp.conf to match your setup. Example using the default PPU IP (10.10.10.10). Note, it is important to comment or delete the public ntp server entries so they are not being used!

```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool
(http://www.pool.ntp.org/join.html).
#server 0.centos.pool.ntp.org iburst
#server 1.centos.pool.ntp.org iburst
#server 2.centos.pool.ntp.org iburst
#server 3.centos.pool.ntp.org iburst
server 10.10.10.10 iburst
```

As root start the ntpd service

```
service ntpd start
chkconfig ntpd on
```

Create a crontab entry to run the update automatically.

```
crontab -e
```

Add following entry and save (again using the default PPU IP)

```
00 * * * * /usr/sbin/ntpdate -u 10.10.10.10
```

* Does not apply to PPU mini

In case of a reboot, the time offset after booting might be too large. In that case, please execute the command below on the detector server:

`/sbin/ntpdate -u <PPU IP>`, *e.g.*

```
/sbin/ntpdate -u 10.10.10.10
```

4. FURKA & GRIMSEL

FURKA and GRIMSEL are designed to synchronize data between several detector servers and several storage locations. The data flow is illustrated in Fig. 1. FURKA copies data from the detector servers to the PPU ramdisk, GRIMSEL copies the same data from ramdisk to the PPU disk (Internal Storage) and to an external disk (if configured). In the following sections FURKA and GRIMSEL are described in detail.

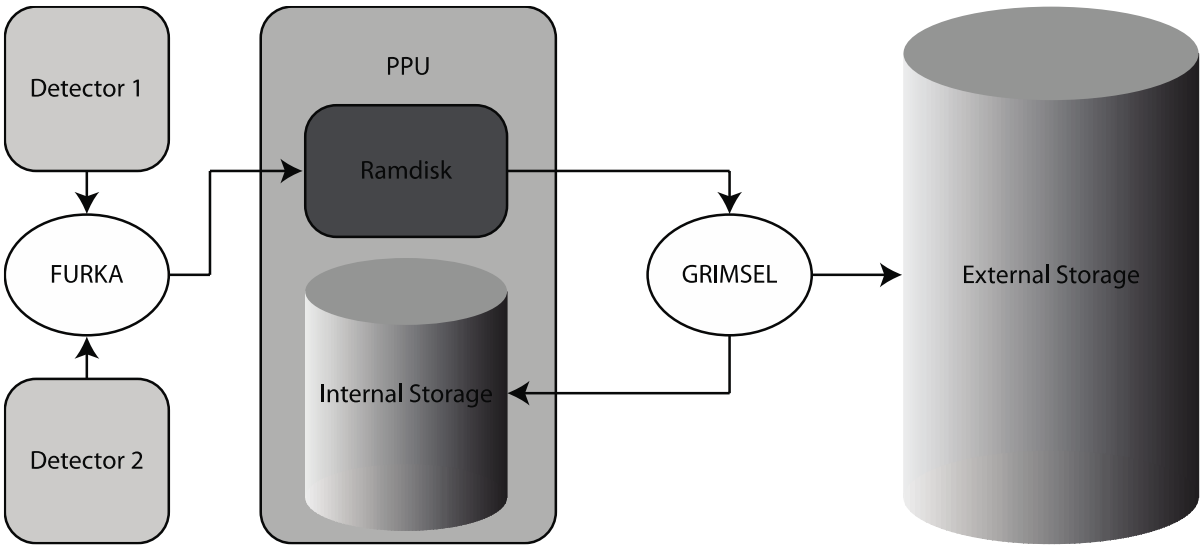


Figure 1: Data flow of FURKA and GRIMSEL.

4.1. FURKA

4.1.1. Installation

4.1.1.1. Remarks

Data acquired with Pilatus detectors are written to a ramdisk (tmpfs) on the detector server. FURKA is intended to copy the data from this ramdisk to the PPU, where it is stored on another ramdisk for further processing.

The PPU and the detector server come with FURKA preinstalled. New installations of FURKA are strongly discouraged. If FURKA needs to be reinstalled, please install individual rpm packages for the detector server and the PPU via yum.

Detector server:

- Become root
  `yum install furka-ppu-server-<version>.rpm furka-ppu-rsync-<version>.rpm`
- If you use a firewall on the detector server, make sure Port 49371 is open

PPU:

- Become root
  yum install furka-ppu-client-<version>.rpm furka-ppu-rsync-<version>.rpm

- Edit `/etc/furka_dectris.conf` (see section Configuration) and adapt license key and remoteIP... The license key is located on `/var/lib/furka/license.key`. Without a license file furka_dectris will run in single-thread mode.

4.1.2. What's new

FURKA 2.0

- FURKA is multi-node capable, i.e. it can copy data from several detector servers to the PPU.
- Improved communication with GRIMSEL

FURKA 3.0

- Minor adaptions for compatibility with GRIMSEL 3.0.

FURKA 3.1

- Minor adaptions for compatibility with GRIMSEL 3.1

FURKA 3.2

- Minor adaptions for compatibility with GRIMSEL 3.2

FURKA 3.3

- Latency period for image transfers to PPU decreased from ~4s to ~0.2s
- Configuration parameter minpause defined in ms (recommended value 10)

FURKA 3.4

- Configure option*: usrdir* ("*uid_gid*") may be configured

FURKA 3.5

- Bugfix in *usrdir* option
- Minor adaptions for compatibility with GRIMSEL 3.5

FURKA 3.6

- Minor adaptions for compatibility with GRIMSEL 3.6

FURKA 3.7

- RPM based installers

FURKA 3.7.1

- Bug fixes
- Changed path of the license key to `/var/lib/furka/license.key`
- CentOS 7 support

4.1.3. Running FURKA

4.1.3.1. Remarks

For highest speed and stability, Pilatus detectors should write all data to ramdisk on the detector server (`/ramdisk` is preconfigured). FURKA is intended to copy the data from this ramdisk to a PPU, where it is stored on another ramdisk for further processing. In the PPU target directory, the directory uid_gid is created (depending on the configuration. See usrdir option in Configuration), to which the data from the detector server is copied. uid, gid and time stamp of the copied files are set according to the FURKA settings.

To guarantee proper file access to the correct users, it is strongly recommended to restart FURKA after every user change or after each beamline shift (see starting options below).

4.1.3.2. Detector Server

After the installation, a specially compiled version of rsync should already run. You can start, stop and restart the rsync daemon via the command

```
sudo /etc/init.d/rsyncd_dectris start|stop|restart
```

4.1.3.3. PPU

Control FURKA via the command (with administrator privileges):

/etc/init.d/furka ((start|restart) [<start options>]) | stop | status

- start options:
  - `-d | --detector <detID>:<source>:<target>`
    - detID: Detector ID for the detector as defined in the config file `/etc/furka_dectris.conf`
    - source: Path on the detector server relative to `/ramdisk`. Default is `/` which means `/ramdisk`.
    - target: Path to the location, in which data is copied. Default is `/ramdisk`. In target the directory uid_gid is created where the data is copied. A '/./' within target does not change the behavior of FURKA. It tells GRIMSEL which directory structure of target should be replicated (confer Section 4.3).
  - `-u | --user <uid>:<gid>` Set the uid and gid of the copied files. Default is 500
- status: Query status of FURKA, in particular number of processes and validity of the license.

Example:

Let us assume we have the detectors 1MF and 2MF, which write their data to `<1MF-server>://ramdisk/` and `<2MF-server>://ramdisk/` respectively. Calling

```
/etc/init.d/furka restart \
-d 1MF:/:/ramdisk/1mf \
-d 2MF:/:/ramdisk/2mf \
-u 500:501
```

will restart FURKA in such a way that all detector 1MF files (excluding empty directories) are copied from

`<1MF-server>://ramdisk/`

to

`<processing-server>://ramdisk/1mf/500_501/`,

whereas the files of detector 2MF are copied to

`<processing-server>://ramdisk/2mf/500_501/`.

The uid (gid) of the copied files are set to 500 (501).

4.1.4. Configuration

The following parameters can be set globally in the configuration file of FURKA `/etc/furka_dectris.conf`:

- logfile: The path and file-name of the FURKA log-file.
- logfile_max_number: This number states how many old log-files are kept.
- logfile_max_size: Maximum size of log-file. Whenever maximum size is reached, FURKA creates a new log file.
- uid, gid: uid and gid of the copied files.
- usrdir: Directory that is created below target at FURKA startup (i.e. at the begin of a user shift). All files are copied there. May contain the strings $UID and $GID that are replaced by uid and gid. If left empty no user directory is created and all files are directly copied to target.
- licensekey: The license key that is needed to run FURKA in multi-thread mode.

For every detector `/etc/furka_dectris.conf` contains a section beginning with the line

detectorID <Detector Identifier>

where <Detector Identifier> is an arbitrary string (without spaces) that identifies the detector. In the lines following the keyword detectorID the parameters below are set:

- remoteIP: IP address of the detector server.
- source: Directory relative to `<detector-server>://ramdisk/` from where the data is copied (usually /).
- target: Directory to where the data is copied (usually /ramdisk).
- minpause: Set to 10
- interrun_pause: Should be set to 500 for all detectors, which are read out via DCB interface (Gigastar), and to 0 if the detector is read out via DCBe interface (ethernet).

4.1.5. Possible Issues

- Restarting rsyncd_dectris on the detector server may block FURKA because of broken communication between FURKA and rsyncd_dectris. Hence restart FURKA whenever rsyncd_dectris has been restarted, e.g. after a reboot of the detector server.
- The FURKA and GRIMSEL license is tied to one of the network interfaces. To ensure the validity of the license keys make sure that all network interfaces remain activated.

4.2. GRIMSEL

4.2.1. Installation

4.2.1.1. Remarks

The PPU comes with GRIMSEL preinstalled. New installations of GRIMSEL are strongly discouraged. If GRIMSEL needs to be reinstalled, please consider the following points:

GRIMSEL only works in combination with FURKA. Hence make sure FURKA is properly installed on the PPU. Secure the old configuration file of GRIMSEL, which will be replaced by a new one.

4.2.1.2. Installation

- Become root
  `yum install grimsel-<version>.rpm`
- You may reuse the license file of FURKA (if `/var/lib/furka/license.key` does not exist, obtain the path from the FURKA configuration file `/etc/furka_dectris.conf`).

4.2.2. What's new

GRIMSEL 2.0

- Time synchronization of PPU and detector server is no longer needed to relate the data to the right user.
- Improved communication with FURKA.
- Performance gains in copying data and link generation.

GRIMSEL 3.0

- GRIMSEL copies data to several storage spaces

GRIMSEL 3.1

- Python interface for querying copy status and reaction on messages

GRIMSEL 3.2

- Bug fixes
- Performance increase

GRIMSEL 3.3

- Minor adaptions for compatibility with FURKA 3.3

GRIMSEL 3.4

- Minor adaptions for compatibility with FURKA 3.4

GRIMSEL 3.5

- Python interface works with python 2.6
- New configure option: create_links
- New configure option: preserve_uid_gid

GRIMSEL 3.6

- Bugfix: Applies to confguration create_links set to NO. If the external storage is faster than the internal storage, files from the internal storgage might not be deleted properly.
- *Configuration option preserve_uid_gid has been replaced by preserve_uid and preserve_gid*!

GRIMSEL 3.7

- RPM based installers

GRIMSEL 3.7.1

- Bug fixes
- Changed path of the license key to `/var/lib/gimsel/license.key`
- CentOS 7 support

4.2.3. Running GRIMSEL

4.2.3.1. Remarks

Grimsel permanently copies image data written by FURKA from the FURKA target directory (in the following called */ramdisk*) to the local PPU disk and depending on the configurartion to external storage. Note, any additional data created on */ramdisk* is not copied! uid, gid and time stamp of the copied files are preserved. GRIMSEL removes files from */ramdisk* whenever a user change takes place, *i.e.* whenever FURKA is restarted. GRIMSEL will delete all files belonging to the old user as soon as they have been copied to the local disk on the PPU. The idea behind that is that whenever a user change takes place, the data of the former user is no longer needed on */ramdisk*. This particular behavior of Grimsel makes it necessary to *regularly restart FURKA after each user change to prevent /ramdisk from overflowing*. When GRIMSEL notices */ramdisk* (or the directory to which the variable ramdisk_mountpoint in the configuration file is set to) is likely to overflow, it converts files on */ramdisk* into symbolic links of the files on the disk. It is strongly recommended to restart FURKA after every user change or after each beamline shift, though.

GRIMSEL needs exactly one internal storage device, which is conveniently the local PPU disk. In addition at most one external and several optional storage devices may be set up in the configuration file (see section Configuration below). If external storage is defined, the internal PPU storage is cleared after a user change and files on the internal storage are eventually turned into symbolic links to the external storage. Without external storage it is the beamline administrator's responsibility to keep the PPU storage clean.

All storage devices must be mounted locally at the PPU. GRIMSEL assumes that the devices are always available. This might be an issue for external USB drives.

4.2.3.2. Start and stop GRIMSEL

Control GRIMSEL via the command (administrator privileges):

```
/etc/init.d/grimsel start|stop|restart|status
```

GRIMSEL should be started prior to FURKA because a restart of FURKA tells GRIMSEL that a user shift has taken place. The correct start order is guaranteed at booting. Note: a restart of GRIMSEL results in a restart of FURKA without start options. `/etc/init.d/grimsel` status queries the status of GRIMSEL, in particular the validity of the license.

4.2.3.3. Configuration

The following parameters can be set globally in the configuration file of GRIMSEL `/etc/grimsel_dectris.conf`:

- logfile: The path and file-name of the GRIMSEL log-file.
- logfile_max_number: This number states how many old log-files are backuped.
- logfile_max_size: Maximum size of log-file. When the maximum size is reached GRIMSEL creates a new log file.
- target: Directory where the data is copied (usually /data).
- ramdisk_mountpoint: Usually set to /ramdisk. This is the mount point of the tmpfs-filesystem to which FURKA writes. GRIMSEL permanently checks this directory for overflow and eventually converts files into symbolic links to the files on target.
- licensekey: The license key is usually the same as for FURKA.
- create_links: If set to YES, source files are eventually turned into symbolic links to target files, if the source mount point is likely to overflow. If set to NO in case of overflow source files that have been copied are deleted
- preserve_uid: If YES, the copied target files get the same uid as the source files, which is basically the ownership set by FURKA. If set to NO, setting the ownership is left to the operating system.
- preserve_gid: If YES, the copied target files get the same gid as the source files, which is basically the ownership set by FURKA. If set to NO, setting the ownership is left to the operating system. The Linux system running on the PPU will set gid either to 0 (root) or to the gid of the parent directory if setgid is set.

For every storage device, `/etc/grimsel_dectris.conf` contains a section beginning with the line

storageID <Storage_Identifier>

where <Storage_Identifier> is an arbitrary string (without spaces) that identifies the storage. In the lines following the keyword storageID the parameters below are set:

- type: Either INTERNAL or EXTERNAL or OPTIONAL. You need exactly one internal (the PPU internal disk) and at most one external device. Several optional storage devices may be added. If external storage exists, the fill state of the internal storage is permanently checked and the internal storage is prevented form overflowing by turning files there into symbolic links to the external storage. After a user shift, the internal storage is cleared as well. Without external storage, it is the beamline administrator's responsibility to keep the internal storage clean.
- target: Target directory where the data is copied.
- n_copy_processes: Number of processes writing data. To optimize write performance a number between 8 and 20 (hardware dependent) is recommended.
- mountpoint: This parameter is ignored if type is not INTERNAL. If an external storage device exists, the fill state of mountpoint is regularly checked for overflow.

4.2.4. Adding Files to the GRIMSEL Pipe

GRIMSEL obtains the information which files to copy from the GRIMSEL file pipe `/var/run/grimsel_dectris_files`. By default GRIMSEL copies only files generated by FURKA. In order to make GRIMSEL copy additional files write their paths to the file pipe:

```
echo /some_dir/./path/to/file >
/var/run/grimsel_dectris_files
```

Note, the '`/./`' within the file path tells GRIMSEL to replicate the directories `path/to/file` but not `/some_dir` in the target directory (see Sections 4.1 and 4.3).

4.2.5. Interacting with GRIMSEL

There exists a python API for basic communication with GRIMSEL. In particular, one can query the status of GRIMSEL with respect to the pending and copied files. Another interface lets you react on warning and error messages.

The GRIMSEL python modules are submodules of dectris.grimsel and may by imported via

```
from dectris.grimsel import <some submodule>
```

The Python packages for CentOS 6 are found here:

`/usr/lib64/python2.6/site-packages/dectris/grimsel/examples/`,

for CentOS 7 here:

`/usr/lib64/python2.7/site-packages/dectris/grimsel/examples/`.

The examples, can be executed by calling python with the –m flag. E.g:

```
python -m dectris.grimsel.examples.GrimselStatus
```

executes the module

```
/usr/lib64/python<Python version>/site-
packages/dectris/grimsel/examples/GrimselStatus.py
```

4.2.5.1. Query the GRIMSEL Status

The module DQueryStatusClient comes with tools to figure out how many files per user shift and storage unit have been copied and are still pending. It defines the class DQueryStatusClient and its member functions:

- class DQueryStatusClient(host=None,port=None)
  Create an object of type DQueryStatusClient.
  - host String representation of the IP address of the GRIMSEL computer
  - port Port number (either numeric or string) defined by status_port in `/etc/grimsel_dectris.conf`
- DQueryStatusClient.host()
  DQueryStatusClient.setHost(host)
  DQueryStatusClient.port()
  DQueryStatusClient.setPort(port)
  Convenience functions to get and set the host and the port of the DQueryStatusClient object
- DQueryStatusClient. queryStatus(nusers=-1)
  Contacts GRIMSEL and returns a list of python dictionaries with the following keywords:
  - usertime, uid, gid: Start time, UID and GID of the user shift
  - storage: storageID
  - pending, copied, failed: Number of pending, copied and failed files of the user

  nusers is the number of user shifts to list. If nusers = -1 all user shifts GRIMSEL remembers (confer the keyword history_max_entries in /etc/grimsel_dectris.conf) are returned.

The example below demonstrates the use of DQueryStatusClient. The python program contacts GRIMSEL every ten seconds and prints the status (source code: `/usr/local/dectris/python/examples/GrimselStatus_0.py`):

```
 1     import sys
 2     import time
 3     from dectris.grimsel import DQueryStatusClient
 4     def prettyTable(stringTable):
 5
 6         nrows = len(stringTable)
 7         ncolumns = len(stringTable[0])
 8         # the maximum width of each column entry is the width of the
column
9          columnWidths = []
10         for i in range(0,ncolumns):
11             column = list(stringTable[j][i] for j in range(0,nrows))
12             columnWidth = max(map(len,column))
13             columnWidths.append(columnWidth)
14
15         tableString = ""
16         # print header
17         for i in range(0,ncolumns):
18             width = columnWidths[i]
19             tableEntry = stringTable[0][i]
20             tableString += tableEntry + ' ' * (width-len(tableEntry))
21             if i < ncolumns-1:
22                 tableString += " | "
23             else:
24                 tableString += "\n"
25         # print horizontal line
26         tableString += '_'*(sum(columnWidths)+3*(ncolumns-1))+'\n'
27         # following lines
28         for row in stringTable[1:]:
39             for i in range(0,ncolumns):
30                 width = columnWidths[i]
31                 tableEntry = row[i]
32                 tableString += tableEntry + ' ' * (width-len(tableEntry))
33                 if i < ncolumns-1:
34                     tableString += " | "
35                 else:
36                     tableString += "\n"
37         return tableString
38
39
40     client = DQueryStatusClient.DQueryStatusClient("127.0.0.1","47199")
41     header = ["User Time", "UID", "GID", "Storage",
42         "Pending", "Copied", "Failed"]
43     while 1:
44         userList = client.queryStatus(5)
45         table = [header] + list( map (str, [ userEntry["usertime"],
46                                      userEntry["uid"],
47                                      userEntry["gid"],
48                                      userEntry["storage"],
49                                      userEntry["pending"],
50                                      userEntry["copied"],
51                                      userEntry["failed"] ] )
52                                      for userEntry in userList)
53         print prettyTable(table)
54         time.sleep(10)
```

In line 41 a DQueryStatusClient object is created. We run the program on the same machine as Grimsel. Hence the IP may be chosen as 127.0.0.1. For the port we choose 47199 i.e. the default port number. In a while loop (line 44) GRIMSEL is regularly contacted. The return value of this request is converted to a two-dimensional array of strings that is passed to prettyTable() (lines 54, 5) to obtain a nicely formatted table. Please have a look at the example file

/usr/lib64/python<Python version>/site-packages/dectris/grimsel/examples/GrimselStatus.py

to obtain a version of this program, which is based on libncurses.

4.2.5.2. Reacting on GRIMSEL Messages

The module DAbstractMessageClient provides an interface to react to GRIMSEL messages. It defines the abstract class DAbstractMessageClient and its member functions:

- class DAbstractMessageClient(host=None,port=None,timeout=0)
  Create an object of type DAbstractMessageClient.
  - host String representation of the IP address of GRIMSEL
  - port Port number (either numeric or string) defined by message_port in `/etc/grimsel_dectris.conf`
  - timeout A broken connection to GRIMSEL will be opened again after timeout seconds or a DClientSocket.DSocketNoConnect exception will be raised, if timeout = 0.
- DAbstractMessageClient.host()
  DAbstractMessageClient.setHost(host)
  DAbstractMessageClient.port()
  DAbstractMessageClient.setPort(port)
  DAbstractMessageClient.timeout()
  DAbstractMessageClient.setTimeout(port)
  Convenience functions to get and set the host, port and timeout of the DAbstractMessageClient object
- DAbstractMessageClient. connect()
  Connects to GRIMSEL and permanently waits for messages. Whenever a message from GRIMSEL arrives, it is passed to processMessage. If the connection to GRIMSEL breaks, an exception of type module DClientSocket.DSocketNoConnect is raised if timeout = 0. If timeout ≠ 0, the connection will be reopened.
- DAbstractMessageClient.processMessage(errcode, shortmessage, longmessage)

  Reimplement processMessage in a subclass of DAbstractMessageClient. The long message is what you may want to pass on to the user. The short message defines the error code more precisely. The possible values for errcode listed in Error! Reference source not found. are defined the module DAbstractMessageClient.py .

Table 1: Troubleshooting

Error Code	Short Message	Description
FILLSTATE_LIMIT	mount point	Mount point is filled up to the limit defined in the configuration file (fillstate_limit in grimsel_dectris.conf). This is not an error; it just means GRIMSEL is starting to convert old files on the mount point into symbolic links.
ALMOST_OVERFLOW	mount point	Mount point is likely to overflow
DISK_OVERFLOW	mount point	Mount point has reached fillstate of 100%
COPY_ERROR	src:dest	Failed to copy src to dest
LINK_ERROR	file name	Failed to create link
DEL_ERROR	file name	Failed to delete file

The code below demonstrates how to run for every error code a specific callback function:

```
 1     import re
 2     import sys
 3
 4     from dectris.grimsel import DAbstractMessageClient

 5     class MessageClient(DAbstractMessageClient.DAbstractMessageClient):
 6         def __init__(self):
 7
super(DAbstractMessageClient.DAbstractMessageClient,self).__init__()
 8             self.setHost("localhost")
 9             self.setPort(47198)
10             self.setTimeout(10)
11
12


13         def fillStateLimit(self, shortMessage):
14             print "Fill state limit on :" + shortMessage
15         def almostOverflow(self, shortMessage):
16             print shortMessage + " is likely to overflow"
17         def diskOverflow(self, shortMessage):
18             print shortMessage + " has reached 100%"
19         def copyError(self, shortMessage):
20             p = re.compile('([^:]+):([^:]+)')
21             m = p.match(shortMessage)
22             src = m.group(1)
23             dest = m.group(2)
24             print "Failed to copy " + src + " to " + dest
25         def linkError(self, shortMessage):
26             print "Failed to create symbolic link " + shortMessage
27         def delError(self, shortMessage):
28             print "Failed to delete " + shortMessage
29


30         def processMessage(self, errcode, shortMessage, longMessage):
31             print "Long message: " + longMessage
32             options = {
33                 DAbstractMessageClient.FILLSTATE_LIMIT :
self.fillStateLimit,
34                 DAbstractMessageClient.ALMOST_OVERFLOW :
self.almostOverflow,
35                 DAbstractMessageClient.DISK_OVERFLOW : self.diskOverflow,
36                 DAbstractMessageClient.COPY_ERROR : self.copyError,
37                 DAbstractMessageClient.LINK_ERROR: self.linkError,
38                 DAbstractMessageClient.DEL_ERROR : self.delError}
39             options[errcode](shortMessage)


40     mc = MessageClient()
41     mc.connect()
```

Class MessageClient (line 5) is derived from DAbstractMessageClient and redefines processMessage() (line 30). The dictionary options (line 32) maps every error code to a specific callback function, to which shortMessage is passed

4.2.6. Possible Issues

- GRIMSEL expects every storage device found in the config file to be mounted. If this is not the case, GRIMSEL will nevertheless write data to the dedicated target directory and at some point fill the root partition.
- The FURKA and GRIMSEL license is tied to one of the network interfaces. To ensure the validity of the license keys make sure, that all network interfaces remain activated.

4.3. Replication of Directory Structures

The following example shows how the directory structure of the data is replicated by FURKA and GRIMSEL.

Let us assume the detector writes data to

`<detector-server>://ramdisk/path/to/data`

If we set source to / and target to `/ramdisk/./imgdir` (see Sections Running FURKA and Configuration) FURKA appends the usr directory (usually set to uid_gid) to target, and data will appear on the PPU in

`/ramdisk/imgdir/uid_gid/path/to/data`

because `/` in source refers to `/ramdisk` on the detector side. The `/./` in the target path is ignored by FURKA, however, GRIMSEL will use it as a starting point, which path information has to be replicated. In our example GRIMSEL will write the data to (target = `/data`)

`/data/imgdir/uid_gid/path/to/data`.

If we had omitted /./ the data would be written to

`/data/uid_gid/path/to/data`.

This is due to the fact that FURKA implicitly adds `/./` to target if omitted.