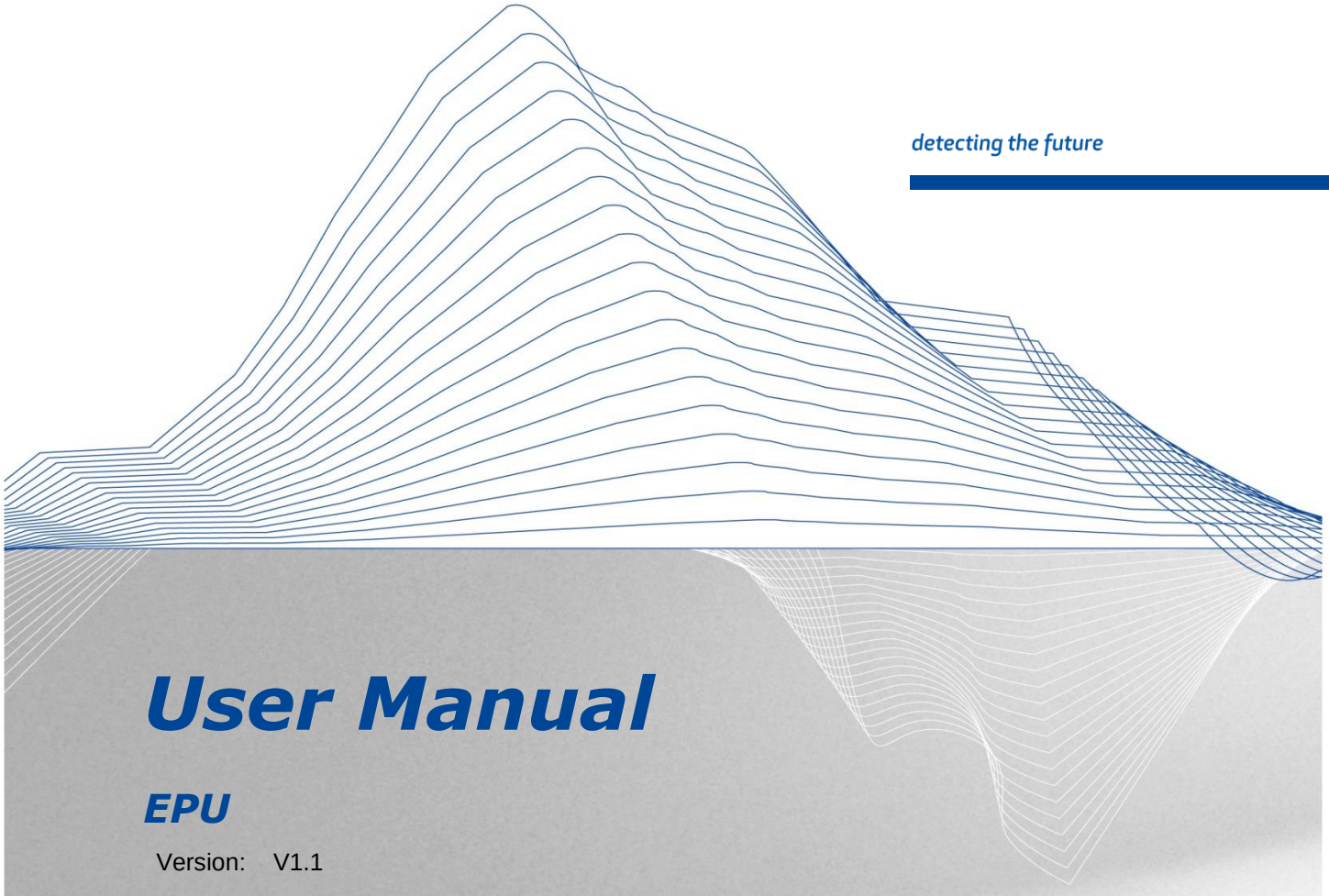
DECTRIS®

detecting the future



User Manual

EPU

Version: V1.1

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

Table of Contents

1. Document History 3
   1.1. Actual document 3
   1.2. Changes 3
2. Address and Support 4
3. EIGER Processing Unit EPU 5
   3.1. Pre-installed Software Packages 5
   3.2. Third-Party Software 5
   3.3. Data Processing with the EPU 5
4. FURKA-EPU & GRIMSEL 7
   4.1. Changes from PPU to EPU 7
   4.2. FURKA-EPU 7
      4.2.1. Running FURKA-EPU 7
      4.2.2. Examples 8
      4.2.3. Configuration 8
   4.3. GRIMSEL 9
      4.3.1. Running GRIMSEL 9
      4.3.2. Start and stop GRIMSEL 9
      4.3.3. Configuration 9
      4.3.4. Adding Files to the GRIMSEL Pipe 10
      4.3.5. Interacting with GRIMSEL 10
      4.3.6. Query the GRIMSEL Status 10
      4.3.7. Reacting on GRIMSEL Messages 12
      4.3.8. Possible Issues 14

1. Document History

1.1. Actual document

Version	Date	Status	prepared	Approved
1.0	11/13/2017	draft	SK	

1.2. Changes

Version	Date	Changes
1.0	12/10/2015	First version of the document
1.1	04/04/2018	Revised and updated

2. Address and Support

DECTRIS Ltd.

Taefernweg 1

5405 Baden-Daettwil

Switzerland

Phone: +41 56 500 21 00

Fax: + 41 56 500 21 01

Email: support@dectris.com

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

This document is for informational purposes only. DECTRIS reserves the right to make changes without further notice to any details herein. The content provided is as is and without express or implied warranties of any kind.

3. EIGER Processing Unit EPU

The EIGER processing unit (EPU) efficiently complements EIGER detector systems at high demand experimental stations. It consists of a high-end server providing substantial processing power and dedicated software packages eliminating bottlenecks in many of today's experiment installations.

EPU key advantages are:

- Highly reliable and performant data transfer (both, EPU mini and XL)
- Fast local storage for several weeks of regular user operation (EPU XL only)
- Fast data processing using XDS third-party-software (EPU XL only)

3.1. Pre-installed Software Packages

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

The default password for the normal user `epu` is set to `EIGER_PU`.

The default password for the super user `root` is set to `!epu<serial number>!`.

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

- FURKA-EPU (copies data from the ramdisk of the detector server to the ramdisk of the EPU). Beware that FURKA-EPU in the EPU is different from FURKA, which is a module used in the PPU for PILATUS detector systems.
- GRIMSEL (copies data from ramdisk to disk)

EPU XL only:

- XDS (crystallographic data reduction package http://xds.mpimf-heidelberg.mpg.de/
- Neggia
- Albula

3.2. Third-Party Software

The third-party software XDS is proposed for crystallographic data processing. The architecture of the EPU allows taking 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 `/var/lib/xds`.

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)

3.3. Data Processing with the EPU

This feature is exclusively available for the EPU XL.

To process with XDS, create XDS.INP with the metadata stored in the master file. If you do not have a mechanism to extract the metadata at your beamline, use one of the tools suggested in the XDS wiki (http://strucbio.biologie.uni-konstanz.de/xdswiki/index.php/Eiger).

For fastest processing, use the Neggia plugin (https://github.com/dectris/neggia) and enable parallel execution of XDS. Adjust XDS.INP as follows:

- Point LIB= to where the Neggia plugin is saved, e.g. LIB= /usr/local/lib64/dectris-neggia.so

- Set MAXIMUM_NUMBER_OF_JOBS= and MAXIMUM_NUMBER_OF_PROCESSORS= to values similar to each other, with MAXIMUM_NUMBER_OF_PROCESSORS= larger and the product of the two numbers slightly smaller than the total number of threads you want to use.

We recommend MAXIMUM_NUMBER_OF_JOBS= 8 and MAXIMUM_NUMBER_OF_PROCESSORS= 18 if only one XDS job is run and data are transferred at the same time. This will use 144 threads for processing, leaving 16 threads for copying and the system.

Space available on the ramdisk of the DCU should be monitored using the command:

```
curl http://<IP of DCU>/filewriter/api/1.5.0/status/buffer_free
```

4. FURKA-EPU & GRIMSEL

FURKA-EPU and GRIMSEL are capable of synchronizing data between several storages. The data flow is illustrated in Fig. 1. FURKA-EPU copies data from the detector server to the EPU ramdisk, GRIMSEL copies the same data from the ramdisk to the EPU storage (Internal Storage) and optionally to an external storage. In the following sections FURKA-EPU and GRIMSEL are described in detail.

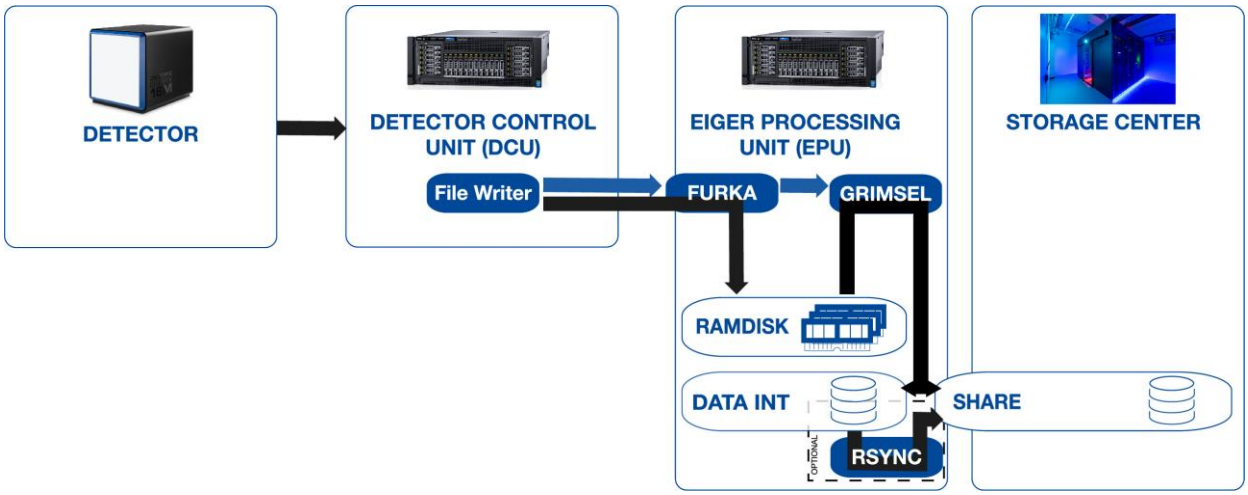


Fig. 1: Data flow of FURKA-EPU (abbreviated here as “FURKA”) and GRIMSEL.

4.1. Changes from PPU to EPU

- There is no default user directory. The user directory must be explicitly defined.
- Multi-detector support cancelled.
- A license key for furka is not needed.
- Furka config file changed (JSON).

4.2. FURKA-EPU

4.2.1. Running FURKA-EPU

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

EPU:

Control FURKA-EPU via the command (with administrator privileges):

```
service furka ( (start|restart) [start options] ) | stop | status)
```

- `start options:`
  - `-u [<uid>]:[<gid>]:[<userdirectory>]` Set the uid and gid of the copied files. userdirectory is appended to the target path of the files. userdirectory must be a relative path. Any of <uid>, <gid> and <userdirectory> may be omitted.
- `status:`
  - Query status of FURKA-EPU, in particular number of processes and validity of the license.

Userdirectory must be a relative path. If used as absolute path the folder structure is created in RAM and on HDD but no files are copied

4.2.2. Examples

Example 1:

Calling

```
sudo service furka restart -u 1000:1000
```

will restart FURKA-EPU in such a way, that all EIGER detector files (excluding empty directories) are copied

to

```
<processing-server>://mnt/data_buffer.
```

Example 2:

Calling

```
sudo service furka restart -u 500::HelloWorld
```

will restart FURKA-EPU in such a way, that all EIGER detector files (excluding empty directories) are copied

to

```
<processing-server>://mnt/data_buffer/HelloWorld/.
```

Note the two ":" in the second example, while there is only one ":" in the first one.

4.2.3. Configuration

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

```
{
   "EIGER_HOST": "10.42.41.10",
   "GID": 0,
   "LOG_FILE": "/var/log/furka_dectris.log",
   "TARGET_DIR": "/mnt/data_buffer",
   "UID": 0,
   "USER_DIR": ""
}
```

UID, GID and USER_DIR are overridden at FURKA-EPU start via the –u option.

4.3. GRIMSEL

4.3.1. Running GRIMSEL

GRIMSEL permanently copies **image data written by FURKA-EPU** from the FURKA-EPU target directory (in the following called /mnt/data_buffer) to the local EPU disk and optionally to external storages. Note, any additional data created on */mnt/data_buffer* is **not** copied! uid, gid and the time stamp of the copied files are preserved. GRIMSEL removes files from /mnt/data_buffer whenever a user change takes place, *i.e.* whenever FURKA-EPU 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 EPU. The idea behind that is, whenever a user change takes place, the data of the former user is no longer needed on /mnt/data_buffer. This particular behavior of GRIMSEL makes it necessary to ***regularly restart FURKA-EPU after each user change to prevent /mnt/data_buffer from overflowing***. When GRIMSEL notices /mnt/data_buffer (or the directory, to which the variable ramdisk_mountpoint in the configuration file is set to) is likely to overflow, it converts files on /mnt/data_buffer into symbolic links to the respective files on the disk. **It is strongly recommended to restart FURKA-EPU after every user change or at least after each beamline shift.** This can be done either explicitly by the command `sudo systemctl furka restart` or by using a different uid.

Exactly one internal storage has to be configured in order to run GRIMSEL. By default the local EPU disk is used as internal storage. In addition at most one external storage and several optional storages may be set up in the configuration file (see section

Configuration below). If an external storage is defined, the internal EPU storage is cleared after a user change and files on the internal storage are in case of overflow substituted for symbolic links to the external storage. Without an external storage it is the beamline administrator's responsibility to keep the EPU storage clean.

All storages must be mounted locally on the EPU. GRIMSEL assumes that the storages are always available. This might be an issue when working with removable USB drives.

If multiple storages are defined, the slower storage system will limit the speed with which GRIMSEL copies data from the RAM disk. Therefore, if the storages have significantly different write speeds it is advisable to define the faster storage as internal storage and sync data from this faster storage to all slower storages.

Rsync provides a simple solution to sync the data from one storage to another.

```
rsync –avz source_storage dest_storage
```

4.3.2. Start and stop GRIMSEL

Control GRIMSEL via the command line (administrator privileges):

```
service grimsel start|stop|restart|status
```

GRIMSEL should be started previous to FURKA-EPU because a restart of FURKA-EPU informs GRIMSEL that a user transition has taken place. The correct start order is guaranteed at booting. Note that a restart of GRIMSEL results in a restart of FURKA-EPU without start options.

`service grimsel status` queries the status of GRIMSEL, in particular the validity of the license.

4.3.3. 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 /mnt/data_buffer. This is the mount point of the tmpfs-filesystem, to which FURKA-EPU writes. GRIMSEL permanently checks this directory for overflow and eventually converts files into symbolic links to the files on `target`.

- `licensekey`: The license key preinstalled.
- `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-EPU. 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-EPU. If set to NO, setting the ownership is left to the operating system. The Linux system running on the EPU will set gid either to 0 (root) or to the gid of the parent directory if setgid is set.

For each storage `/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`. Exactly one internal storage (the EPU internal disk) must be configured and at most one external storage. Several optional storages may be added. If an external storage exists the fill state of the internal storage is permanently monitored and the internal storage is prevented from overflowing by substituting files for symbolic links to the respective file on the external storage. After a user shift the internal storage is cleared. Without an external storage it is within the beamline administrator's responsibility to keep the internal storage clean.
- `target`: Absolute path to the target directory where the data is copied.
- `n_copy_processes`: Default between 8 and 20 (hardware dependent).
- `mountpoint`: This parameter is ignored if `type` is not `INTERNAL`. If an external storage exists, the fill state of `mountpoint` is regularly checked for overflow.

4.3.4. Adding Files to the GRIMSEL Pipe

GRIMSEL obtaines 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-EPU. 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 Section 4.1).

4.3.5. Interacting with GRIMSEL

A python API for basic communication with GRIMSEL is available. 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 python modules are submodules of dectris.grimsel and can be directly imported in the python interpreter. The examples have been tested with Python 2.7.

4.3.6. Query the GRIMSEL Status

The module dectris.grimsel.`DQueryStatusClient` comes with tools to monitor the number of 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 code queries Grimsel every ten seconds and prints the status (source code is also available on the EPU: `/usr/lib64/python2.7/site-packages/dectris/grimsel/examples/GrimselStatus_0.py`)

```
 1    import sys
 2    import time

 3    sys.path.append(sys.path[0] + "/../")
 4    import DQueryStatusClient

 5    def prettyTable(stringTable):
 6
 7        nrows = len(stringTable)
 8        ncolumns = len(stringTable[0])

 9        # the maximum width of each column entry is the width of the column
10        columnWidths = []
11        for i in range(0,ncolumns):
12            column = list(stringTable[j][i] for j in range(0,nrows))
13            columnWidth = max(map(len,column))
14            columnWidths.append(columnWidth)
15
16        tableString = ""

17        # print header
18        for i in range(0,ncolumns):
19            width = columnWidths[i]
20            tableEntry = stringTable[0][i]
21            tableString += tableEntry + ' ' * (width-len(tableEntry))
22            if i < ncolumns-1:
23                tableString += " | "
24            else:
25                tableString += "\n"

26        # print horizontal line
27        tableString += '_'*(sum(columnWidths)+3*(ncolumns-1))+'\n'

28        # following lines
29        for row in stringTable[1:]:
30            for i in range(0,ncolumns):
31                width = columnWidths[i]
32                tableEntry = row[i]
33                tableString += tableEntry + ' ' * (width-len(tableEntry))
34                if i < ncolumns-1:
35                    tableString += " | "
```

```
36                else:
37                    tableString += "\n"

38        return tableString
39
40


41    client = DQueryStatusClient.DQueryStatusClient("127.0.0.1","47199")
42    header = ["User Time", "UID", "GID", "Storage",
43        "Pending", "Copied", "Failed"]
44    while 1:
45        userList = client.queryStatus(5)
46        table = [header] + list( map (str, [ userEntry["usertime"],
47                                    userEntry["uid"],
48                                    userEntry["gid"],
49                                    userEntry["storage"],
50                                    userEntry["pending"],
51                                    userEntry["copied"],
52                                    userEntry["failed"] ] )
53                                    for userEntry in userList)
54        print prettyTable(table)
55        time.sleep(10)
```

In line 41 a DQueryStatusClient object is created. The program is run on the same machine as GRIMSEL. Hence the IP is set 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) in order to obtain a decently formatted table. Please have a look at the example file `GrimselStatus.py` to obtain a version of this program that is based on libncurses.

4.3.7. Reacting on GRIMSEL Messages

The module dectris.grimsel.`DAbstractMessageClient` provides an interface to react on 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 or the connection will be opened again after timeout seconds else.
- `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 Table 1 are defined the module `DAbstractMessageClient.py` .

Table 1

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     sys.path.append(sys.path[0] + "/../")
4     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

4.3.8. Possible Issues

GRIMSEL is expecting 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.