The ATLAS Barrel Alignment Readout System

Reference Guide

2nd Edition

Robert Hart, Karol Poplawski

Nikhef, Amsterdam, Netherlands

Abstract

The barrel alignment system of the ATLAS muon spectrometer consists of up to 6000 optical lines, each one built up of a camera, a light source, a coded mask and a lens. Three layers of multiplexing are applied, controlled by eight embedded Linux PCs, each one equipped with a frame-grabber, which acquires the images to be analyzed. The results are stored into a database for off-line track reconstruction. The multiplexers and frame-grabber are controlled by the Rasdim server and its main client is the WinCC SCADA system. This document describes the setup of the system in USA15 and how it is used by WinCC. It also contains instructions how to setup, maintain and debug the system.

Contents

Introduction to the 2nd edition.

Mid December 2016 the entire controls infrastructure of the barrel alignment in USA15 was rejuvenated by the introduction of the so-called *Ethernet based frame-grabbers*. The 8 PCs, running Windows XP and containing the DT3162 frame-grabber, were replaced by 8 embedded Linux machines equipped with a Xilinx Zynq Z7015 FPGA, emulating an ADC which behaves as the new frame-grabber. The entire description (history, sources, manuals, documents, etc.) can be found on the Nikhef Redmine page:

https://www.nikhef.nl/nikhef/departments/ct/po/Atlas/Rasnik/Rasdim/html/index.html

NB: This document does not replace the first edition but emphasizes more on the differences and its new equipment. The general concept described in the 1st edition is still valid.

1 General Description

The optical lines (also known as *channels*) of the barrel alignment system are mounted in the ATLAS cavern, mainly attached to the MDT chambers. A three-layer multiplex-scheme (TopMux → MasterMux → RasMux) connects the channels to the 8 TopMuxes located in USA15. Eight embedded Linux PCs, equipped with an FPGA emulating a frame-grabber, and for the remainder of this document called **EPC**, are utilized to control the readout, which in general means:

1. Select channel
2. Grab image
3. Analyze image
4. Store results into database

2 Rejuvenation of the readout controls

The main differences of the alignment controls upgrade of December 2016 are:

1. **Rasdim PCs**: The 8 Windows-XP machines were replaced by 8 EPCs containing a Xilinx Zynq Z7015: dual core ARM Cortex A9, including an FPGA emulating the functionality of the grabber. The EPCs are named *epc-mdt-algn-0x* ($1 \le x \le 8$) and control TopMux[*x*] respectively. Although the boards work properly, they do have a couple of drawbacks. See Appendix 7.1 for details.
2. **WinCC**: the clients of the Rasdim servers are now running on 2 Linux PCs (*pcatlmdtbal11* and *pcatlmdtbal12*) running CentOS[1] provided and maintained by the ATLAS *sysadmin* group.
3. **Supervisor**: remains *pcatlmdtbal9* but moved to a Linux PC like *pcatlmdtbal11* and *pcatlmdtbal12*. Its functionality still holds the *dns*-server for the DIM communication, determination of the state/status of the barrel alignment and running an *ntpd*-daemon for a proper date/time.
4. **Analysis**: The *Foam* analysis has been replaced by *Soap*. Both *Soap* and *Spot* are modified as servers and run on both *pcatlmdtbal11* and *pcatlmdtbal12*.

[1] CentOS: de facto standard CERN Linux OS

2.1 Network connectivity

The EPCs run a Linux version which is not supported nor maintained by CERN. For security reasons it was decided to connect them behind a separate switch. The supporting CentOS machines: *pcatlmdtbal[9,11,12]* are not only connected to the ATN (ATLAS technical network), but also connected to the switch and thus provide a gateway to the EPCs (see Figure 1).

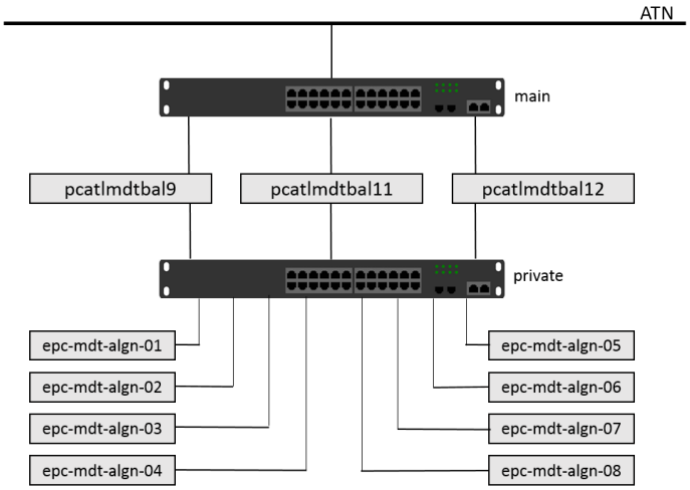


Figure 1: Barrel alignment network connectivity

2.2 Rack Y.28-19A1

Like the TopMux the EPCs come in pairs of two. A rack mountable device called RASVIG contains 2 EPCs (see picture 1).

Picture 1: RASVIG module

Figure 2: Rack Y.28-19A1

Figure 2 shows rack Y.28-19A1 (front-view) with the equipment for the barrel alignment. It is incomplete and not to scale. It shows the switches, RASVIGs and TopMuxes in their relative positions. The gaps below the TopMuxes are used for the long cables to the cavern UX1.

Somewhere between TopMux5/6 and RASVIG2 there is a 1U high lockable drawer (not shown) with some handy tools and documentation. There is no KVM switch nor spare RASVIG installed in the rack. The power supplies of the TopMuxes, 24 V dc, are located between pcatlmdtbal12 and RASVIG 4.

RASVIG1 contains *epc-mdt-algn-01* (left) and *epc-mdt-algn-02* (right) to control respectively TopMux1 and TopMux2, RASVIG2 contains *epc-mdt-algn-03* and *epc-mdt-algn-04* to control TopMux3 and TopMux4, etc.

NB: Rack Y.28-19A1 is connected to UPS.

2.3 WinCC

Each Rasdim server, running on the embedded Linux PCs (i.e. *epc-mdt-algn-0x*), has its own dedicated client, namely a WinCC project called ATLMDTBAL*x*. In total 8 WinCC projects, which are equally distributed to run on *pcatlmdtbal11* (ATLMDTBAL[1-4]) and *pcatlmdtbal12* (ATLMDTBAL[5-8]). On *pcatlmdtbal9* the WinCC project ATLMDTBAL9 runs to supervise these projects and to determine a proper state/status for the ATLAS-MDT FSM tree.

2.4 Analysis

Two types of coded masks are applied: the *Soap*[2] type and the *Spot* type, hence for each one a distinct analysis server is available. The *Soap* type handles images with a chessboard pattern with encoded edges, the *Spot* type images have a mask with four holes. Around 90% of the ATLAS channels are of type *Soap*, the remaining 10% of type *Spot*. Both servers communicate with the *libcurl* package.

1. **SOAP**: Written in Java and at the moment the de-facto analysis module for Rasnik masks.
2. **SPOT**: Written in C++.

To balance the load a *Soap* and *Spot* server are running both on *pcatlmdtbal11* and *pcatlmdtbal12*. Again pcatlmdtbal11 is meant for the group epc-mdt-algn-0[1-4], ATLMDTBAL[1-4] and *pcatlmdtbal12* is meant for group *epc-mdt-algn-0*[5-8], ATLMDTBAL[5-8].

2.5 Rasdim

The Rasdim server is the key-element (the spider in the web) of the alignment system. It runs on each EPC. Figure 2.1 shows the central position of the server, when analyzing a channel as requested by the WinCC client.

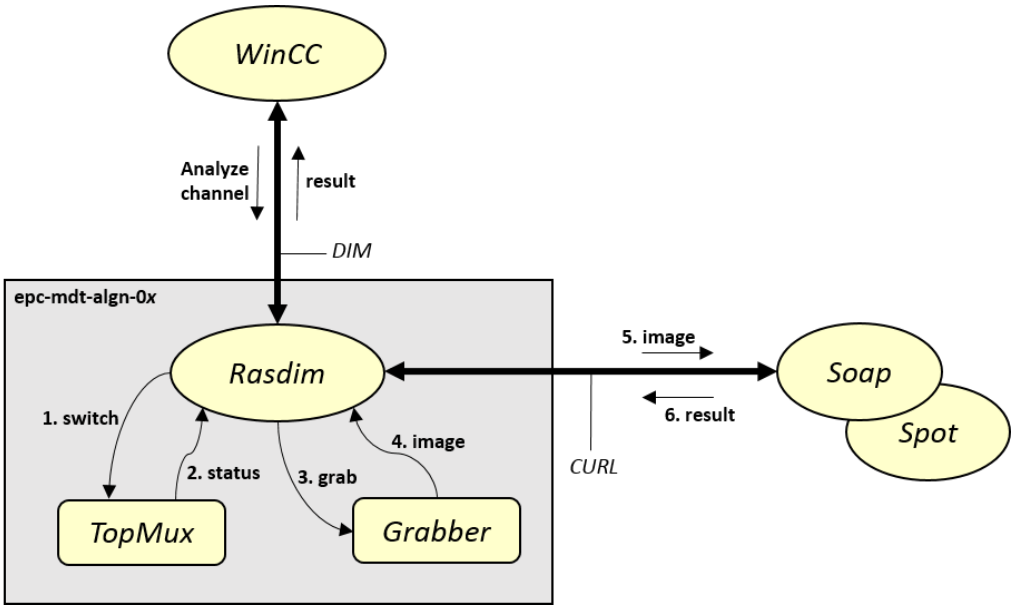


Figure 2.1: Rasdim control flow

The communication protocol between Rasdim and its clients is DIM[3] and between the server and the analysis servers is cURL[4]. Rasdim is written in C++ and cross-compiled for an ARM processor. See Appendix 7.5 for more details.

[2] Formerly known as Rasnik and Foam.
[3] https://dim.web.cern.ch/dim_info.html
[4] It uses the http protocol built upon the *libcurl* library.

3 Technical Aspects

This chapter describes in more detail the various aspects of the barrel alignment equipment. Some parts can be considered as a user manual as well.

3.1 RASVIG

Each RASVIG contains 2 embedded Linux PCs. Each of them has 2 cables connected (located at the front) to the TopMux.

1. **RS232** (COM)-cable: to be used for selecting and setting a particular channel (i.e. camera, light source and possibly camera settings).
2. **DVI-I**[5] video cable: to obtain the captured image.

At the back of the RASVIG you may find for each EPC an ethernet connection and 2 USB ports. The USB ports are labeled 1 and 2 and have the following purpose:

1. **USB1**: *OTG* connector, see chapter 2.3 of the FLORIDA Topic RASNIK Product Guide for more details.
2. **USB2**: */dev/console*, output of the kernel.

3.2 TopMux

The RS232 or COM-cable is connected to the connector labeled RS232. The DVI-I cable is connected to the connector labeled DT3162. The RasMux and MasterMux are passive entities, but the TopMux contains a simple processor accepting commands via the COM-cable. The protocol is completely synchronous and provides commands like switch on camera and light source for an arbitrary channel. On success the belonging video signal is passed onto the DT3162 (DVI-I) output.

WARNING: The 8 major output connectors are not identical! Outlet 1 until 6, **demand** a 3-layer mux scheme including the MasterMux in between. This is the case for ATLAS, as cabled in USA15. For small readout setups, like the ones at Nikhef, MPI Munich and Ann Arbor Michigan, no intermediate MasterMux is used. For this purpose outlet 7 and 8 must be used, but **only** if the dip-switch at the front is set properly. If not, it may damage the TopMux!

3.3 EPC µSD-card

Each EPC is equipped with a µSD-Card of 64 GB. The card itself is easily replaced. Before the µSD-card can be used, the right image has to be installed, currently version 4.6.0 built on June 16 2017. The command `uname -a` on any EPC should yield the following output:

```
Linux epc-mdt-algn-01 4.6.0 #1 SMP PREEMPT Fri Jun 16 10:15:49 CEST 2017 armv7l GNU/Linux
```

Executing the following script on a Linux machine and using a card-reader will install the proper version on the µSD-card, but works only if it is an already used µSD-card with an earlier or equally and damaged version. The script expects to see `/dev/sdb1` and `/dev/sdb2` on the card. It is mainly used for upgrading to a new version. New µSD-cards should be treated differently. A copy should be made (using `dd`[6]) of an existing µSD-card (with the version mentioned above) and written back (using `dd` again) onto a new card. See chapter 3.3.2 for more details.

[5] The –I means it is completely (all pins) connected.
[6] Dump data command of Linux.

```
#!/bin/sh
# first unmount the automatically mounted mounts done by connection
umount /dev/sdb1
umount /dev/sdb2

mount /dev/sdb1 /mnt/boot
mount /dev/sdb2 /mnt/root/

cp -f /project/ct/po/miami/linux-4.6/boot/* /mnt/boot

cd /mnt/root/
tar xvf /project/ct/po/miami/linux-4.6/nikhef-image-topic-miami-rasnik-xc7z015.rootfs.tar.gz
sync    # this takes a while
cd
umount /mnt/boot
umount /mnt/root
```

The filesystems (devices) of the current Linux system is shown below (output of the *df* command):

```
[epc-mdt-algn-01][rhart][~] df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/root               398316    135416    254708  35% /
devtmpfs                494716         0    494716   0% /dev
tmpfs                   515708        36    515672   0% /run
tmpfs                   515708        84    515624   0% /var/volatile
tmpfs                       64         0        64   0% /media
/dev/mmcblk0p1           63346      3770     59576   6% /media/mmcblk0p1
/dev/mmcblk0p2          398316    135416    254708  35% /media/mmcblk0p2
/dev/mmcblk0p3        60406528  44634624  14723328  75% /media/mmcblk0p3
[epc-mdt-algn-01][rhart][~]
```

For the alignment system only 2 filesystems are relevant:

1. **`/dev/root`**: container of the Linux system itself, including the binaries, libraries, configuration files, etc. Contains also `/home` where the directories of the users reside.
2. **`/dev/mmcblk0p3:`** biggest file system (initially empty). Used as location for the alignment binaries and temporary data storage. The data storage contains log files, result files and images (saved as `.tif`).

3.3.1 Modifying the µSD-card

In order to use the *bare* system the following instructions has to be carried out. The machine ***meun*** at Nikhef is used as example. Within the Nikhef network, the machine ***meun*** with its predefined IP number and MAC address is already registered. Another machine within the Nikhef network is called ***krit***. The common folder */project/ct/po/miami* within the Nikhef environment is used to store the major binaries, images and configuration files of the EPCs.

1. Logon using the USB OTG connector as described in chapter 2.3 of the FLORIDA Topic RASNIK Product Guide.
2. Change or add the *root* password.
3. Add the usual users: *karolp*, *rhart*, *saclay*, *mpi*, *umich*, *guest*.
   NB: The passwords for root and the usual users are kept secret.
4. Edit */etc/hostname* to the DNS[7] hostname of the EPC: → ***meun***

[7] Domain Name System, **not** the DIM dns server!

5. Modify */etc/network/interfaces*:
   cd /etc/network
   vi interfaces
   ```
   #iface eth0 inet dhcp
   iface eth0 inet static
   address 192.16.192.135
   gateway 192.16.192.80
   netmask 255.255.255.0
   hwaddress ether 76:77:98:9F:3B:52
   ```
   [8]
6. Connect the EPC to the Nikhef network and restart the board:
   shutdown –r now

The EPC, now known as ***meun***, is restarted and could be pinged from within the Nikhef network. In order to modify the EPC as alignment machine the following files and binaries have to be copied and installed on it. The public key *id_rsa_rasnik* could be found there as well. They are located inside the directory: */project/ct/po/miami/puppis*[9]. For reasons of convenience the entire contents of the ***puppis*** directory is also stored in a so-called *tar* file, named: */project/ct/po/miami/puppis.tar.gz*

7. Logon login.nikhef.nl and make sure the public key *id_rsa_rasnik* resides in *.ssh*.
8. Copy the *puppis* tar file to ***meun***:
   scp /project/ct/po/miami/puppis.tar.gz user@meun:~
   NB: user is of course one of the usual users.
9. Logon to meun:
   ssh user@meun
10. Extract the tar file:
    tar –xvf puppis.tar.gz
11. Copy and execute the standard *.bashrc* file for EPCs:
    cp puppis/.bashrc .
    . .bashrc
12. Become super-user and execute the *.bashrc* file :
    su
    . .bashrc
13. Fix the missing link:
    ln –s –f /lib/ld-2.22.so /lib/ld-linux.so.3
    Without this link it is not possible to execute a *.elf* binary!
14. Create a symbolic link called ***balign*** to the biggest filesystem:
    ln –s –f /media/mmcblk0p3 /balign
15. Copy the puppis/balign folder to /balign:
    cp –r puppis/balign/ /balign*
    Two folders are copied: **bin**, containing the binaries with their configuration files and **data**, meant for temporary storage like results, logging and images.
16. Copy the pre-defined *interfaces* files and create the appropriate link.
    cp puppis/etc/network/interfaces /etc/network*
    ln –s –f /etc/network/interfaces.meun /etc/network/interfaces

[8] The EPC boards do not have a fixed MAC-address. Its MAC address is set by the *hwaddress* entry inside the /etc/network/interfaces file.
[9] Puppis is Latin for mirror.

The */etc/network* directory contains now the *interfaces* files for all known Rasdim setups. It's a matter of the proper link which defines the identity of the EPC.

17. Set the correct time zone:
    For CERN, MPI and Nikhef: *ln –s –f /usr/share/zoneinfo/Europe/Paris /etc/localtime*
    For UMICH: *ln –s –f /usr/share/zoneinfo/America/Chicago /etc/localtime*
18. Copy the *ntpd* and alignment daemon startup scripts:
    cp puppis/etc/init.d/ /etc/init.d*
19. Select the appropriate IP number of the *ntpd* daemon depending on your location:
    vi /etc/init.d/ntpd.sh
    Make sure only one will be started. The one belonging to your location should be selected, the others are made inactive by a # in front of it.
20. Create the following symbolic links in order to start them at startup (run level 5):
    cd /etc/rc5.d
    ln –s –f ../init.d/ntpd.sh S99ntpd.sh
    ln –s –f ../init.d/rasdimd.sh S99rasdimd.sh
    ln –s –f ../init.d/rasdim_watchdog.sh S99rasdim_watchdog.sh
    ln –s –f ../init.d/cleanerd.sh S99cleanerd.sh
    ln –s –f ../init.d/wdogd.sh S99wdogd.sh
21. Create the appropriate link for the *rasdim.ini* file:
    ln –s –f /balign/bin/ini/rasdim.ini.meun /balign/bin/rasdim.ini
    Chapter 4.1 describes the *rasdim.ini* file in more detail.
22. Restart the board and it should work.
    shutdown –r now

3.3.2 Install an unused µSD-card.

Before an unused or new µSD card can be installed and overwritten, an image of a µSD card of a working EPC has to be copied first. Make sure that the EPC is running well, has the right version and does not have a corrupt file system. The procedure for **Linux** machines is as follows:

1. Shutdown the EPC and switch its power off.
2. Take the µSD card from the EPC and put the card into a USB card-reader.
3. *$sudo umount /dev/sdb*[10]
4. *$sudo dd if=/dev/sdb of=/tmp/image bs=1m*
5. Wait for approximately 2-3 minutes per GB. When finished the following message will appear:
   59640+0 records in
   59640+0 records out
   62537072640 bytes transferred in 16103.223197 secs (3883513 bytes/sec)
6. Eject the µSD card.
7. Save the image for later usage. It may be useful to put the name of the copied EPC to the image name. In this example the Nikhef EPC named *krit*.
   $cp /tmp/image /project/ct/po/miami/uSDimage_krit

[10] The sdb name can vary depending on the disk insertion order, sdb for 2nd disk, sdc for 3nd disk, etc. Check it with *$ls /dev*.

In order to install an existing image onto a new µSD card, use the following procedure:

1. Insert a new µSD card with at least the same capacity into a USB card-reader.
2. *$sudo umount /dev/sdb*
3. *$sudo dd if=/project/ct/po/miami/uSDimage_krit of=/dev/sdb bs=1m*
4. Wait for approximately 1-2 minutes per GB. When finished the following message will appear:
   59640+0 records in
   59640+0 records out
   62537072640 bytes transferred in 6317.329912 secs (9899289 bytes/sec)
5. Eject the µSD card.

To modify the newly uploaded image on µSD card with its final details follow the instructions below:

1. Insert the new µSD card into the desired EPC.
2. Power on the EPC – first the main switch and after 20 seconds the EPC button (it will illuminate green).
3. Logon using the USB OTG connector as described in chapter 2.3 of the FLORIDA Topic RASNIK Product Guide. If at Nikhef and the image is that of krit, connect the krit Ethernet cable to it and login using ssh and become super-user.
4. Change the parameters and links according to points 4, 16 (for UMICH as well 17), 19 & 21 from chapter 3.3.1.
5. Then restart the EPC by executing a command *$ shutdown -r now* as super-user.

4 Rasdim

The Rasdim server, like the other alignment binaries, is located inside the folder */balign/bin*. As member of run-level 5 it is started as part of the boot procedure. Its name is *rasdimd*, which is in fact a link to a binary of the latest version (currently **Rasdim-822.elf**[11]). An important feature of the server is its initialization file called: *rasdim.ini*. It contains the set of parameters necessary for the behavior of the server. Table 4.1 shows the list of parameters and the values for *epc-mdt-algn-04*, *meun* and the default.

parameter	***epc-mdt-algn-04***	***meun***	***default***
Version	8.2.2	8.2.2	8.2.2
Server	4	1	99
DnsHost	10.145.70.17	145.116.54.78	galego.nikhef.nl
GrabDelay	800	800	800
I2cGrabDelay	30	30	30
SkipFrames	8	8	8
ComPort	ttyPS1	ttyPS1	ttyPS1
BaudRate	4098	4098	4098
SoapDir	/balign/bin/soap	/balign/bin/soap	./soap
SoapExec	soap-service-1.5	soap-service-1.5	soap-service-1.5
SoapHost	10.145.70.18	localhost	localhost
SoapPort	8081	8081	8081
SpotHost	10.145.70.18	localhost	localhost
SpotPort	8095	8095	8095
LogDir	/balign/data/log	/balign/data/log	.
ImageDir	/balign/data/images	/balign/data/images	./data/images
ResultDir	/balign/data/results	/balign/data/results	./data/results
BlackEdgeX	5	5	5
BlackEdgeY	3	3	3
RasnikEnhance	0	0	0
SpotEnhance	0	0	0
TransWidth	5	5	5
Debug	0	0	0

Table 4.1: rasdim.ini configuration parameters

The first action the server does, is to read the *rasdim.ini* file and take over its parameters. Then it writes the configuration back into a file named *rasdim_dump.ini*. In general the *rasdim.ini* and *rasdim_dump.ini* are the same. In case the *rasdim.ini* file is not present, the parameters are set to their default values, hence the *rasdim_dump.ini* contains only default values. The default value is also set for each parameter not listed in the *rasdim.ini* file. The *rasdim_dump.ini* file always contains the parameters the Rasdim server runs with.

[11] The number 822 denotes the version number, hence version 8.2.2.

4.1 Rasdim configuration parameters

A description of the parameters of table 4.1 is listed below. Some of them belong to each other and are described commonly.

- **Version**: currently 8.2.2
- **Server**: this number is extremely important for the ATLAS setup. In order to distinguish them, each Rasdim server has its own number. On epc-mdt-algn-0*x*, the server has the name Rasdim*x* and the belonging WinCC project is ATLMDTBAL*x* ($1 \leq x \leq 8$).
  Outside ATLAS (the production and test setups) the server number is always 1.
- **DnsHost**: the IP number of the machine the *dns*-server is running. For the 8 ATLAS EPCs it is `10.145.70.17`, also known as *pcatlmdtbal9*.
- **GrabDelay**: the delay time in milliseconds before an image is grabbed. Without an I^2C setting, it takes some time until a camera is ready due to its auto exposure.
- **I2CGrabDelay**: the delay time in milliseconds before an image is grabbed when using any I^2C setting. Using such a setting sets the camera directly into the desired mode. Hence the short delay time.
  NB: Almost all ATLAS channels have an I^2C setting.
- **SkipFrames:** the grabber has an internal (FIFO) buffer in which it keeps the last grabbed frames. To get the images of the selected channel a number of frames (8 seems to be fine) have to be skipped first in order to avoid an image of the previous channel.
- **ComPort:** the name of the RS232 serial port device, always *ttyPS1* (to be found in */dev*).
- **BaudRate:** transmission speed of the RS232 port. Always 4098. It's a #define from <termios.h> and belongs to the value of 115200 bits/s.

All EPCs support Java and are capable of running the *Soap* server. Using the local *Soap* server is very slow and unacceptable for ATLAS. Hence, it uses a *Soap* server on a powerful CentOS machine. For *epc-mdt-algn*-[01-04] the machine *pcatlmdtbal11* (`10.145.70.17`) is used and for *epc-mdt-algn*-[05-08] it uses *pcatlmdtbal12* (`10.145.70.18`). The same is valid for the *Spot* server.

In case the remote *Soap* server is unreachable, the EPC will fall back onto the local *Soap* server. The non-ATLAS EPCs (production and test setups) will always use the local *Soap* server. The *Spot* server is not available on the EPCs, hence it is used only remotely. The non-ATLAS EPCs do not use the *Spot* server, because they do not use *Spot* channels.

- **SoapDir:** the *Soap* analysis, based on Java, is located here.
- **SoapExec:** the name of the *Soap* executable (a jar file) inside the **SoapDir**.
- **SoapHost:** the IP number of the machine the *Soap* server is running on. Localhost means the EPC itself.
- **SoapPort:** the port number of the *Soap* server.
- **SpotHost:** the IP number of the machine the *Spot* server is running on. Localhost is meaningless, because the EPC does not support it.
- **SpotPort:** the port number of the *Spot* server

For the logging, results and images separate directories are used:

- **LogDir**: location of the log files generated by the Rasdim server. For each day a separated log file is generated with the following syntax: rdlog_*yyyymmdd*.txt
- **ImageDir**: location of the directories (for each day 1) in which the images for that day are stored. Syntax of the directory is: img*yyyymmdd*. The syntax of an image inside the directories is: *channel*.*yyyymmdd*.*hhmmss*.tif. Not only the date but also the time of the image.

- **ResultDir**: location of the result files. Contains for each grab and analysis its result. For every day a result file is generated with the following syntax: rd_*yyyymmdd*.txt

The Rasnik cameras use a VV5430 CMOS sensor with a width of 384 pixels and a height of 287 pixels. The analysis servers expect images with a width of 392 pixels and a height of 292 pixels. The grabbed image is too small and to solve this (and maintaining the aspect ratio) the Rasdim server adds a black edge around the image, which is determined by the following 2 parameters:

- **BlackEdgeX**: the number of black pixels at the left side. The number of black pixels on the right side is determined by this number, namely: 392 - 384 - BlackEdgeX.
- **BlackEdgeY**: the number of black pixels from the top. The number of black pixels at the bottom is determined by this number, namely: 292 - 287 - BlackEdgeY.

The Rasdim server has the ability to enhance the image to get a better contrast and brightness. Most images use only a small range of the 8-bit black and white values. If enhanced the entire image is scanned for its brightest value and darkest value. These values are used as maximum 0x0 and minimum 0xFF and the entire image is now transformed by a linear fit using these limits. It turned out that almost all enhanced images could not be analyzed.

- **RasnikEnhance**: If not zero, enhance all images of type *Soap*.
- **SpotEnhance**: If not zero, enhance all images of type *Spot*.

One of the most notorious errors is the GRABBER error (value 3), also described as: "*No grab within timeout*". The Rasdim server was unable to grab the image. A possible cause may be the quality of the signal, which is quantified by the number of samples/pixels before a signal moves from **sync** level to **blanking** level (and vice versa). This number is known as the **max_transition_width**. If exceeding this number, the GRABBER error is raised. The grabber of the EPC has a couple of dynamic levelling parameters (see also chapter 3 of the FLORIDA Topic RASNIK Product Guide and Appendix 7.2). The **max_transition_width** parameter has a maximum value of 8, which is also the value at startup of the EPC.

- **TransWidth**: the default value of the **max_transition_width** variable.
- **Debug:** If not zero the Rasdim server yields more debug information which can be found in the log file.

4.2 Rasdim startup

At startup the Rasdim server performs the following actions:

1. Assign the configuration parameters by reading the *rasdim.ini* file as described in chapter 4.1.
2. Initialize the frame-grabber and create a buffer for the images. Initially the buffer is filled with an artificial image called the *epoch*, showing a picture of all black and white values from top to bottom. On error exit.
3. Open the connection to the TopMux, reset it and reads its version number. On error exit (happens mainly if the RS232 cable is not connected).
4. Set the **max_transition_width** parameter to the default as read from the *rasdim.ini* file.
5. The Rasdim server runs with a couple of threads. A special thread is started called *_ABS* (from Anti-lock Braking System) which runs during the lifetime of the server. It detects possible deadlocks and when it does, the Rasdim server commits suicide. It may seem redundant, because a deadlock never occurred so far.

6. Setup the DIM connection and server.
    a. Connect to the DNS server (parameter **DnsHost**).
    b. Set up the *Image* service: each image grabbed is exposed.
    c. Set up the *Stdout* service: each analysis result is exposed as an ASCII line including the name of the channel, date/time and the results.
    d. Set up the RPC service. The most important one. It is capable of receiving commands and sending back the results. Runs in a separate thread.
    e. Start the server with the name: Rasdim*x* (*x* is the **Server** parameter).
7. Check the connectivity with the *Soap* server. If it is localhost or the remote one is not responding, a local Soap server will be started using the *system(3)* call. The following command, on for instance meun and based on table 4.1, is executed:
   $java –jar –Dport=8081 –DfixedFlip=true /balign/bin/soap/soap-service-1.5.jar
   NB: When **SoapHost** is not localhost and the server is unreachable, the Rasdim server shall with each analysis request try to use the remote **SoapHost** first, before it uses the local server. It continues doing so until the remote host becomes available.
8. Check the connectivity with the Spot server. If not available, just continue.
9. Expose the *epoch* image (already available in the image buffer) using the DIM *Image* service. See picture 4.1. Notice the black edge!
10. Enter the main loop and wait for commands.

Picture 4.1: The epoch image

4.3 Max Transition Width

The *TransWidth* parameter is an important value, especially for ATLAS, because it is a kind of sensitivity argument. The default value is 5 and for around 95 % of the ATLAS channels this value is adequate. However, some channels fail with this setting, not always but sometimes more than 50% of obtaining an image. Setting the *TransWidth* to 6 for these channels improves the gathering of the images dramatically. So, why don't we use a higher (and more sensitive) value for all channels? The answer is

no, because the majority of the channels which are doing well with *TransWidth* 5, may skip lines gathering the images with *TransWidth* 6 or higher. Skipping a line corresponds to several microns in the output of the analysis. This jitter is unacceptable. The conclusion is that almost all channels use a *TransWidth* value of 5, and a well-defined set of them is using value 6. Values other than 5 or 6 are not used nor recommended.

Each channel has its own value of the *TransWidth* parameter. In order not to change the DIM RPC interface between the Rasdim server and its clients (i.e. WinCC), one of the arguments, as part of the grab and analyze request, which became obsolete, is now in use for this purpose. For WinCC it is the `.nimgs` data-point element (of type RDChannel). To change it, go to the *Edit Channel* panel, select the desired channel and open its *Main* panel (see figure 4.1). Within the *Misc* frame you can find (and change) this parameter, but be careful!

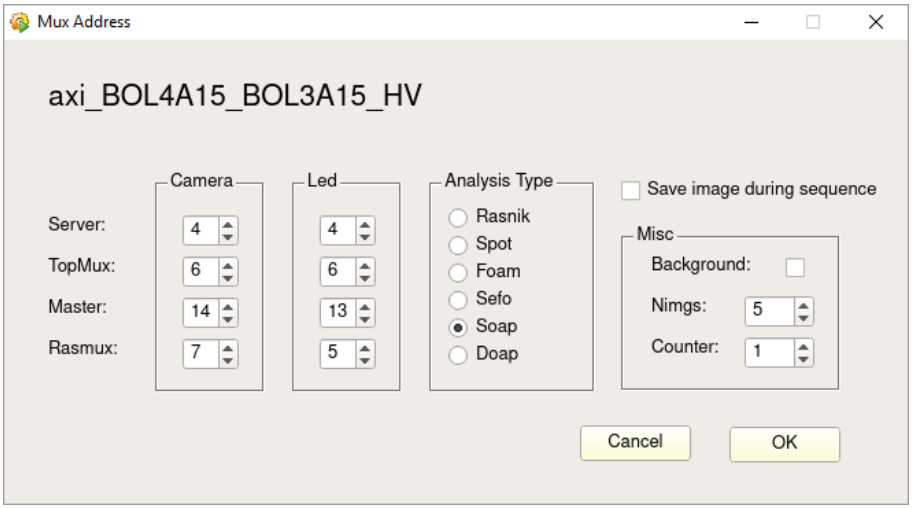


Figure 4.1: Channel main panel

On *pcatlmdtbal9*, WinCC project ATLMDTBAL9, a panel is available which shows a quick overview of the *TransWidth* settings. It is called *RDsvCountDpe.pnl* and an example is shown in figure 4.2. It shows the output of the '`List Channels == 6`' button. Currently there are 53 channels with a *TransWidth* value 6. The '`List Channels: img != 5 & img != 6`' button yields 0 channels. The button '`List Channels: img == 5`' shows the list of the remaining channels (more than 5800). There is also a button named '`List Channels: cnt!= 1`". The data-point element `.cnt` indicates the number of consecutive grab and analyze actions of that particular channel. Like the Nimgs, it can be found in the *Misc* frame of the *Main* panel (see figure 4.1). Normally this number is set to 1 for all channels. However, there are a couple of channels which always fail (and even set the EPC in a deadlock). These channels are taken out of the sequence by setting the `.cnt` data-point element to 0.

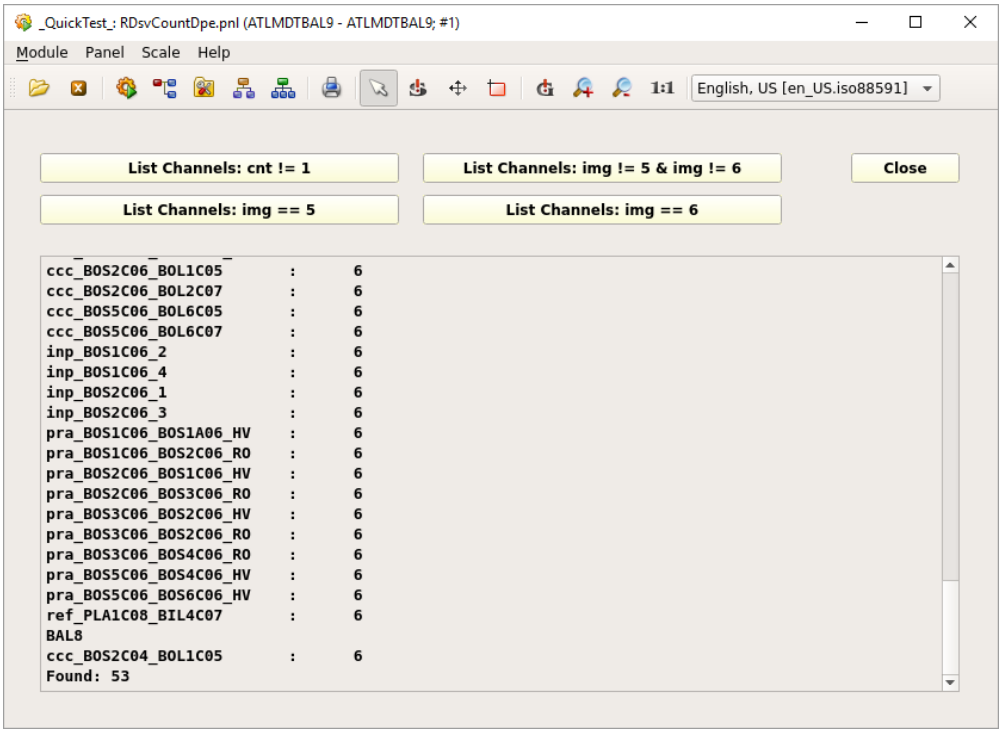


Figure 4.2: Count and nImg overview

4.4 Permanent timeout problem

The GRABBER or timeout errors are compared to other error like MUX (camera or light-source not connected) or ANALYSIS (the image could not be analyzed) errors, rare. But sometimes the frame grabber enters a mode in which all subsequent image grabbing's fail due to a timeout. The only way to solve this is to restart the EPC! The Rasdim server detects when 10 subsequent timeouts occur. Then it performs a *shutdown –r now* and within a minute everything is up and running again. Just before it restarts, the first channel of the set of 10 channels which went into timeout, is added to the log file: *LogDir/grabErrors.txt*. This permanent timeout behavior is particularly located on *epc-mdt-algn-07*, on which it happens once to twice a day. A solid explanation is not found yet, although we have to keep in mind that this EPC handles the most remote channels on the C-side of the barrel. It might also be an electrical problem, due to a ground loop. The number of subsequent timeouts (10) is hard coded inside the Rasdim server and not part of the configuration as found in the *rasdim.ini* file.

5 Supporting programs

Besides the Rasdim (*rasdimd*) and Soap server, there are a couple of other executables (binaries or scripts) to be found in `/balign/bin`. All of them can only be executed as super-user. The executables with the suffix *.elf* (Executable and Linkable Format) are binaries and are developed and cross-compiled at Nikhef (see Appendix 7.5). The *.ini* files are configuration files and the ones without suffix are shell-scripts. The following list explains them (in random order):

1. **rasdim_watchdog**: a shell-script which ensures the Rasdim server is running. If for any reason the Rasdim server is not running, within 60 seconds it will be restarted. The script, as member of run-level 5, is started during boot up. Appendix 7.3 shows the script.
2. **adc_offset**: a shell-script obtaining the main parameters from the frame grabber IP. Appendix 7.4 shows the script.
3. **mmm**: low-level debug program to check the connectivity with the TopMux via RS232. Before using it, make sure that the Rasdim server and the rasdim_watchdog are not running, because only one application may open the RS232 port. It shows exactly which characters are send and received. Typical commands are:
   V: get firmware version of the TopMux
   O: reset the TopMux
   Cxyz: switch camera on outlet TopMux[*x*], MasterMux[*y*] and Rasmux[*z*]
   Lxyz: switch light-source on outlet TopMux[*x*], MasterMux[*y*] and Rasmux[*z*]
4. **wdogd**: the watchdog daemon and like the *rasdimd* daemon it is a symbolic link to the current version. It has the following task: Whenever the watchdog is not refreshed within a certain interval, the board is restarted automatically. The interval time and its log folder are found in its *.ini* file: `wdog.ini`. The default value for the interval time is 60 seconds and the default log area is `/balign/data/log/wdlog`. The actual used parameters are found in `wdog_dump.ini`.
5. **cleanerd**: daemon which prevents the occupancy of `/dev/mmcblOp3` (a.k.a. `/balign`) to become more than a certain level. Like the *rasdimd* daemon it is a symbolic link to the current version. Out of date files in `/balign/data` are removed at regular intervals. If necessary it restarts the EPC due to the mysterious increase of disk-usage. See also Appendix 7.1 item 5 for more details. The daemon uses the following *.ini* file: `cleaner.ini`. The parameters are shown in table 5.1.

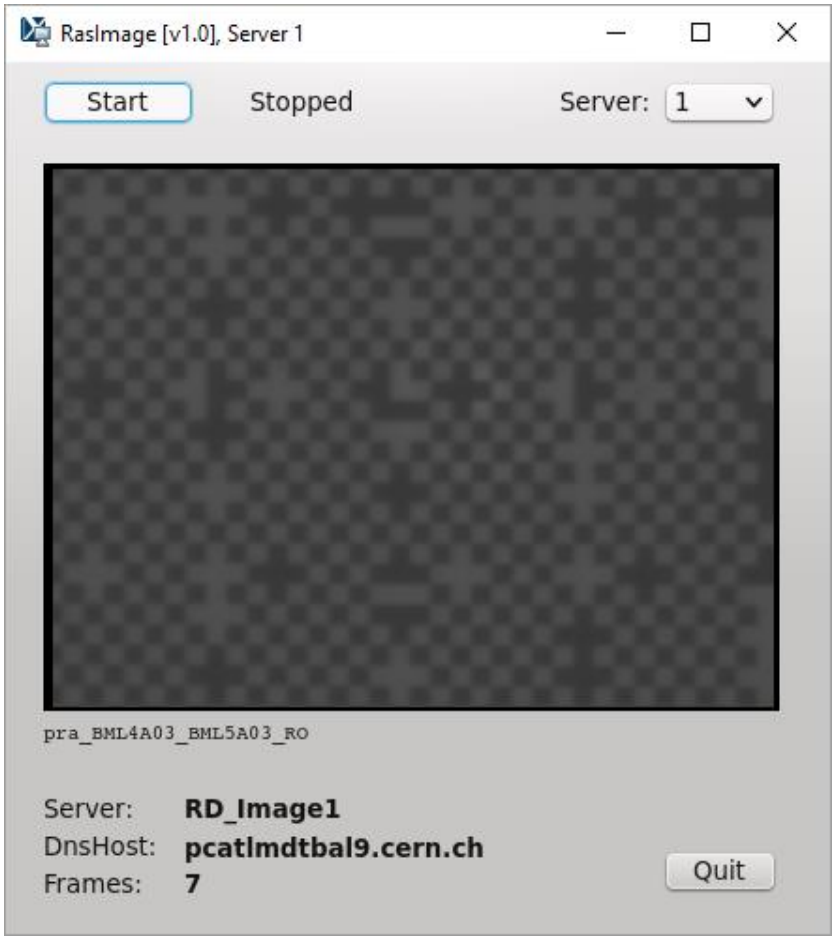
parameter	***default***	***description***
Version	`1.0.1`	Current version
LogDir	`/balign/data/log/clog`	Log folder
Interval	`14400`	Loop time, 14400 seconds is 4 hours
Months	`2`	Files older than 2 months are removed
CleaningDay	`4`	The 4th of each month the /balign/data area is checked
DiskUsage	`80`	If higher, reboot the board
RdLogDir	`/balign/data/log`	Directory of the log files
RdImageDir	`/balign/data/images`	Directory of the images
RDResultDir	`/balign/data/results`	Directory of the result files
RdLogHdr	`rdlog_`	Header of the log files
RdImgHdr	`img`	Header of the image directories
RdResultHdr	`rd_`	Header of the result files

Table 5.1: cleaner.ini configuration parameters

The daemons *rasdimd*, *wdogd*, *cleanerd* and the shell script *rasdim_watchdog* are member of run-level 5 and started at bootup. They are added (as symbolic links) to the directory: `/etc/rc5.d`. For information where to find the source files of the programs and how to compile them, look at Appendix 7.5.

5.1 RasImage

The *RasImage* program shows the images taken by the Rasdim server. In fact it connects to its DIM service providing the images. There is a Windows version, mainly used at the production sites and development setups, and a Linux version used at ATLAS. The ATLAS version is based on *Qt* and written in C++. Due to the limited distribution of *Qt*, the *RasImage* binary can only be executed on *pcatlmdtbal9*. Picture 5.1 shows its view. The server combo-box sets the server to BAL[1-8].



Picture5.1: RasImage viewer

Execute the following command on *pcatlmdtbal9* to start *RasImage*:

```
$ export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/sw/atlas/sw/lcg/external/qt/4.8.4/x86_64-slc6-gcc47-opt/lib
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/atlas-home/1/rhart/projects/dim/linux/
$ /atlas-home/1/rhart/projects/Rasnik/Release/RasImage
```

NB: the *RasImage* binary and the *Qt* distribution have to be moved as soon as possible to the general Production area: `/det/dcs/Production`.

6 Miscellaneous

6.1 WinCC

The WinCC projects ATLMDTBAL[1-8], acting as a client of the Rasdim server, all share a common repository containing the scripts and panels. They can be found inside the common ATLAS DCS area: `/det/dcs/Production`. Inside this area, each of the ATLAS sub-detectors has its own directory. The barrel alignment belongs to the MDT sub-detector, hence all files, scripts, panels, etc. can be found inside `/det/dcs/Production/ATLAS_DCS_MDT`. The MDT WinCC projects reside here. The common one for the alignment projects is called: `ATLMDTBAL`, hence the *config* file for the alignment projects contains the line:

```
proj_path = "/det/dcs/Production/ATLAS_DCS_MDT/ATLMDTBAL"
```

The supervisor project ATLMDTBAL9 (running on *pcatlmdtbal9*) includes an additional project line providing specific scripts and panels.

```
proj_path = "/det/dcs/Production/ATLAS_DCS_MDT/ATLMDTBAL/ATLMDTBALSRV"
```

6.2 Supporting Daemons

The (DIM!) *dns* server and the *soap* and *spot* servers are all started automatically by means of specific scripts at boot up and checked regularly at specific intervals just like the *rasdim_watchdog* script shown in Appendix 7.3.

The *dns* server runs on *pcatlmdtbal9* and on both *pcatlmdtbal11* and *pcatlmdtbal12* a *soap* and a *spot* server is running. The scripts for starting and checking the *soap* server and *spot* server can be found in `/det/dcs/Production/ATLAS_DCS_MDT/linuxScripts` and are called `bal_soap_watchdog.sh` and `bal_spot_watchdog.sh` respectively.

7 Appendices

7.1 Topic EPC drawbacks

The Topic EPC boards with its specific Linux system do have a couple of drawbacks.

1. **IPMI**: not supported. There is no possibility to monitor remotely the load and memory usage of the EPC. For the same reason it is not possible to power on and off the board. Hence, whenever a power-cycle is needed it has to be done manually.
2. **File-system** check/repair: there is no tool or application, like *fsck(8)*, to repair a corrupted file system. Hence, the EPCs are vulnerable to power-cuts. If corrupted, the only way to repair it is to replace the µSD-card.
3. **Permanent Time-out**: The frame-grabber (the ADC of the Xylinx FPGA) sometimes enters a mode in which it cannot recover, except by a reboot of the EPC. See also chapter 4.4 for more details. The *libiio* package of Analog Devices does not provide a function to perform a hard reset of the ADC.
4. **Domain Name System:** The EPC Linux system lacks the full usage of DNS, hence everything is accomplished by real IP-numbers.
5. **Disk-usage** of `/dev/mmcblOp3` (a.k.a. `/balign`): on this file-system most activity concerning opening, writing and closing of files takes place. There are 2 ways in which the disk-usage of /balign increases:
    a. The usual way during the image handling by adding data to the log, result and image files.
    b. The mysterious increase of the disk-usage during image handling. The allocated space of the files does not reflect the space left according to the *df(1)* command. After a reboot the not understood occupied disk-space is returned.

7.2 Frame grabber IP

The base address of the frame grabber IP block is 0x43D00000. The following table shows the list of registers and their meaning.

Offset	*Mode*	*Name*	*Description*
0x000	RW	control	bit 0: framegrabber_enable bit 7: testmode_enable
0x004	RO	status	bit 0: framegrabber_idle bit 1: framegrabber_active bit 2: framegrabber_fifo_active bit 3: framegrabber_in_sync bit 6: fifo_write_error bit 7: fifo_read_error
0x040	RW	h_sync_width	width of sync pulse (in clocks/pixels)
0x044	RW	h_bp_width	width of back porch (in clocks/pixels)
0x048	RW	h_active_video_width	width of active video, pixel data (in clocks/pixels)
0x04C	RW	h_fp_width	width of front porch (in clocks/pixels)
0x050	RW	sync_level_min	lower limit if video signal is at sync level
0x054	RW	sync_level_max	upper limit if video signal is al sync level
0x058	RW	blanking_level_min	lower limit if video signal is at blanking level
0x05C	RW	blanking_level_max	upper limit if video signal is at blanking level
0x060	RW	sync_drop_threshold	level to detect the sync pulse (if video signal drops below this level, start of sync pulse is detected)
0x064	RW	max_transition_width	number of clocks/pixels before signal moves from blanking to sync level (and vice versa)

0x068	RO	sync_level_min	used value
0x06C	RO	sync_level_max	used value
0x070	RO	blanking_level_min	used value
0x074	RO	blanking_level_max	used value
0x078	RO	sync_drop_threshold	used value

Table 6.1: Frame grabber IP

The registers from offset 0x040 until 0x04C (the registers names starting with `h_`) are used when the grabber found the sync within the pixel data line, so that it can skip the sync pulse and front/back porch.

The registers from offset 0x050 until 0x060 (the RW register names starting with `sync_` or `blanking_`) are used to find the sync pulse. The corresponding RO registers (offset 0x068 until 0x078) contain (when the dynamic leveling algorithm is switched on to find the sync) the values based on the input signal.

7.3 rasdim_watchdog

The rasdim_watchdog script:

```
#!/bin/sh
#
# rasdim_watchdog ------
# script to check whether the rasdim server is running
# if not, (re)start it
#
set -e
RASDIMSERVICE=rasdimd
RASDIMDAEMON=/etc/init.d/$RASDIMSERVICE.sh
TMPFILE1=/tmp/ps$$_out1
TMPFILE2=/tmp/ps$$_out2

if [ ! -f $RASDIMDAEMON ]; then
  logger -s -t "RASDIM" "FATAL: File $RASDIMDAEMON does not exist"
  exit 1
fi

while true
do
  sleep 60
  ps > $TMPFILE1
  grep -v grep $TMPFILE1 > $TMPFILE2
  if ! grep -q $RASDIMSERVICE $TMPFILE2 ; then
    logger -s -t "RASDIM" "WARNING: $RASDIMSERVICE not running, try to restart"
    $RASDIMDAEMON start
  fi
done
exit 0
```

7.4 adc_offset

The adc_offset script:

```
#!/bin/sh
echo "base = 0x43D00000
echo -n "max_transition_width (0x43D00064) = "; devmem 0x43D00064
echo -n "sync_level_min       (0x43D00068) = "; devmem 0x43D00068
echo -n "sync_level_max       (0x43D0006C) = "; devmem 0x43D0006C
echo -n "blanking_level_min   (0x43D00070) = "; devmem 0x43D00070
echo -n "blanking_level_max   (0x43D00074) = "; devmem 0x43D00074
echo -n "sync_drop_threshold  (0x43D00078) = "; devmem 0x43D00078
```

7.5 Sources and Compilation directives

The C++ sources of the programs are located at: `/project/ct/po/workspace/Topic`
Not only the program sources are located here, but also the sources of the supporting libraries like DIM and the *libconfigfile* package.

The *svn* repository for of Rasdim and Soap is located at: `/project/ct/po/svn`

The Xilinx SDK is used to (cross) compile and link the programs. Execute the following commands to use this SDK, which is based on the *Eclipse* IDE.

```
$ source /eda/fpga/xilinx/SDK/2016.4/settings64.sh
$ xsdk -vmargs -Dorg.eclipse.swt.internal.gtk.cairoGraphics="false"
```

It only works if you are member of the group *et* and have the following line in your .bashrc file:

```
export LM_LICENSE_FILE=@loue.nikhef.nl
```

The Rasdim sources are compiled with the following command:

```
arm-xilinx-linux-gnueabi-g++
```

and linked with the following arguments:

```
-L../../dim/lib -L../../lib
-ldimcxx -ldimc -lconfigfile -lpthread -lcurl -lgnutls -lnettle -lhogweed
-lgmp -lz -liio -lusb-1.0 -lavahi-client -lavahi-common -lxml2 -ldbus-1
```

For the Rasdim server an extensive *Doxygen* page exists. It can be found at:

https://www.nikhef.nl/nikhef/departments/ct/po/Atlas/Rasnik/Rasdim/html/index.html