Pixel DAQ Manual

Pixel DAQ Team

October 2, 2019

Contents

Chapter 1

Setting up the system

1.1 General policies on working in SR1

This chapter summarises what needs to be done in order to start working with the detector in SR1. Some parts are described in more detail in other areas of the documentation.

Several users work in SR1 lab for developments. If you want to work on a particular stave you should book it first. Here is the procedure:

- Go to this link: The WorkPlanner
- Select *"ToothPix Schedule"* from the drop down menu and then *"Enter..."*
- Click on *"My Shifts"*, navigate to the date you want to book the sister and select the staves you want to use
- For each stave tick ☐ DCS (For detector control) and ☐ DAQ (for allocation rights) boxes. In the case you need to run local data taking also tick the ☐ LTP (local trigger processor) box
- Click on Update My Shifts

Some caveats:

- Don't book slots if you are not going to use them. If something happens and you can't use them, please un-book.
- Try to not book slots for full days if not strictly necessary. If something needs long time for validation, book the night.
- Be flexible and interact with people: if you need something for a fast test, ask if you can allocate a ROD for a little bit on the SR1 Hotline or personal Skype chats. Expect other people to make such requests, from time to time.

1.2 Checkout and compile the code

The Pixel DAQ code repository is located in

```
https://gitlab.cern.ch/atlas-pixel/atlaspixeldaq
```

you can clone it to your home area in SR1 for example using

```
git clone https://<USER>@gitlab.cern.ch/atlas-pixel/atlaspixeldaq.git -b <BRANCH> <DESTDIR>
```

with your username, the branch you want to clone and the destination directory you want to clone the repository to. Currently the recommended branch is

```
sr1/PixelDAQ-02-07-01
```

Next step is to create a soft link to the <DESTDIR> like this

```
ln -s <DESTDIR> daq
```

this is important so that the various scripts know the location of your freshly downloaded code. The following step is to personalise the .xml configuration files to avoid clashing with other users. To do so just

```
cd daq/Applications/Scripts/
./FixXml.sh
```

Fixing the xml, files should be done only the first time, right after you checked out the repository. In case you need to revert the xml files you can do

```
./FixXml.sh - -
```

(Note some further manual cleanup might be needed - check with `git status`.) At this point should be good to compile the code. Just

```
cd $HOME/daq
make
```

In some cases the compilation might fail because of race conditions in the Makefile that might not been completely ironed out at the time of the writing of this note. If this is the case, please try to make the project again. If the compilation was successful, go to

```
cd Applications/PixLib/Examples/
make
```

to make a set of utilities that are needed in a later stage. In order to be able to run data taking, the pixRCD needs to be compiled

```
make pixRCD
```

You will probably also need the BOC commands. They are not required to run callibrations or data taking, but are useful utilities to inquire the BOC status.

```
cd $ROD_DAQ/IblUtils/Boc2
source ./setup.sh
make
```

1.3 Load the Software and the Firmware to the ROD

For the following steps you need to know the names, IP addresses and attached "cables" (JTAG programmers) of the computers in SR1 which can be found in the following spreadsheet:
https://docs.google.com/spreadsheets/d/1s-ktpfiLgkcWlEr5HXKpSdO-Cmn3PE66lwZ5NXgxGcA/

The software for the ROD PPC and for the ROD Slaves is not compiled by the main Makefile. First go to the PPC code location, then compile the code

```
cd $PPCpp
make
```

The environment variable

```
$PPCpp=daq/RodDaq/IblDaq/RodMaster/Software/PPCpp/
```

is a very useful one. Then is necessary to compile the Slave software:

```
cd $HOME/RodDaq/IblDaq/RodSlave/Software
make
```

For some reasons not completely ironed out yet, the Makefile of the histogrammer is bit sloppy: it might be necessary to hit make up to three times to compile properly the histogrammer code. Please check that after the make there are present the `.elf` files in the `bin/` folder. Once the code is compiled one can load it to the desired ROD using the VME bus or the Xilinx JTAG cable. Check Sec.5.9.1 for more information.

- Xilinx cable:

  ```
  cd $HOME/daq/scripts; iblRodCtrl <RODNAME> boot|start all|slaves|cpp
  ```

- VME bus:

  ```
  /det/pix/daq/binaries/IBLROD/M8/iblRodCtrlVme ROD_NAME --flashFw --flashSw --pixFw|iblFw
  (--run)
  ```

  where `--flashSw` flash the software, and `--flashFw --pixFw|iblFw` flash the firmware, remember to specify the FE flavour for IBL or Pixel, depending on the ROD that you are working on. Fur-

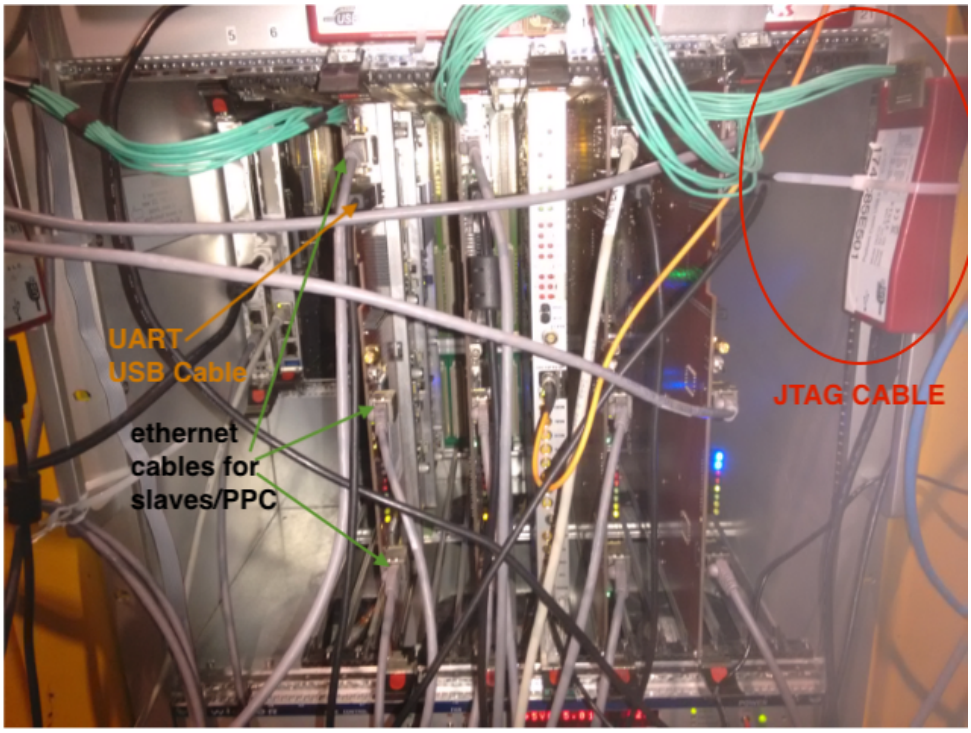


Figure 1.1: Picture of SR1 RODs in one Crate

ther expert options are available like loadPROM and restartPROM. The `--run` option executes the command. First do a dry run without it to check the files are all available in the proper locations.

1.3.1 Setup a UART log

The standard output of the ROD PPC or the ROD Histogrammers can be read back via an **UART Lite**. There are different methods to retreive the UART log from PPC and slaves. One way is to connect a USB cable from ROD to an host machine, then the UART is read back using c-kermit, this method is being deplecated, however is possible to retreive the UART via USB upon request by running `screen /dev/ttyUSB0 115200` on the host machine. A different method to retreive the UART is using the VME BUS on the SBC, this method could interact with the flashing of the ROD via VME, so a further protection has been added. In SR1 different scripts have been implemented for a safe readback of the UART via VME that will be described below:
To set up the read back run the following command:

```
/home/pixeldaq/scripts/startVMEUart SBC_name
```

Where SBC_name is the name of the SBC (pixrcc03 and pixrcc06 for Crate1 and Crate3 respectively). Logs are saved to a file (the path is displayed on screen), which you can view realtime using:

```
/home/pixeldaq/scripts/viewVMEUart ROD_NAME
```

This script will start the VME UART readed in case that is not running. You can browse the displayed log pressing "b" in the window then search for a word or scroll up and down using arrows or pgUp or pgDn. To close the browsing mode press "q". Alternatively you can retreive the logs from the following location `/det/pix/logs/VmeUart/ROD_XX_XX/PPClog_current`. There is also the possibility to log the output from the slaves, to do this you just have to append to the command the source that you want to read:

```
/home/pixeldaq/scripts/viewVMEUart ROD_NAME north|south|master
```

By default only master logs are stored. Please, remember to stop storing slave logs after you are done with your debugging, to do this just run viewVMEUart with the `master` option. On the other hand, from time to time can be useful to stop the logging process using the following command:

```
/home/pixeldaq/scripts/stopVMEUart SBC_name
```

Note that when the start and the stop of the VME UART acts on all the RODs belonging to the SBC.

The lines in the UART log might be bit messed up, this is because the PPC application is a multithread application and the threads can print out things at the same time.
Fig. 1.2 shows a few lines of example of dump of the UART log right after the booting of the PPC.

Please note that the IP of the PPC (192.168.2.150) and of the two slaves (192.168.1.151, 192.168.1.152), at the end the Firmware and Software versions are printed out and the PPC application is ready to receive commands via Ethernet. The warning on the Git Hashes version is deprecated and can be disregarded.

1.4 Turning up DCS

In order to turn on the modules in SR1 it is necessary to start the DCS consoles. There is one for Pixel and one for IBL, which are started by different scripts.

- IBL FSM panel `source /dcs/scripts/newATLPIX_SR1_IBLFSM.bash`
- PIX FSM panel `source /dcs/scripts/startATLPIX_SR1_FSM.bash`

In Fig. 1.3, the DCS console for Pixel staves in SR1 is shown.

- **Power On procedure:**
  - After opening DCS select the stave that you want to turn on. The SR1 spreadsheet shows which stave belongs to which ROD.
  - First the optoboard needs to be powered up, then LV needs to be turned on and finally the High Voltage. Left Click over the command box in DCS (see 1.3 second figure) to see the available commands. Issue, in this order:
    * `SWITCH_ON_OPTO` → `OPTO_ON`
    * `SWITCH_ON_LV` → `LV_ON`

```
16:15:33.067: ------lwIP Socket Mode Demo Application ------
16:15:33.067: Board IP: 192.168.2.150
16:15:33.067: Netmask : 255.255.252.0
16:15:33.067: Gateway : 192.168.0.100
16:15:33.067: MAC address: 000A3579110F
16:15:33.067:
16:15:33.067: link speed: 1000
16:15:33.067: Starting xemacif_input_thread. THREAD_STACKSIZE: 2097152 2
16:15:33.068:
16:15:33.068:                Server   Port Connect With..
16:15:33.068: --------------------- ------ --------------------
16:15:33.068:          echo server      7 $ telnet <board_ip> 7
16:15:33.068: Command Server listening on port  10763
16:15:33.068: IPBus server listening on port  50000
16:15:33.068: QuickStatus network thread up and running...
16:15:33.068: QuickStatus daemon up and running...
16:15:33.068: LogPublisher daemon up and running...
16:15:33.068: Reconfig at ECR daemon up and running...
16:15:33.068:
16:15:33.068: XVCD: Listening to incoming connections on TCP port 2542.
16:15:33.068: Setting slave network configuration for slave 0
16:15:33.068: MAC: 00-0a-35-79-12-0f
16:15:33.068:
16:15:33.068: IP: 192.168.2.151
16:15:33.068:
16:15:33.068: Mask: 255.255.252.0
16:15:33.068:
16:15:33.068: Gateway: 192.168.0.100
16:15:33.068:
16:15:33.068: Created Socket 2
16:15:33.068: Starting QuickStatus daemon...
16:15:33.068: Setting slave network configuration for slave Created QSNetwork socket 3
16:15:33.068: 1
16:15:33.068: MAC: 00-0a-35-79-13-0f
16:15:33.068:
16:15:33.068: IP: 192.168.2.152
16:15:33.068:
16:15:33.068: Mask: 255.255.252.0
16:15:33.068:
16:15:33.068: Gateway: 192.16Connected to the server
16:15:33.069: Starting SysStatus daemon...
16:15:33.069: 8 server
16:15:33.069: Starting RecoAtECR daemon...
16:15:33.069:
16:15:33.069:  Reco At ECR loop closed. Exiting...
16:15:33.069: Acquired QS pointer at D6979C8
16:15:33.069: Slave 0 SW hash: bd6ba756d65900e021bbecadd0bfb61a156ae047
16:15:33.069: Slave 0 FW hash: a0a047421b8ebb22d90facf3b48db0da6aeda35
16:15:33.069: Slave 1 SW hash: bd6ba756d65900e021bbecadd0bfb61a156ae047
16:15:33.069: Slave 1 FW hash: a0a047421b8ebb22d90facf3b48db0da6aeda35
16:15:33.069: Master SW hash: bd6ba756d65900e021bbecadd0bfb61a156ae047
16:15:33.069: Master FW hash: 6c0.
16:15:33.069: IPBUSD: Listening to incoming connections on TCP port 5000
16:15:33.069: Socket associated with IP: 0 and port: 49153
16:15:33.069: Starting LogPublisher daemon...
16:15:33.206: bith IP: 0 and port: 49153
16:15:33.267: 22af608ef12
16:15:33.267: WARNING: Git hashes are inconsistent!
16:15:33.267: Starting server (waits for connections from a client)...
```

Figure 1.2: Example of a UART log right after the boot of the PPC. Some lines are messed up due to concurrent threads. Image produced on 25th January 2018, might be obsolete.

* `SWITCH_ON_HV` → `HV_ON`

The commands need some time to be propagated as well as the voltages and currents need some time to ramp up. During this time the command box will display `TRANSITION` state. You might experience DCS WARNINGs or ERRORs during the ramp up. They should be transitory and you can disregard them. In the case they stay after the transition state is

finished and the staves are in states such as `OPTO_ON`, `LV_ON` or `HV_ON`, then turn off the system and contact the SR1 responsible or the DAQ coordinators.

- **Monitor the pre-amplifier state or other quantities:**
  - It is always possible to check the trends and history of currents / temperature / voltages of the modules right-clicking on the correspondent boxes showing the current values on the DCS panel. For example, right-clicking on the temperature of module M5A in Fig. 1.3, the temperature trend will be shown. Same holds for the other quantities. Note: archiving is not available for the second half of 2018!
  - If the modules have been configured with preamplifiers on (see **??**), the DCS state should be in `READY`. (This is not always true in SR1 since the DCS limits are not well defined, but should be always true in the PIT.)
- **Power Off procedure:**
  - Make sure to configure the modules to pre-Amplifier off state. The state should read `HV_ON` or `STANDBY`
  - At this point is possible to bring them back to `STARTED`, left-clicking on the status box and issuing `BACK_TO_STARTED` command. This command will follow the power on procedure in the reverse order and turn off the modules safely.

1.5 Starting up the infrastructure

The high-level code communicates with the RODs through the Pixel Infrastructure, which is a collection of applications that run on different machines, for details see Chapter 2. In order to run data taking and calibration, the Pixel Infrastructure needs to be "up". The first time the Pixel Infrastructure is started, it is necessary to be initialized (`INITIALIZE`) and then configured `CONFIGURE`.

The configurations are stored in a database and the only thing needed is to publish to the Information Service (IS) the right tags that will point to the actual configurations that the Pixel Infrastructure will pick up. To do that it is necessary to use the Tag Manager application. You can start it just typing `TagManager` (Remember to compile the PixLib examples). You need to select the right tags from the drop-down menus, see Fig. 1.4. The following options are valid for SR1 [1]:

- **Connectivity Domain**:*Connectivity-SR1*
- **Connectivity Tag**: *SYSTEST-V6*
- **Configuration Domain:** Configuration-SR1. Should be the only option.
- **Configuration Tag:** *NEWDSP-DEFAULT* is OK.
- **Module Configuration Tag:** The default, *MODCFG-DEFAULT* is the default configuration for SR1. This tag points to the configuration of the modules, which can be modified manually, see **??**, or by a tuning, see **??**. Select the one you want or need accordingly.

[1] For the PIT, talk to Pixel Run Coordinator.

- **PixRunConfig:** This is the run configuration, i.e. the configuration that is used to run local data taking.
  - NORM_1A_80 would setup normal data taking, one BC read-out window and 80 Mbit/s read-out speed. SIM_1A_80_20hits instead uses the BOC emulator to simulate hits from the detector (useful to factorise out the detector), one BC read-out window, 80 Mbit/s read-out speed and 20 hits per module (module?).
  - NORM_1A_160 would setup normal data taking, one BC read-out window and 160 Mbit/s read-out speed.

  There are various configurations to choose from and you can edit your own. See Sec. 3.2.2.
- **Warm Start Disable:** empty is fine.
- The panel **Pixel Flags** can be left as it is and doesn't need to be modified. In case of doubt, talk with Pixel Run Coordinator, don't touch it yourself.
- Be sure that **Update XML files** is *not ticked* and click on **Publish Information in IS**. You'll see that the section under **Tag in IS** will populate. You are done and can close TagManager.

1.6 Starting the Calibration Console

Pixel and IBL calibration is triggered by the Calibration Console application. To start the calibration console, it's enough to type `Console -e` in the command line. The `-e` flag means expert use, and it's used to allow to turn on the pixel pre-amplifiers. Therefore you need to use it in the case you want to really perform calibration. If you don't, the STANDBY tag will be published in IS automatically, that will prevent you to perform such operations. Once you start the Console you'll be prompted the GUI shown in Fig. 1.5. A good reference to the Console functionalities is given in the Pixel Detector Calibration Manual. Some of the function

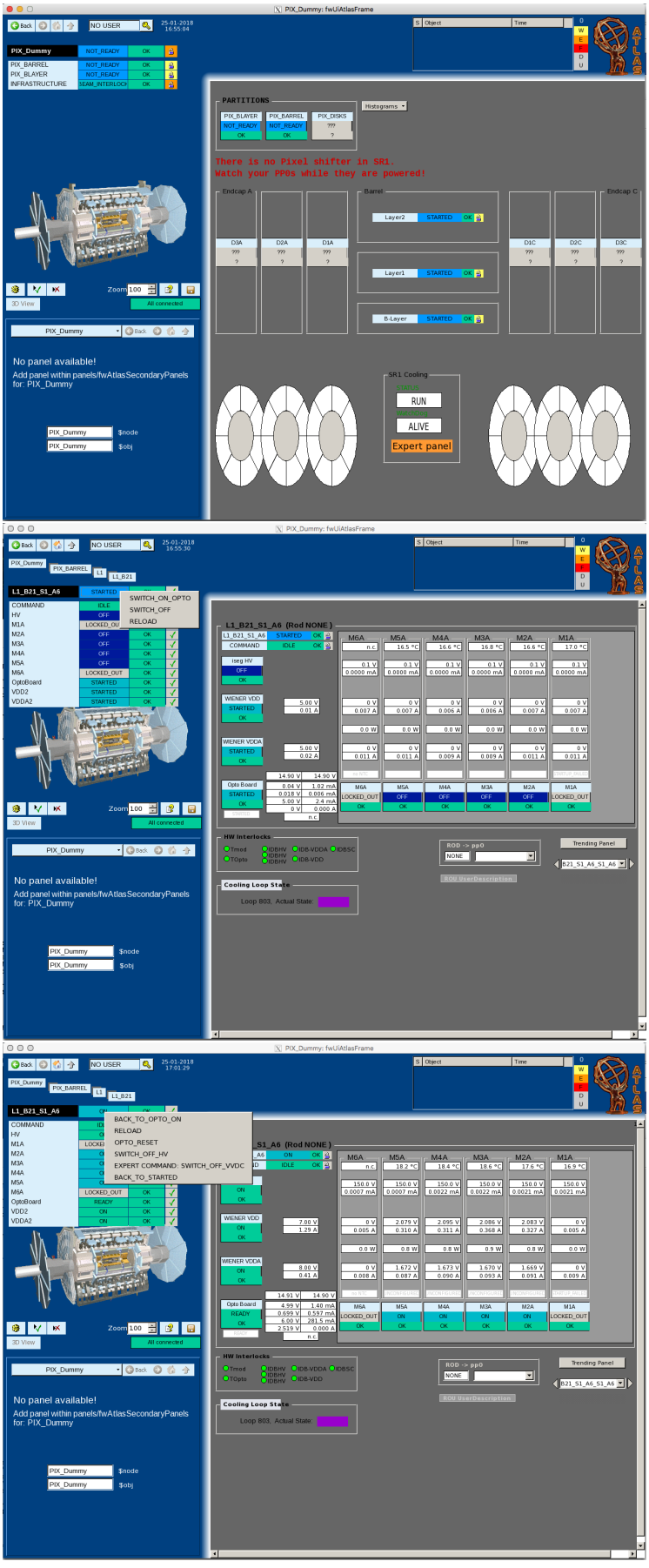


Figure 1.3: Screenshots of the DCS console to activate the modules

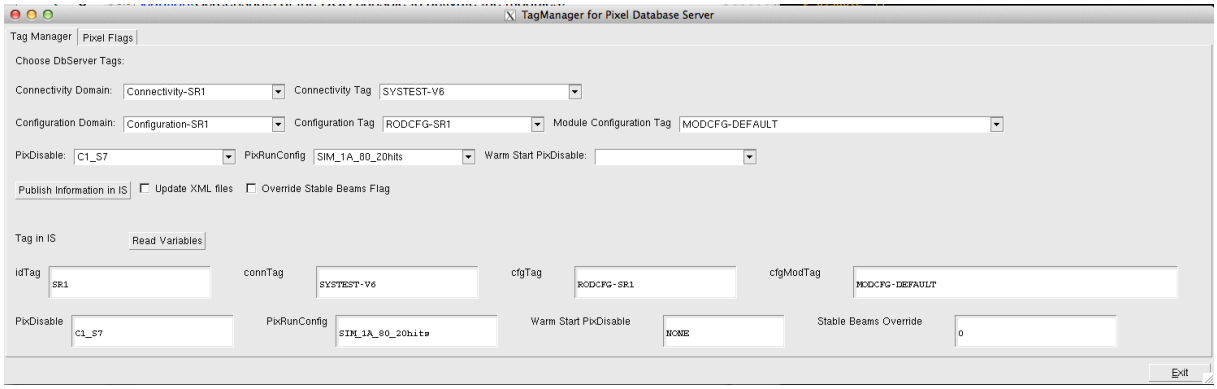


Figure 1.4: TagManager GUI

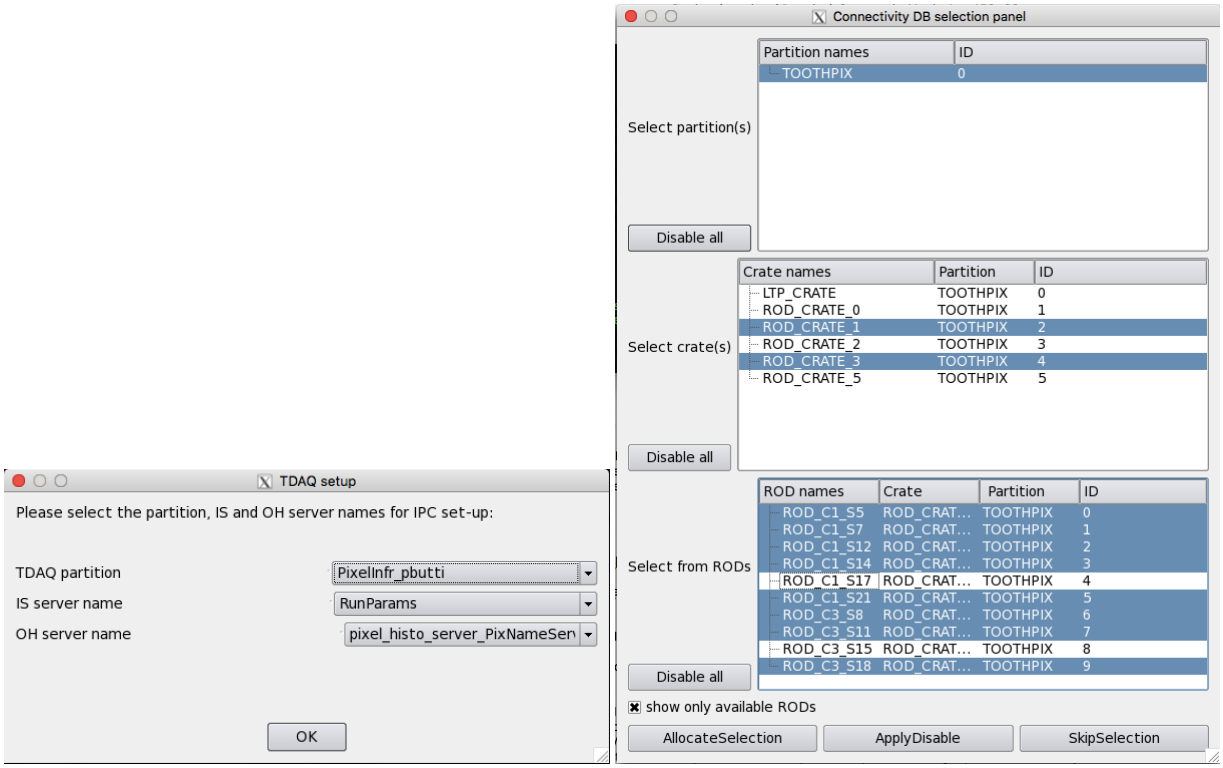


Figure 1.5: Left: first screen at the opening of the console. Default values are fine. Right: rod allocation screen

Chapter 2

Pixel Infrastructure

The *Pixel Infrastructure*, in jargon *The Actions*, has a similar structure and GUI of the ATLAS infrastructure. The scope of this document is not to go into the details of the GUI, as they can be learnt from ATLAS shifter training, but rather help a new Pixel DAQ user to understand the Pixel related details. There are some differences between the SR1 and PIT infrastructure, but since the general structure is very similar this document will explain the SR1 actions in more details. In this chapter will be discussed, among various other things, what are the applications controlled by the actions and where do they run, how to add an additional application to the infrastructure, how to debug the actions applications using `valgrind` and/or `gdb` and how to locate and read the actions logs.

2.1 Starting the infrastructure

In this section are described the steps that need to be taken in order to start the actions from a new installation of the `pixeldaq` software. The first thing to do after a successful installation of the pixeldaq software is to set up the database tags that need to be picked up by the actions. In fact, the whole set of parameters describing the module configurations, the run configurations,

2.2 Checking the database, changing module configuration

- `$PIX_LIB/Examples/DbServerStatus`
- `PixDbBrowser`
- `$PIX_LIB/Examples/ChangeCfg`

2.3 Set up the working conditions

- Compile PPCpp

```
Connecting to partition PixelInfr
Trying to connect to DbServer with name PixelDbServer  .....
Succesfully connected to DbServer PixelDbServer

================================================================================================
 Tag                                                    # objects   # of revs   Size (KB)
================================================================================================
 Configuration-PIXEL18//PIT_BOC                            2344        2348        3374
 Configuration-PIXEL18//PIT_MOD                          117185       58842      123409
 Configuration-PIXEL18//PIT_MOD_IBL_TUNE                 117185       58923      124270
 Configuration-PIXEL18//PIT_MOD_IBL_TUNE-S000069878        2016        2016       21489
 Configuration-PIXEL18//PIT_MOD_IBL_TUNE-S000069879        2016        2016       21489
 Configuration-PIXEL18//PIT_MOD_IBL_TUNE-S000069880        2016        2016       21489
 Configuration-PIXEL18//PIT_MOD_STANDBY                  117175      117175      246452
 Connectivity-PIXEL18//PIT-CONN                            3461        3461        1174
================================================================================================
 Total number of revisions       246798      Total size (KB)       563708
================================================================================================
```

Figure 2.1: Output of DbServerStatus command

- Compile Slaves

- Flash ROD

- Disable the dumping of the bin.

2.4 Clean up remote control of the Pixel Infrastructure in P1

To gain control of an already running infrastructure select "Control" in the menu "Access Control" in iGui.

In the case you cannot gain the command of the Pixel Infrastructure because someone still has the control of it, one could force that user to release the control. First start the remote GUI:

```
start_rm_gui
```

You can check which infrastructures are up, select the one you want to gain control of and free the resources of the user control. This is useful in the case someone in the SCR forgets the infrastructure under his own control and left. Better to crosscheck with the person before doing this operation, but might be useful when urgent actions are required and the person is unresponsive.

2.5 Command line control of the Infrastructure

It is possible to use terminal commands to control the status of the infrastructure. For the help just do:

```
rc_sender -h
```

In SR1, once you ran the `start_infr` command and you have the infrastructure up, you can change state to `RUNNING` with this sequence of commands:

- `rc_sender -c INITIALIZE -p PixelInfr_<USER> -n RootController`

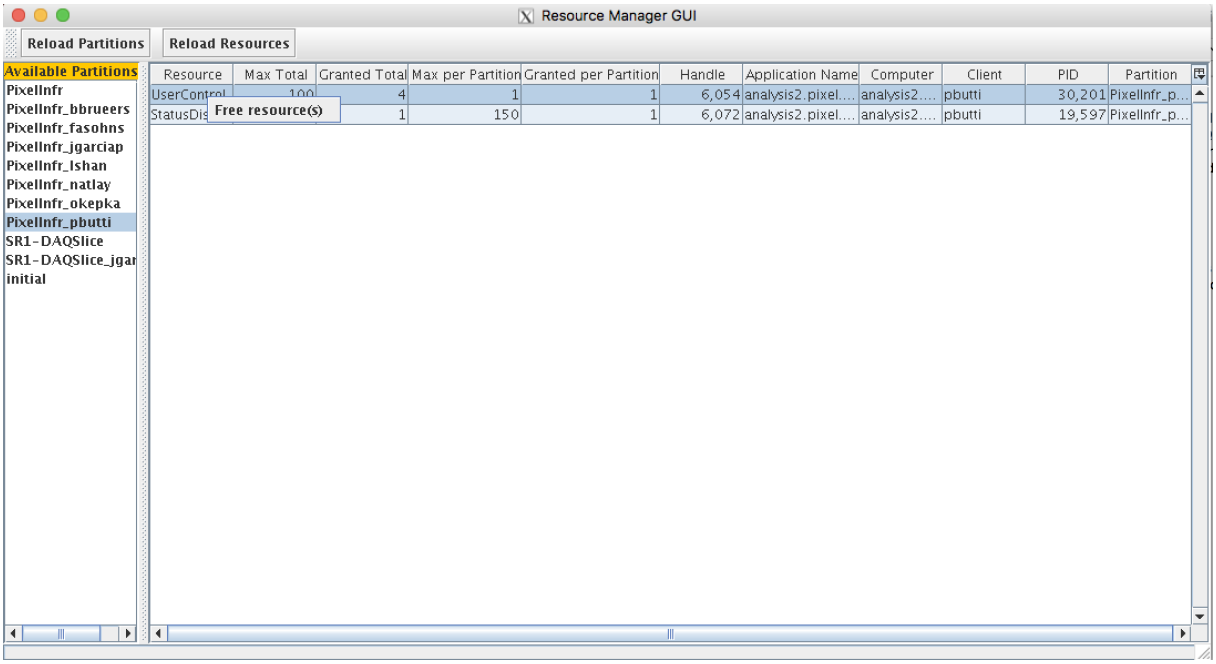


Figure 2.2: Remote gui

- `rc_sender -c CONFIGURE -p PixelInfr_<USER> -n RootController`

- `rc_sender -c CONNECT -p PixelInfr_<USER> -n RootController`

- `rc_sender -c START -p PixelInfr_<USER> -n RootController`

The name of the controller should be "RootController" in SR1 and the same for all the infrastructures. To hold or resume a trigger use `trg_command_trigger`:

- `trg_command_trigger -p SR1-DAQSlice_<USER> -n RCDPixelTtcCrateSr1 -c HOLD`
- `trg_command_trigger -p SR1-DAQSlice_<USER> -n RCDPixelTtcCrateSr1 -c RESUME`

2.6 Segment changes in SR1 and in the PIT

In order to change ...

2.7 Clean up and kill the partition

In some rare cases, such as crate powered off and on, restart of the machines where pmgserver runs or similar disruptions, the infrastructure can be in a *bad state* not easily recoverable by just closing the GUI,

kill/restart the single applications or restart the infrastructure. A possible way to recover is to kill all the applications in a more clean way using the pmg commands. First list the active processes running in your partition:

```
pmg_list_partition -p PixelInfr_<USER>
```

If there are some, kill them by:

```
pmg_kill_partition -p PixelInfr_<USER>
```

Then if you list again the active processes, the list should be clean. You should now be able to start the partition regularly.

2.8 Debug a pixel infrastructure application

In order to debug some of the PixLib applications it's sometimes necessary to run *Valgrind* and/or *gdb*. As described above, the various applications are spawned directly by the partition through the *Process Manager Server* (pmgserver), which runs on the various hosts. In the following is described the procedure.

2.8.1 Using Valgrind

Since the applications are started automatically, in order to run *Valgrind* on one of the executables it's necessary to 'trick' the actions to run a particular script with the same name of the application that needs to be debugged. Here is the procedure for running *Valgrind* on the Fit Server, as an example

- Locate where the application executable is located. The FitServer executable, for example is located[1] in `Applications/installed/x86_64-slc6-gcc49-opt/bin/PixFitServer`
- Change the executable name, so the actions won't run it directly
  `cp PixFitServer PixFitServer.exe`
- Create a new script with the same name of the old executable
  `touch PixFitServer`
  And write in it the following:

  ```
  #!  /bin/bash
  /usr/bin/valgrind --trace-children=yes <complete_path_to/PixFitServer.exe> $@
  ```

  The '$@' is necessary to run the PixFitServer with the right infrastructure parameters.
- At this point restart the application from the infrastructure gui (or restart the actions). The script should be called and the FitServer will run under Valgrind. You can create a suppression file if needed, check Valgrind webpage on how to do that.
- The Valgrind output is flushed in the .err log, while the executable output is still in the .out log. You can change the logging if wanted.

[1] The path given here is valid when using TDAQ6, as it uses gcc 4.9. For TDAQ7 the compilation folder changes as we use gcc 6.2

- You should now be able to debug using Valgrind tools

Important: Proper cleanup

There is an important thing to notice. Let's still consider the case of running Valgrind on the FitServer. The process that is now started by the pixel infrastructure is the call to the script that starts Valgrind on the FitServer. This means that when the Actions are stopped only the script will be closed, not the Valgrind process. This means that if you restart the actions n times, you will have n Valgrind processes running on the fit machines. This is not a good thing as they will take memory and cpu. You need to be aware that it is necessary to kill them one by one.

In SR1 you own the processes so just `ssh` to the right host and `kill -9 <Valgrind Process ID>`. In the PIT the process is owned by *crpixel* user. In order to kill the process you will need to ssh to the right host and `sudo kill -9 <Valgrind Process ID>`, give your account password and that's it. Remember that you don't have full sudo rights, only a few commands, such as kill, can be issued under sudo.

2.8.2 Using gdb

Using gdb on an infrastructure process is quite straightforward in SR1. First thing is to `ssh` to the right host where the process runs. There just type

```
gdb -p <PID>
```

And you should be available to hook to the running application. Hit *continue* to start the application under gdb and wait for the crash to get the stack trace.

Chapter 3

Data Taking

This chapter illustrates how to run the local data taking in SR1. The work flow is valid also in the PIT with minimal changes to the commands. Such changes are listed in the following sections. **ToDo:Describe the usage of `TagManager`(sequence of launch (infr, initialize → `TagManager`, start infr), disable map and `RodMon`). Maybe a dedicated section on those, to refer from here**

3.1 Running Local Data Taking

To start the daq slice from the command line run

`start_daqslice_sr1` for SR1
`start_daqslice` for PIT.

After one runs the command, if no major problems with PMG Server occur, an iGUI similar to 3.1 should appear. If the DAQ slice is opened in display mode, gain control of it via the menu **Access Control → Control**. Here are listed the steps that need to be followed in order to run local data taking.

1. **Setup the Segments & Resources**: Open the "Segments & Resources" panel, expand it and set it up in order to match your disable. In the snapshot 3.2 is shown how is the setup to run local data taking for **only** C3_S15. You should notice that:
   - The PIX_ROD_CRATE_3_RODS is **enabled** and under this entry the ROD_C3_S15_SET is **enabled**, while the other RODs are **disabled**
   - The Segment_PixelROS02_RODS is **enabled** and under this entry the ROD_C3_S15_SET is **enabled** while the other RODs are **disabled**
   - The Segment_PixelROS02_GNAM is **enabled** and under this entry Appl_PixelGNAM is **disabled** while Appl_PixelROS02 is **enabled**.

   When one wants to enable item which is disabled and whose parent is also disabled (it would show as gray), one first needs to enable the parent item by hitting the "Commit & Reload" button. The dependent item state will change to settable.

   In the case `C3_S11` needs to be activated for data taking this is the procedure:

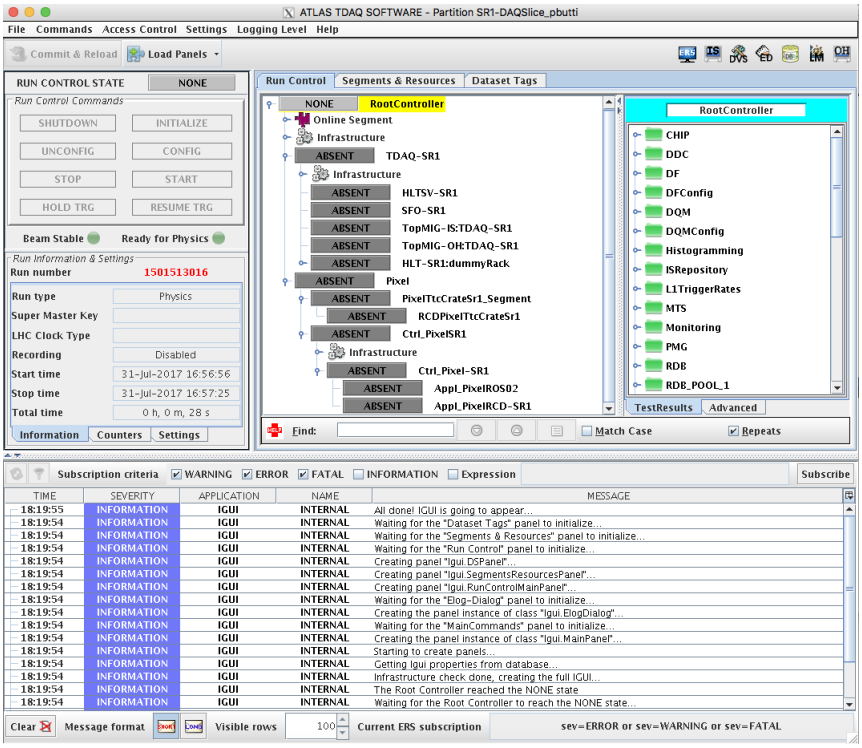


Figure 3.1: GUI window when the daq slice is opened. The various actions are grayed out since it has been opened in display mode.

- Right click and enable ROD_C3_S11_SET under PIX_ROD_CRATE_3_RODS
- Right click and enable Segment_PixelROS04_SET
- Click on Commit & Reload to pick up the changes, then Yes and finally click on the box and Reload (alternatively Reload all if you know about all the other changes only).

2. If there are changes in the segments one has to:

   - Click on Commit & Reload to pick up the xml changes.
   - Select Yes in the pop-up window that appears.
   - Click on the box relative the the xml changes one did and Reload. Alternatively it is possible to "Reload all" if sure of **all** the other xml changes, because everything will be reloaded.

3. The next step is to initialize the DAQ slice by clicking on `INITIALIZE`. The screenshot Fig. 3.3 (a) shows the DAQ slice in initial state. The pointed out error in the log is expected to appear in SR1 and can be ignored.

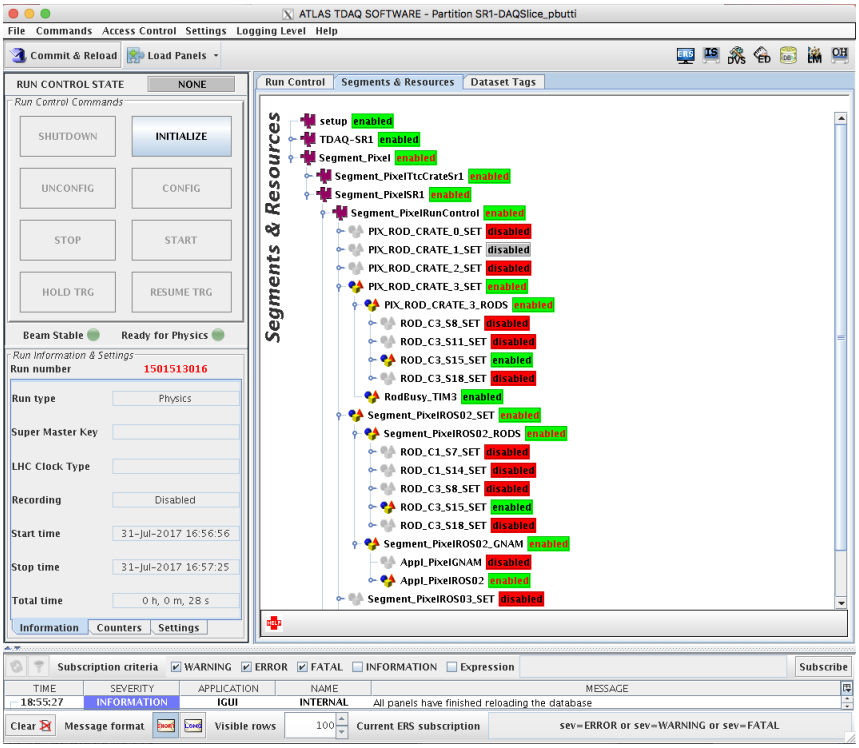


Figure 3.2: Example of the Segments & Resources to run data taking on —`C3_S15` in SR1.

3.2 Terminal commands for the DAQ slice

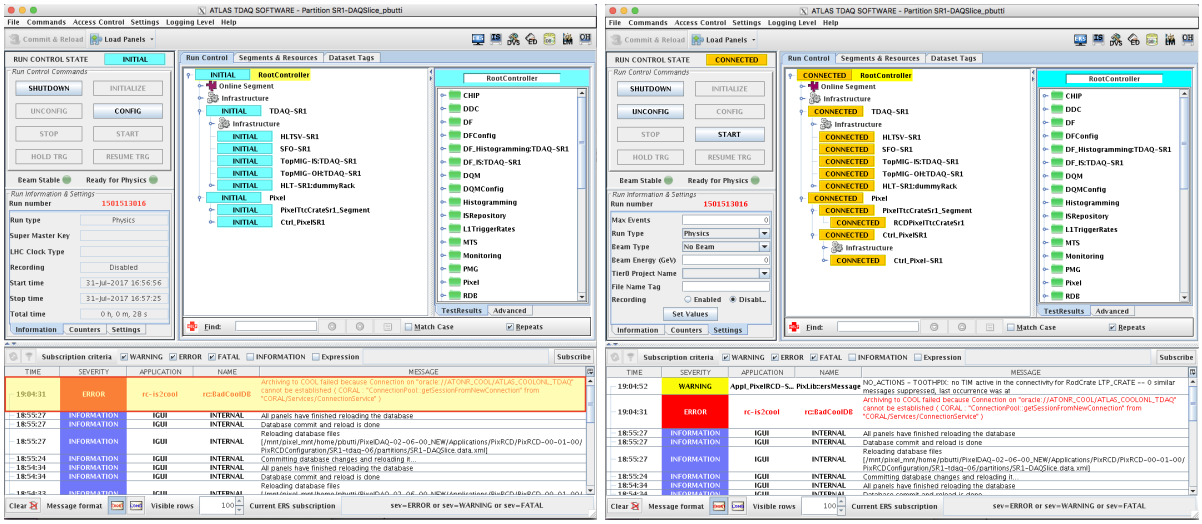
There are available a series of terminal commands to control the states of the DAQ slice. These commands are the same of the ones described in Sec. 2.5, changing the infrastructure name to `SR1-DAQSlice_<USER>`.

3.2.1 Some SR1 Data Taking trouble-shooting

The system in SR1 is not **extremely** stable and few problems can arise. Listed here there are some problems that are known at the time of when these notes have been written. The list here might be obsolete and these problems might not arise anymore or new ones might be listed present.

- With the switch to TDAQ7 all of SR1 has been connected to ROS `ros05` by default. Older ROS are still installed and can be connected on request.
- During the `INITIALIZE` transition the Appl_PixelROSXX[1]don't come up, e.g. due to PMG server communication issue. Try to `ssh` to the RosXX machine
  - If you can't `ssh`, go **physically** to SR1 and reset the ROS pressing the ON/OFF button.
  - If you can `ssh`, try to reboot the ROS connecting as `root`.

[1]Here "XX" stands for whatever ROS machine, e.g. 02 or 04



(a) (b)

Figure 3.3: The SR1 DAQ slice in `INITIAL` (a) and `CONNECTED` (b) states. During the two states the daq slice passes through `CONFIGURED` state. During this transition the module configuration is sent and can be monitored in the UART of the RODs.

- In the case you took a clean branch (which means validated code), you modified the segments and your L1 Trigger rate stays at 0, you probably made a configuration error: control that your segments match the disable tag that has been published in the `TagManager`. Then be sure you enabled the ROSes connected to the RODs you want to use for your data taking. If all looks OK and you can't still run local data taking, probably something is still wrong. Seek for a DAQ expert to look over your configuration and update this notes.

- Error message: `PixelSR1_LTP_MasterTrigger@LTPModules::setup() No valid clock detected Settings will be undetermined.` Normally, the `DAVE` LTP asynchronous card is used to provide trigger to the experimental setup. When the system is powercycled however, some settings is not properly propagated. A workaround was found to run the date taking *once* without the DAVE card after the powercycle and then re-enable it again. **ToDo:A better technical decription of the problem needed, consult JuanAn**.

  - Disabled the DAVE card and enable synchronous LTP in the following file.

    ```
    Applications/PixRCD/PixRCD-00-01-00/PixRCDConfiguration/
    SR1-tdaq-06/segments/Segment\_PixelTtcCrateSr1.data.xml
    ```

    `Commit & reload` the new changes:

    ```
    "RODBusyModule" "PixelRODBusy"
      </rel>
    - <rel name="Configuration">"LTPExpertConfig" "PixelSR1_ExpertConfig_wDAVE"</rel>
    + <rel name="Configuration">"LTPExpertConfig" "PixelSR1_ExpertConfig"</rel>
    ```

  - Perform a datataking

- Revert the changes in `xml` file and use the DAVE card.
- Configure the Dave card by running ~pixeldaq/DaveModule_TDAQ7/setupDAVE.sh

3.2.2 Modify the run configuration

The run configuration can be modified by command line using the `runConfEditor` script.

- SR1 `~pixeldaq/runConfEditor.sh --help`

  Example:

  ```
  ./RunConfigEditor --dbPart PixelInfr_mbindi --idTag SR1 --connTag SYSTEST-V6 --cfgTag
  RODCFG-SR1 --cfgModTag MODCFG-DEFAULT --runConf NORM_1A_160
  ```

- PIT

  `$PIX_LIB/Example/runConfEditor --help`

3.2.3 Change the L1 rate

It is possible to change the L1 rate via a simple script that changes the Dave Card settings in SR1:——
`~pixeldaq/DaveModule/changeRateDave.sh <n>`
This script sets the L1 to a rate $\nu = 40\text{kHz}/(2^{n-1})$. It can be used during the data taking session itself and doesn't require to stop run or hold the trigger.

In the PIT the script is located in

`/det/pix/daq/expert/`

and will connect to the `sbc-pix-tcc-02` to change the daveCard rate. Using $n = 1$ will set the L1 rate to 50kHz for IBL. For Pixel, instead, one changes the L1 through the `SetRunFrequency` command in `PixLib/Examples`.

3.2.4 Recording data

In the PIT is possible to record data from the local data taking. This can be used to run the noise mask, see 5.7, or for debugging purposes. In Figure 3.4 the recording panel in the DAQ slice is shown.

- The DAQ slice has just started and it's before **before** the **INITIAL** state.
- Access to the *Settings* panel
- Select the number of maximum events
- Select *Enable* for the Recording
- Click on *Set Values*

Recorded data is saved in the folder `/det/pix/data/localDataTaking/`. In the PIT the rate of the recording rate can be set modifying the HLT rate. The maximum recording rate has been found to be around $\sim$ 3kHz. Here is the procedure to modify the HLT steps (this is useful for NoiseMask for example, see Section 5.7

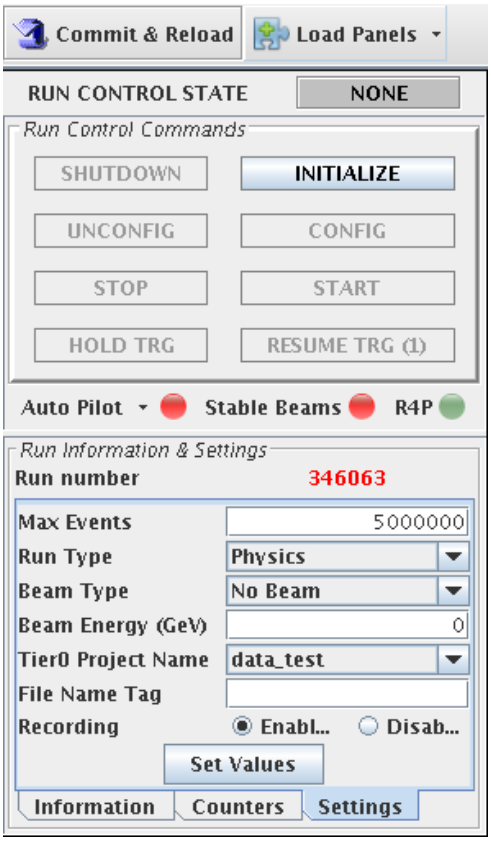


Figure 3.4: Control panel to record local data taking

- There are two separate files for IBL and Pixel, respectively:
  - `Applications/PixRCD/PixRCD-00-01-00/PixRCDConfiguration/PIT/segments/TDAQ-IBL.data.xml`
  - `Applications/PixRCD/PixRCD-00-01-00/PixRCDConfiguration/PIT/segments/TDAQ-PIX.data.xml`
- In these files change the HLT rate by changing the *l2Acceptance* variable to a different factor
- For example, if the L1 rate is 10kHz, putting 0.1 will run HLT at 1kHz

3.2.5 Decoding data

The following command can be used to dump a compressed RAW data file to a readable text format

```
dump-eformat-in-data.py -n nevents file.data
```

where `nevents` is the number of events to dump and `file.data` is the RAW data file.

Chapter 4

ATLAS Pixel DAQ Repository

This chapter provides an overview of the relevant ATLAS Pixel DAQ repository contents. Much of the repository is historical or need not be touched by the average user so only recently important parts are highlighted here.

Applications high level code

PixLibInterface link between RodDaq and Applications

RodDaq low level ROD code

4.1 Applications

CalibrationAnalysis analyze calibration results

CalibrationConsole Console application

DBeditor RootDb file editor

PixAnalysis analyze calibration results

PixCalibDbCoral interface to CoralDb

PixLib The PixLib

PixRCD connection between partition and daq slice (for data taking)

RawDataAnalysis code to analyze raw data

Scripts

4.1.1 PixLib

- Applications
- Bits
- Config
- ConfigRootIO
- Examples
- Histo
- PixActions
- PixBoc
- PixBroker
- PixConfDBInterface
- PixConnectivity
- PixController
- PixDbBrowser
- PixDbCoralDB
- PixDbInterface
- PixDbInterfaceFactory
- PixDbServer
- PixDcs
- PixelCORAL_interface
- PixFe
- PixFitServer
- PixHistoServer
- PixLibPolicy
- PixMcc
- PixModule
- PixModuleGroup
- PixRunConfigurator

- PixThreads
- PixTrigController
- PixUDPServer
- PixUDPUartReader
- PixUtilities
- RootDb
- pixlibdll

Applications high level code

PixLibInterface link between RodDaq and Applications

RodDaq low level ROD code

4.2 PixLibInterface

4.3 RodDaq

IblDaq ROD firmware and software

IblUtils ROD tools

HostCommandPattern

Boc2 BOC Commands

RodCrate interface to low level ROD/TIM code

RodUtils utilities to access low level ROD/TIM code

4.3.1 IblDaq

Historically called IblDaq because it was first implemented for IBL, but now is used for pixels + IBL.

Boc Firmware

Software

RodMaster Firmware ROD Virtex5 Firmware

Software ROD PPC Software

PPCpp PPC C++ code

CmdFactory user commands

Production

TestCommands

Tools

UnitTests
CmdPattern HCP (host command pattern) low level code
FrontEnd FE configuration / operations
QuickStatus QuickStatus
ReconfigAtECR Reconfiguration at ECR
Scan scans
SysConnect access to the PPC peripherals
Utilities
src low level code (based on thread example from Xilinx)
inc
boot bootloaders
bsp board support packages
xilkernel_bsp_lwip Default BSP for PPC

RodPrm ROD PRM firmware

RodSlave Firmware ROD Spartan6 Firmware
Software ROD Microblaze Software

new HCP have to be compiled / installed in three places:

- `$PPCpp`
- ...
- `make -C ${ROD_DAQ}/RodCrate && make -C ${ROD_DAQ}/RodCrate inst`

Chapter 5

Calibration

The Calibration Console takes care of steering scans, select a configuration, defne the number of loops that need to be run and the actions that need to be taken at each loop. It defines the scan configuration and which functions are called at PixModuleGroup and PixController level.

5.1 Adding a scan to the Calibration Console

In order to add a scan to the Calibration Console, few files need to be changed and a new preset configuration needs to be added to the database. I'll use here FDAC_FAST_TUNE as example.

- Define the *prepare* and *end* methods in the PixModuleGroup
  - Declare the *prepareFDACFastTune* and *prepareFDACFastTune* in PixModuleGroup.h
  - Define the *prepareFDACFastTune* and *prepareFDACFastTune* in PixModuleGroup.cxx
- Add the *prepare* and *end* methods to the PixScan class modifying `PixController/PixScan.cxx`. The methods need to be places in the case/switch block and activated by new enums. In the case of FDAC_FAST_TUNE the *prepare* and *end* methods are placed under the `PixScan::FDAC_FAST_TUNING`. The only thing that matters for the new methods to be run is defining loop action, the scan type only links to a particular configuration setting.
- Modify PixLibInterface/PixEnumBase.h to add the new scan name to the EnumScanType ScanType enum and to the `m_scanTypeMap`
- In the case your new scan has a different loop action, like `FDAC_FAST_TUNING` in this particular case, this needs to be added to the class EnumEndLoopAction to the enum EndLoopAction and to the `m_EndLoopMap` in order for it to appear in the Console options.
- A preset configuration needs to be added to the `PixLibInterface/PixScanBase_preset.h` file in order to generate an entry to the database
- Finally, in order to let a new fresh scan be selectable in the Console, one has to add a preset configuration to the console database. This needs to be done only once.

- Compile the daq package tools with `make pixTools` from the daq folder. This will compile the `test_*` tools in the Console package.
- Run

```
export timestring=‘date +%s‘
Applications/CalibrationConsole/Console/test_createScanParamDb -r $timestring -t
Base -f PM_FE_I3 -s FDAC_FAST_TUNE
Applications/CalibrationConsole/Console/test_createScanParamDb -r $timestring -t
Base -f PM_FE_I4 -s FDAC_FAST_TUNE
```

5.2 General procedure

5.3 Tested scans, algorithms, configurations and caveats

In PixelModuleGroup, for FEI4, STEPS_8_8DC and STEPS_8_DC are treated the same.

5.3.1 IF_TUNE

The IF_TUNE performs a scan over the PrmpVbnf/IF register setting. The tune algorithm is:

-

5.3.2 IF_FAST_TUNE

The IF_FAST_TUNE performs a binary seach over the PrmpVbnf/IF register setting. The tune algorithm is the following:

- Set all the FDAC values in the middle of the range (4 for the FEI3, 7 for the FEI4).
- Makes the charge conversion from VCAL depending on the selected capacitor (capacitor box for FEI3, Mask type (SCAP,LCAP,BCAP) for FEI4).
- It can start from the values stored in the configuration or from the first value provided in the loop values.
- Checks the average ToT ($\langle ToT \rangle$) over the firing pixels on a FE (no firing pixels implies $\langle ToT \rangle = 0$.
- If $\langle ToT \rangle < tgt$ lower IF, else raise IF. IF is imposed to be greater than 0 and lower than 255 (These limits need to be fixed for FEI3).

Currently the scan is bugged. The end of the scan only takes care of filling the histograms. Logic has been briefly checked and should be fine from a first look.

Important Settings	
Tuning Starts from Cfg	Y
Restore Cfg After Tune	Y
Mask Stage	STEPS_8_8DC
Capacitor	FEI4_ENA_BCAP
Mask Step	1

5.3.3 GDAC_FINE_FAST_TUNE

The GDAC_FINE_FAST_TUNE holds only for IBL and performs a binary search over the `Vthin_Fine` register setting. The tune algorithm runs only on the active modules and is:

- If the start from Cfg values is not selected the tune sets all the `TDAC to 15`, while if it is selected it uses the `TDAC` values stored in the configuration. The `VCoarse_Fine` is always set to 128 at the start of the tune
-
-

The GDAC_FINE_FAST_TUNE doesn't create a full configuration, but only one valid for IBL. This makes sense as this scans only focuses on IBL tuning, however it is not possible to just clone the output configuration of this scan back and use it to configure the full detector. Usually such scan has to be followed by one that actually copies the full detector configuration, such as an `IF_TUNE` or `TDAC_FAST_TUNE`.

5.3.4 GDAC_TUNE

For the B-Layer, Barrel and Disks tuning we use the GDAC_TUNE algorithm.

- Fix the initial TDAC values to 64 or to the configuration values in the case the option *Tuning Starts from cfg values* is selected.
- The algorithm scans fixed values of the GDACs, calculate the occupancy and finally interpolates linearly between them in order to find the proper tuning value for the FEs
- Generally 6 points between 10 and 27 are sufficient to tune quite nicely

In Figure 5.1 the distribution of the tune of the B-Layer at 4300ke made in March 2018 is shown. One thing to check is the number of front ends that accumulate at the extremes of the tuning interval. In this case about 1% of the frontends accumulate at GDAC=10, indicating that they have not been tuned perfectly. When this happens, the TDAC tuning will compensate producing distributions that are not centered around 64 in those FEs. This effect can be seen in Figure 5.2, where a module with proper tuning of GDAC is shown next to a module with few FEs that did not get properly tuned.

It is suggested to not reduce the GDAC tuning range to a value much lower than 10. In fact, it has been observed that during the tuning of the GDAC some modules can have large spikes in the current as shown in Figure 5.3. The spikes have a time duration compatible with one scan step. The reason behind this effect is due to the fact that all the FEs of a module are set to a low threshold at the same time, causing a larger current in the module. This is not an optimal situation and should be improved in the

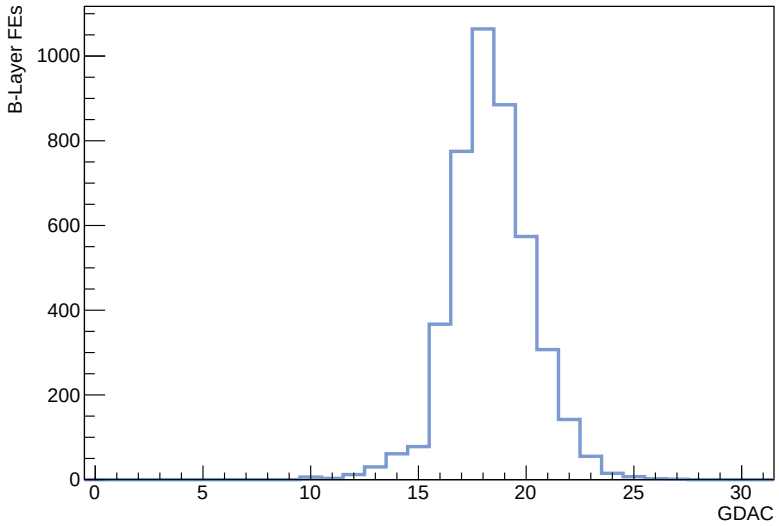


Figure 5.1: Distribution of tuned GDACs for B-Layer at 4300ke.

software. The second step in GDAC sets larger thresholds to the module FEs, therefore the current goes back to normal values. During scans after the GDAC tune, current spikes are not seen as the front ends are tuned to a proper threshold.

Finally the GDAC_TUNE has been observed to have about $100e^-$ of lower bias with respect to the tuning point. In order to have better distributions for the follow up TDAC tuning, it is suggested to tune to $100e^-$ above the tuning point.

5.3.5 TDAC_TUNE

5.3.6 TDAC_FAST_TUNE

5.3.7 FDAC_TUNE

5.3.8 FDAC_FAST_TUNE

The FDAC_FAST_TUNE implements a binary search on the FDAC tuning parameters. It has been tested only for IBL, not for Pixel yet. The scan is generally faster than the FDAC_TUNE and should give same results.

- ...

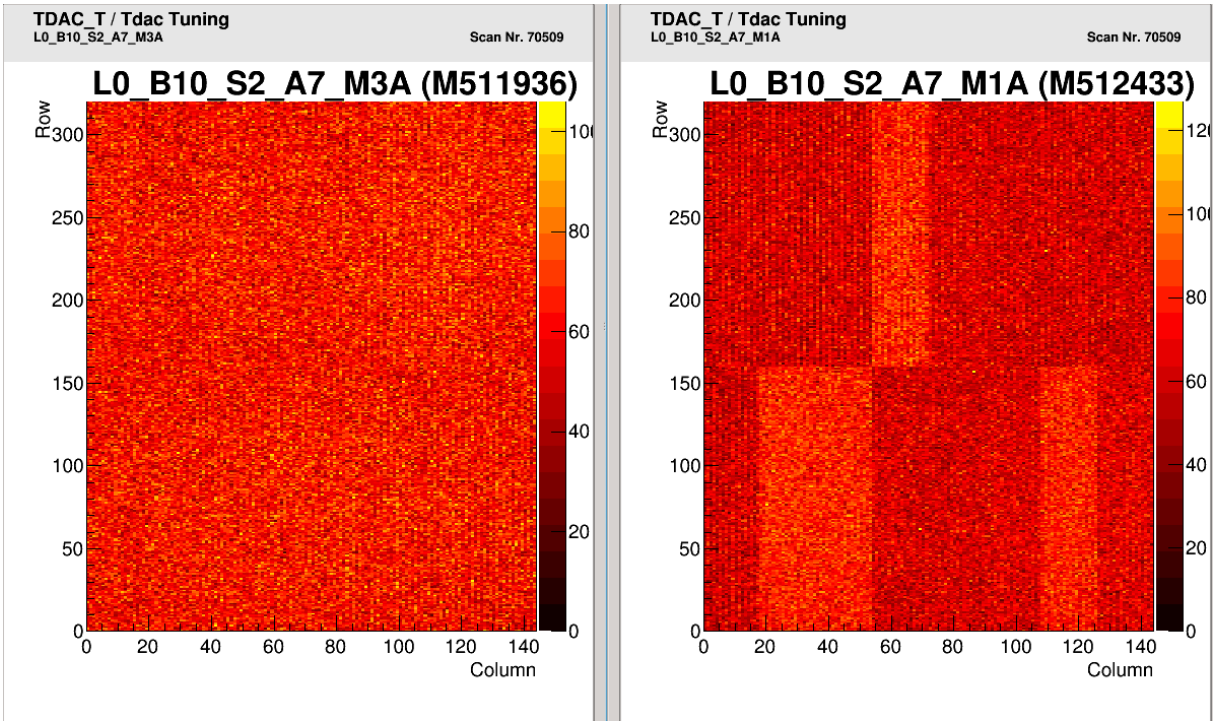


Figure 5.2: Left: module with GDAC properly tuned, where is not possible to distinguish FE dependend offsets. Right: module with few FEs with GDAC out of tuning. For those FEs the TDAC distributions have generally higher value in this case.

5.3.9 T0 SCAN

5.3.10 BOC_DEL_DSP_THR_DEL

5.4 Calibration procedure

Here are defined the tuning procedures and the configuration tags that need to be used in order to run basic calibration on Pixel and IBL. Before starting calibration is necessary to run monitoring scans in order to assess the status of the detector. After every tuning is necessary to run monitoring scans to assess the good tuning. The minimal things to check are the ToT and Threshold mean values for the modules. Then the Pixel Registers distributions and the PrmpVbf and Vthin_Fine distributions. Finally is fundamental to run a TOT-Charge calibration scan that needs to be provided to Pixel Offline group to update the database with the proper interval of validity.

FEI4 Tuning procedure The tuning procedure was defined during production and qualification of the IBL modules which is referenced in **Malte's thesis**

The suggested tuning algorithm is:

1. Global threshold adjustment (GDAC_FINE_FAST_TUNE)

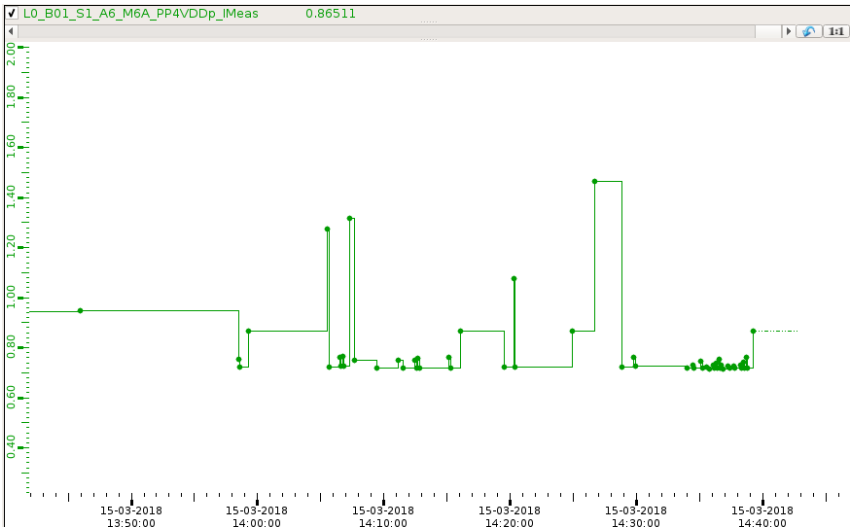


Figure 5.3: Module current during a GDAC tune. Spikes in current to the maximum limit of 1.45A can be seen at the beginning of the scan. They last for about 3 minutes which is the time of the GDAC scan step.

2. Global feedback current tuning (IF_TUNE)
3. Global threshold tuning (GDAC_FINE_FAST_TUNE)
4. Pixel threshold tuning (TDAC_FINE_FAST)
5. Pixel threshold feedback current adjustment (FDAC_TUNE)
6. Pixel level threshold re-tuning starting from the previous results (TDAC_FINE_FAST from config values)

This should be the minimal set of tuning to be performed during regular re-calibration of the detector during data taking. In the past years, especially 2016/2017, there was quite a bad habit to tune only Pixel level registers. This created a trend in the pixel register distributions and caused them to not be anymore centered around the mean value of the tuning range. In Fig.5.4 this effect is shown, together with the scans used to measure it.

FEI3 Tuning procedure

The tuning procedure for FEI3 can follow the same procedure of the FEI4. However, since the interplay between treshold and ToT is lower in FEI3, one can apply a faster procedure:

1. Global threshold adjustement (GDAC_TUNE)
2. Pixel threshold adjustement (TDAC_FAST_TUNE)
3. Global feedback current tuning (IF_FAST_TUNE)

DATE	THR	TOT	GDAC	IF	TDAC	FDAC	THR_F	TOT_F
10/11/2017	2459	8.34	/	/	68565	68566	2517	8.1
07/11/2017	/	/	/	/	68518/26/8	68519/27/9	/	/
24/10/2017	2465	8.29	/	/	68125/30	68126/31	2502	8.09
19/10/2017	2479	8.22	/	/	68002/30/5	68003/34/6	2501	8.08
10/10/2017	2478	8.23	/	/	67734	67735	2513	8.09
02/10/2017	2415	8.06	/	/	67682/6	67683/7	2500	8.05
19/09/2017	2502	8.5	67403	67406	67408	67407/9/19	2499	8.08/8.03
11/08/2017			/	/	66963	66964	2502	8.05
18/07/2017			66850	66852	66850/4	66853/5	66857	66856
05/07/2017	2409	8.3	/	/	66803	66804	2516	8.05
19/06/2017	2492	7.73	/	/	66568	66569	2468	8.01
02/06/2017	2558	7.95	/	/	66474	66475	2499	8.03
15/05/2017	2357	8.144	/	/	66334	66335	2500	8.104
24/04/2017	2340	7.369	/	/	65885	65889	2502	8.024

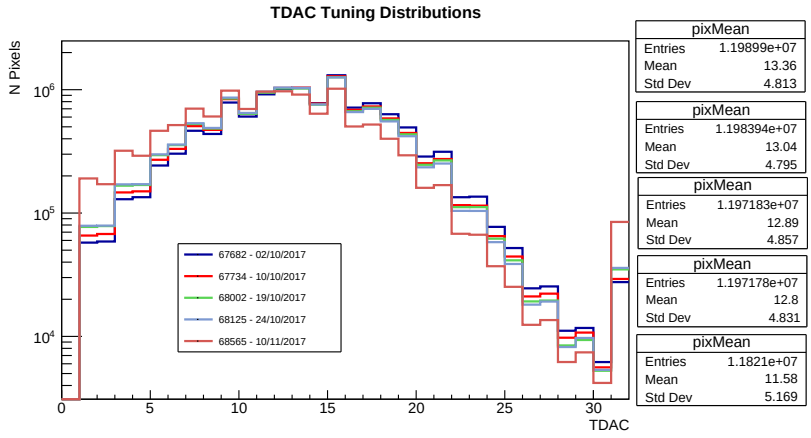


Figure 5.4: Observed TDAC drift in calibration when tuning only Pixel Registers in order to recover periodic detuning of IBL during data taking

4. Pixel threshold adjustement (FDAC_TUNE)

The tuning procedure is described in **Jörn's habilitation**.

5.5 Add features to the Console

A full reference to the QT classes can be found on QT website. We are using version QT4. It is probably possible to use QT designer to modify the layout and connections of the Console (just type `designer` on the terminal) however, personally, I found easier to modify directly the code for small modifications to change the layout and propagate the numbers to the scan configuration. The recipe to do that is outlined in the following steps:

- **Add new items to the Console layout:** navigate to *CalibrationConsole/Console/ui_PixScanPanelBase.h*. Here all various items are defined and you can add objects here to modify the appearance of the Console and of the various panels. At this stage the new features are not passed to a new scan configuration, but you only modified the appearance of the Console

- **Connect the new features to a PixScan Configuration**: open *CalibrationConsole/Console/PixScanPanel.cxx*, find the `m_knowHandles` object and add the following line:

`m_knownHandles.insert(std::make_pair(<name>,<feature>));`
Where *¡name¿* is formed by the *¡configuration name¿_¡variableName¿* such as *general_useAltModLinks*, while *¡feature¿* is the name of the object you just added to the `ui_PixScanPanelBase.h`.

- **Compile the Console**: the compilation needs to be done in `CalibrationConsole` folder. Just hit `make` there and it should pick the changes.

- **Add the configuration object to the Scan:** in order to link the handle to the scan configuration, just create a default object for the scan configuration. Go to `PixController/PixScan.cxx` and add the line
`config[<configname>].addInt(<variableName>,`
`<member name>,<default>,<info>,true);`
where the *¡configname¿* is the configuration type (*general*, *scans*,...), *addInt* is just for adding an integer variable in this example, *variable* is to add the variable to the configuration, *member name* is the name of the member of the PixScan class that needs to be added, *default* is a default value, *info* is a string of info about the variable.

- **Add the member and the set/get methods**: At this point navigate to *PixLibInterface/PixScanBase.h* and add the member to the Scan interface class, add the member and the set/get methods, following the other examples.

- **Modify the serialisation of the scan class:** it is fundamental to now add the new variable to the scan serialisation or you will have run time errors and actions crashes. Go to `$PPCpp/Scan/src/SerializablePixS` and add `prep(<member>)`

- Flash the PPCs

5.6 Dependence on the strobe delay settings

It is very important that are used corect strobe delay settings during the scan

5.7 Noise Mask

Two methods exist to obtain the noise mask. The collected data can be analyzed with external pixel RawDataAnalyxer package. A new method is to process the data as in calibration using histogrammer. These to methods are described in next sections.

5.7.1 Local data taking + RawDataAnalyzer

Calibration concludes taking noise data in local data taking mode in order to mask noisy pixels. There are two methods to obtain data from the detector usable for the noisy pixel masking: recording local data taking or using the histogrammer during local data taking. The noise mask is needed only if the modules are re-tuned and it's not necessary after an optical link tuning.

Recording local data taking
This method is based on the decoding of the recorded bytestream file from local data taking.

1. Clone the current module configuration tag to a test tag, for example `PIT_MOD_NM_TEST`.

2. Enable all pixels in the test configuration tag using
   `$PIX_LIB/Examples/ChangeCfg --enaPix --idTag PIXEL18 --connTag PIT-CONN --cfgTag PIT_BOC --cfgModTag PIT_MOD_NM_TEST --pendTag EnableAllPixels --save`
   One can use `--partitionName <PARTITION>` in order to select only Barrel, Disks or IBL. Check `ChangeCfg.cxx` for that.

3. Publish the tag using `TagManager`, check the *Override stable beams* box in order to turn the preamplifiers on during the run and restart the actions. Select the appropriate runConfig, crosschecking with Pixel Run Coordinator.

4. Run local data taking with recording activated:

   - Set HLT frequency to about 3kHz. To do so rescale the `l2Acceptance` variable to 0.05 in `TDAQ-IBL.data.xml`
     `/Applications/PixRCD/PixRCD-00-01-00/PixRCDConfiguration/PIT/segments/TDAQ-IBL.data.xml`
     Since we run at 60kHz L1 for IBL, a factor of 0.05 will put 3kHz the HLT rate. Commit and reload.
   - Start the ibl daq slice `start_daqslice_ibl`, click on **Settings** panel in the *Run Information & Settings* tab and:
     - Set the Max Events to the desired number of L1 triggerd events (3M takes about 18min of data taking)
     - Tier0 Project Name can be data_test
     - Recording to Enable
     - Click on SetValues
     Then commit and reload.
   - Start data taking. It will stop automatically when the number of recorded events is reached.
   - Data will be saved on the /det/pix/data/localDataTaking/ with the runNumber that was displayed on the daq slice

5. Login to a server machine `ssh -Y pc-pix-srv-02` and setup the **RawDataAnalysis package**.
   `cd /det/pix/data/RawDataAnalysis_newVersion`
   `source setupRaw.sh`

6. Run the decoder on the recorded data. `scriptMakeNoiseMask.sh` is for pixel, `scriptMakeNoiseMaskIBL.sh` is for IBL:
   `./scriptMakeNoiseMaskIBL.sh /det/pix/data/localDataTaking/ <projectName> <runNumber>`
   it will create about 10 parallel jobs that will decode the data. When done merge the single files and transfer them to lxplus[1]. The file can be several GB large, depending on the amount of data you recorded: better to store it in eos.

7. On lxplus it is possible to use RawDataAnalysis package to produce the noise mask on the batch:

   - In the main directory of RawDataAnalysis, do
     `source ./setup.sh`

[1] I usually setup an *scp* tunnel in order to copy directly to lxplus without passing from atlasgw. To do that I open a new terminal (or xterm), and `ssh -L 1234:lxplus:22 <usernam>@atlasgw-exp` , then it's possible to copy files using `scp -P 1234 <file> <username>@127.0.0.1:<path>` [scp tunnel reference]

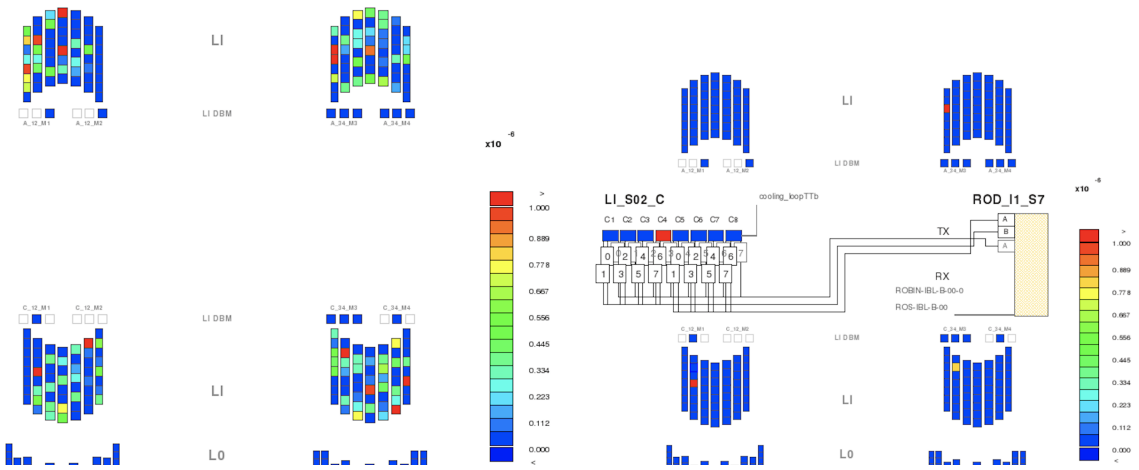


Figure 5.5: Formatter occupancy of IBL taken from RodMon before (left) and after (right) noise masking to the level of 1e-7. Some modules are still noisy due to faulty double columns that have been disabled in a second stage.

- Run the mask computation on the batch:

Once the occupancy map is computed, one has to create the noise mask according to a certain threshold. To do that, there is the macro located in `/det/pix/daq/NoiseMasks/`. Usage:
`CreateNoiseMask <occFile> <maskFile> [threshold] > log_maskFile>.log`
If the threshold is not specified, 10^{-6} is taken. Usually we need to aim to 10^{-7}, but time doesn't always permits it. There are a set of pixels that are hardcoded in the disable. Those pixels, or particular DCs, have been found to be faulty during the years. The macro spits out a root file with histograms with the disable masks. In order to apply the mask one simply runs:
`$PIX_LIB/Examples/ChangeCfg --disPix <noiseMask.root> --idTag <IDTAG> --connTag <CONNTAG> ---cfgTag <PITBOC> --cfgModTag <CFG_TAG> --pendTag <PendTag> --save`

In Figure 5.5 the effect of applying the noise mask in IBL is shown. It is possible to see the occupancy reducing noticeably, apart from 3 modules that have faulty double columns.

During M2 with cosmics in 2018, has been found out that this procedure of noise masking depends on data acquision conditions such as trigger and readout window. In Figure 5.6 the formatter occupancy for three local data taking configurations is shown. In the plots the data around 13:20h are taken with 1BC readout window, while data around 13:40 is taken at 5 BCs readout window. It can be seen that some modules get noisy only when the readout window is expanded. Therefore the data to mask noise needs to be collected with same run settings used during operations in ATLAS. Finally, the effect of masking noisy pixel is seen in the ToT distribution. In Figure 5.7 spikes on the ToT distribution due to noisy pixels together with correct distribution after noise masking are shown.

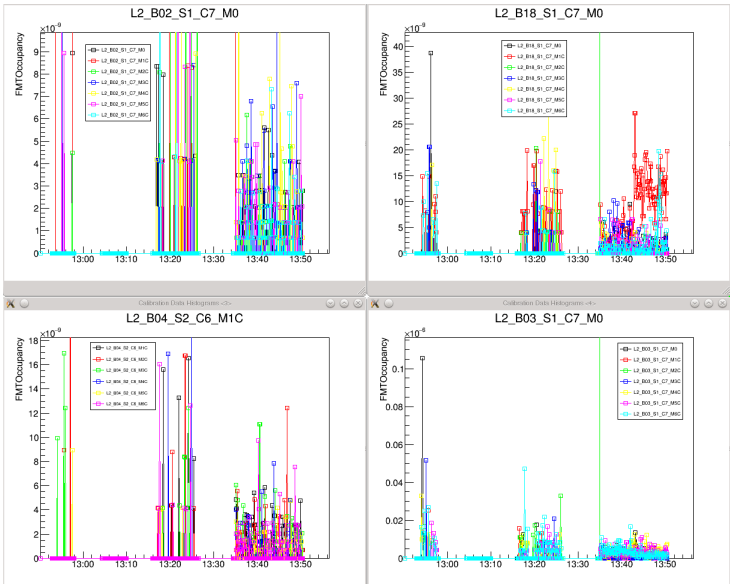


Figure 5.6: The formatter occupancy history for 4 different PP0s on Layer2. It can be seen that

5.7.2 Local data taking + Histogrammer

Use histogrammer during local data taking

This method is based on using the histogrammer during local data taking to monitor the data on the fly.

1. Clone the current module configuration tag to a test tag, for example `PIT_MOD_NM_TEST`.
2. Enable all pixels in the test configuration tag using
   `$PIX_LIB/Examples/ChangeCfg --enaPix --idTag PIXEL18 --connTag PIT-CONN --cfgTag PIT_BOC --cfgModTag PIT_MOD_NM_TEST --pendTag EnableAllPixels --save`
   One can use `--partitionName <PARTITION>` in order to select only Barrel, Disks or IBL. Check `ChangeCfg.cxx` for that.
3. Publish the tag using `TagManager`, check the *Override stable beams* box in order to turn the preamplifiers on during the run and restart the actions. Select the appropriate runConfig, crosschecking with Pixel Run Coordinator.
4. Publish in IS that the histogrammer has run during data taking, using
   `is_write -p partitionName -n Histogramming.useHistogrammer -t Boolean -a value -v 1`
5. Run local data taking:
   - Start the ibl daq slice `start_daqslice_ibl`, click on **Settings** panel in the *Run Information & Settings* tab and:
     - Set the Max Events to 0
     - Recording to Disable
     - Click on SetValues
     Then commit and reload.

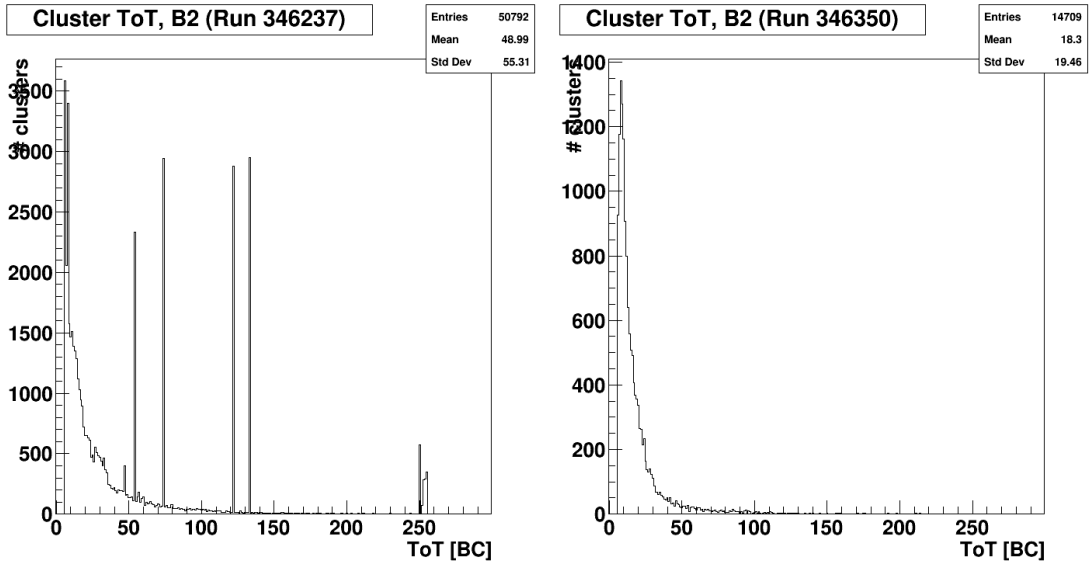


Figure 5.7: ToT distribution during two cosmic runs. Left: before masking problematic noisy pixels at 5BCs, but after masking at 1BC readout window. Right: after masking noisy pixels at 5BCs

- Start data taking. It will not stop automatically. It has to be stopped when wanted number of events is reached.
- Data will be avaiable in OH.

6. Disable useHistogrammer in IS, using
   `is_write -p partitionName -n Histogramming.useHistogrammer -t Boolean -a value -v 0`

7. Use **nmth_processing.py** python script in *daq/Applications/Scripts/nmth* for downloading and processing the data from OH

5.8 Some details about the tag database

5.9 PPC Code and Usage

The PPC software for the IBL readout is located in `/RodDaq/IblDaq/RodMaster/Software/PPCpp`. The code is organised in different folders:

- **CmdFactory**: Contains the declarations and definitions of the various tcp/ip host-ppc commands also called host command pastern.
- **CmdPattern**: Contains the infrastructure for the tcp/ip communication between host and ppc.
- **FrontEnd**: Contains the classes that define and communicate with the sensors.

- **QuickStatus**: The code for the QuickStatus thread
- **Scan**: Package for running scans
- **SysConnect**: BOC-ROD specific classes
- **SysStatus**: Thread that monitors the PPC kernel ticks and status
- **Utilities**: Utilities classes
- **inc/src**: Source code of the PPC server application

The doxygen will be available (TO-DO)

5.9.1 Loading the code to the ROD

Once the developer is happy with his code implementations, the PPC code is compiled issuing `make` in the the PPCpp folder. Once the code is compiled, it needs to be loaded in the ROD in order to run the PPC with the software that has been just generated. There are two ways to do that, via the Xilinx cable or via the VME bus.

Using the Xilinx Cable

In SR1 there is one xilinx cable attached to each ROD. In Tab. 5.1 the Xilinx cables connected to the Host and ROD are shown. The script to load the FW and SW through Xilinx cable is located in `daq/scripts`. It has various operating modes, that are quite self-explanatory reading the script, but the main ones are the following:

- `iblRodCtrl <RODNAME> start cpp` will flash the software to the PPC. The PPC code needs to be compiled and the .elf needs to be located in the right location (if not a self-explanatory error message is shown). Then it starts the PPC server.
- `iblRodCtrl <RODNAME> start all` will flash both the PPC and the histogrammer software. Again, both PPC and Slaves need to be compiled otherwise an error message will be shown.
- `iblRodCtrl <RODNAME> flash master|slaves|all` will load the Firmware in the master or in the slaves or in all the Fpgas respectively.

Using the VME bus

It is not always possible to flash the PPC via Xilinx cable. For example in the PIT there are only a small number of Xilinx cables available and it is only possible to flash via Vme. This is the script that needs to be used:
`/det/pix/daq/binaries/IBLROD/current/iblRodCtrlVme RODNAME --flashSw --flashFw --pixFw|iblFw (--run)`
Where `<RODNAME>` is the ROD name in the following format ROD_YY_SX where Y stands for the crate name and X for the slot number, alternatively you can flash entire crate providing the crate name (this option is only available in the PIT). On the other hand, –flashFw is used to flash the firmware, while you do it you should specify the FE flavour for the ROD either Pixel or IBL `--pixFw|iblFw`, –flashSw is used to copy the software to the flash memory that should be linked to your local repository. Run this command without the `--run` option to make a dry run that only displays what the command is going to

ROD NAME	CABLE ESN	Host Machine Name
ROD_C1_S7	000017DEEAA201	control1
ROD_C2_S16	0000154A2A4D01	control6
ROD_C3_S8	0000154AEA3701	control8
ROD_C3_S11	0000128DB56C01	control7
ROD_C3_S15	0000174C85E501	control7
ROD_C3_S18	0000150F4D9801	control8
ROD_L1_S9	0000150F340701	pc-pix-fit-01
ROD_I1_S7	0000153A7E5201	pc-pix-fit-02

Table 5.1: Xilinx cables ESN numbers associated to which Rod and which Host. Crate2 are not actually used in SR1. I added them here for completeness to list cables that are not used in other places. The last two entries are valid for the PIT only. The reference of this table is the `iblRodCtrl` script updated at 18 Dic 2017. The best reference is the script itself.

do, while use it in order to run the actual command. Further options are available –loadPROM to load a special FW in the PROM and –restartPROM to reset the ROD while loading the default FW in the PROM (similar to a power-cycle of the crate).

ToDo: This command should be mentioned somewhere:

```
RESET_PPC_SW=1 \~pixeldaq/daq/Applications/PixLib/Examples/iblRodFwReset ROD_C1_S7
```

5.9.2 Debugging the PPC code in Xilinx

It is possible to run gdb on the PPC code for some sort of debugging. Some notes are available on the xilinx documentation support (where?). There might be a faster way to do so in our scripts, but this information got lost.

1. Compile the PPC with DEBUG flag
   `make DEBUG=1`
   The PPC code will then be placed in `$PPCpp/debug/bin/` instead of `$PPCpp/bin/` folder, so one needs to update the loading scripts or link the newly compiled file.

2. Open two shells on the machine that has the JTAG cable connected to the slot you want to investigate[2]. These two shells will be called shell1 and shell2, from now on.

3. In shell1 connect to the ROD using the script:
   `~karolos/scripts/run_xilinx xmd`
   As a result an XMD shell should open (the commands that are issued inside this shell are labelled by XMD¿). One needs to connect to the PPC using the following command
   `XMD> connect ppc hw -cable type xilinx_platformusb esn <JTAG cable number> -debugdevice devicenr 8`

[2] At the time of writing this notes, if one wanted to run the debugger on C3_S8 he needed to connect to control8

```
XMD% connect ppc hw -cable type xilinx_platformusb esn 0000154AEA3701 -debugdevice devicenr 8

JTAG chain configuration
--------------------------------------------------
Device   ID Code        IR Length    Part Name
 1       d5059093          16        XCF32P
 2       d5057093          16        XCF08P
 3       4401d093           6        XC6SLX150
 4       d5059093          16        XCF32P
 5       d5057093          16        XCF08P
 6       4401d093           6        XC6SLX150
 7       d5059093          16        XCF32P
 8       632c6093          10        XC5VFX70T

PowerPC440 Processor Configuration
-----------------------------------
Version..............................0x7ff21912
User ID..............................0x00f00002
No of PC Breakpoints.................4
No of Addr/Data Watchpoints.........2
User Defined Address Map to access Special PowerPC Features using XMD:
        I-Cache (Data)........0x70000000 - 0x70007fff
        I-Cache (TAG).........0x70008000 - 0x7000ffff
        D-Cache (Data)........0x78000000 - 0x78007fff
        D-Cache (TAG).........0x78008000 - 0x7800ffff
        DCR...................0x78020000 - 0x78020fff
        TLB...................0x70020000 - 0x70023fff

Connected to "ppc" target. id = 32
Starting GDB server for "ppc" target (id = 32) at TCP port no 1234
```

Figure 5.8: Screenshot of the output of the connection to the PPC using the XMD commands. In red is circled the gdb port in this case. The port can change, so every time take a look to this number.

where the JTAG cable number can be read from the `daq/scripts/iblRodCtrl` script in the daq installation.

4. Once you connected to the PPC it should tell you a certain set of infos, including the port for the gdb. An example is shown in Fig. 5.8.

5. It is now possible to download the software to the PPC

   `XMD> dow <path-to-ppc-folder>/debug/bin/PPCpp.elf`

   Feel free to create a symbolic link to shorten this command.

6.

5.9.3 Monitor the PPC application

5.9.4 Communication between Host and PPC

5.9.5

5.10 Basic BOC commands

In SR1 the BOC commands can be issued from the `pixeldaq` account. In the home area of `pixeldaq` there is the `Boc2-BocFirmware` folder containing the setup and the commands. The first step is to run the setup:

```
source Boc2-BocFirmwware/Boc2/setup.sh
```

Useful commands are located in

```
/home/pixeldaq/Boc2-BocFirmware/Boc2/bin/common
```

In general, the location of the Boc2 folder in the pixel-daq repository is in `/daq/RodDaq/IblUtils/Boc2`.

Here there is a list of basic commands that an user might need to know. For the commands I've tested the `-h,--help` arguments print the help and exit. You are encouraged to use it to crosscheck the usage of each command. The ones marked in red indicate expert commands and would be good to crosscheck with an expert before using them.

- `DumpRegister`: Dump the BOC registers. Very useful to crosscheck the Rx frames in the case of calibration/data taking problems[3] and to check if there optoTune problems (for example if the Delay/gain are at default values)
- `LaserOn/LaserOff`: brings the lasers to On/Off. Very important during Boc Recovery in order to avoid to send crap to the modules that might ruin the hardware.
- `SlinkEnableFTK`: enables the Slink. Important after BOC recovery or no data will be sent through the Slink.
- `ShowVersion`: prints the hash of the BOC FW and if the FW is for Pixel or IBL.
- `CheckDelayP1/SetDelayP1`: check and sets the Tx delays in the Pit. The configuration files are stored in
  `/det/pix/daq/binaries/IBLBOC/DelayConfigs`
  Check the file `current-delays.txt`, this points to the txt file where the delays are saved.
  Usage: `CheckDelayP1 -f <DelayFileTxt>`
  Usage: `SetDelayP1 -f <DelayFileTxt>`
- `ResetBcf/ResetBmf`: it resets the Bcf/Bmf. When there is a power outage (no power cycle) and ROD goes busy during local data taking, this might be useful to recover the ROD.
- `FlashBcf/FlashBmf`: it flashes another FW in the Bcf/Bmf. When there is a power outage (no power cycle) and ROD goes busy during local data taking, this might be useful to recover the ROD, if resetting the Bcf/Bmf didn't work.
- `GetUptime`: tells how much the BOC is up.
- `IblBocMon`: opens the boc monitoring console.
  Usage: `IblBocMon --ip <ip>`, then follow the options
- `IblOptoScan`: runs the opto scan for IBL.
  Usage: `IblOptoScan -l 200 -s 1 --target <north|south> --ip <ip>`

The other commands are usually not needed by a general user.

[3] At the end of calibration the Rx's are disabled which means that if you dump the BOC registers at the end of the scan you'll see the Rx's Frames at 0. You need to first comment out the line in the code that disables the Rx's at the end of calibration in the PPC code and flash again.

SR1	/tdaqfiles/nightly_builds/
PIT	/detpriv/pix/

Table 5.2: Top folder location of the BOC firmwares.

5.11 BOC Firmware location

The BOC Firmware is stored in the following directories:

In order to locate the proper file, the best is to dump the FW hash first, then use the `byHash` folder to retrieve it.

5.12 Boc Recovery - Resetting the BCF and the BMF

After PPC crashes might happen that some of the registers on the BCF and the BMF can get overwritten. There is an internal protection of some of the Tx registers but the Rx registers are not protected. In fact, in the official firmware, only few bits in the TX control register are protected. These bits allow switching off the biphase-mark encoding or switching the TX to 160 Mbit/s mode (which is used for a direct loopback test). Both modes are masked out if the register is protected. For future firmware versions the coarse-delay setting to the protection will be added to the protection.

However the Rx corrupton, can cause several issues, both during data taking and calibration. For example in calibration has been observed that

This can cause scans to proceed OK but return empty histograms. Here are the description of the steps to take in order to check and fix this potential problem. In SR1, I usually run the commands in pixeldaq account

1. Check the BOC registers
   ```
   source Boc2-BocFirmware/setup.sh
   DumpRegisters --ip <BocIP>
   ```
   [4]

2. Check if the BOC registers are corrupted by comparing the DumpRegisters from the BOC with a working BOC. Typically just a BMF reset is needed. A soft reset has been implemented in recent BMF FW versions. This operation doesn't change the status of the lasers unless the BOC is running a old FW version.

3. Soft reset the BMF
   ```
   ResetBmf --ip <BocIP> --target <north|south>
   ```

4. However, from time to time a BCF reset is needed, most likely in the case that the BOC clock is somehow corrupted. This operation sets the lasers to off, so one needs to use particular attention when doing this[5].

[4] In SR1 the BOC Ip's are defined as 192.168.10+crateN.100+slotN

[5] Never proceed to reset the BCF without consulting first the experts around. If in the PIT, never do that without consulting with Pixel Run Coordinator first

5. VVDC should be off when resetting the BCF (also it is safe if the modules are swithed off). If the lasers go off while VVDC is OFF, weird configuration can be sent to the modules. In the PIT one should only do `VVDC_EXPERT_COMMAND_SWITCH_OFF`. This command has to be sent only when the modules are `ON`.

6. Reset the BCF (BMF is also reset)
   `ResetBcf --ip <BocIP>`
   When issued if you want to Boot in Recovery Mode say no. After reseting BCF the laser will go down.

7. Hard reset BMF (lasers go OFF))
   `ResetBmf --ip <BocIP> --target <north|south> --hard 1`

8. Ping the BOC to know when the BCF is up again. Then, bring the lasers on
   `LaserOn --ip <BocIP>`

9. After the reset of the BCF, the ROD loses the clock. Load the rod FW again
   `/det/pix/daq/binaries/IBLROD/current/iblRodCtrlVme <RODNAME> --flashFw --pixFw|iblFw --run`

10. Check that the ROD is up, Lasers are On (Checking the VPin_IMeas value at about 2mA. When laser are off VPin_IMeas is about 0.03mA). If all OK, switch ON VVDC.

5.13 Flash Boc Firmware

To flash the BMF, first make sure that VVDC is OFF! or the modules are switched OFF. source Boc2 ShowVersion –ip ¡ip¿ FlashBmf –ip ¡ip¿ –target north -f bmf_north_revD_ibl.ace FlashBcf –ip ¡ip¿ -f FILE *Verification success* The Filename needs to have "upgrade" in it.

In the PIT the Tx delays are embedded in the Firmware so be sure you match the right BOC firmware matching the filename to the ip. After flashing make sure that Lasers are ON, then switch VVDC ON.

5.14 IBL Opto Tune

The full IBL opto tune needs to be done when there is a power cycle of the IBL crate since the Rx delays are lost and the opto-tune needs to be performed again. There is a script to do that in parallel for the whole I1 crate:
`j̃garciap/scripts/iblOptoScanParallel I1`
This takes also care

11. Load the Opto-Tune

5.15 PPC code

5.15.1 PPC commands

The communication between the host and PPC is done through ethernet via TCP/IP protocol[6]. There is a serie of commands that can be sent to the PPC to make it perform internal actions, for example starting a scan or setting up data taking configuration. A command class is structured in the following form:

- **Command**: this is the body of the commands with the members variables that are set by the host. The command is then processed by the PPC that runs the `execute()` method.
- **Result**: a command might return back a result to the host. The result class is specific for each command and contains the member variables that need to be send back to the host

5.15.2 List of useful PPC commands, usage and expected output

Here is reported a list of general and most used PPC commands. For some of the commands that give back a result at the host a brief description of the output is given, together with an example of the expected output and an usage example. Note: the commands here are listed using the ip, but `--rodName <RodName>` option can be used and `<ip>` indicates the PPC ip.

- `setUart`: changes the uart output when reading via Xilinx USB cable. PPC=0, Slave0=1, Slave1=2
  `--setUart --ip <ip> --uart <0=PPC, 1=Slave0, 2=Slave1>`
- `getVersionInfo`: returns the hashes of the SW/FW for the ppc and slaves:
  `--getVersionInfo --ip <ip>`
- `dumpRodRegisters`
- `dumpRodBusyCounters`

5.15.3 Usage of the serial port

In order to use the Serial port this code snippet might be useful to initialise it (`IblScanL12.cxx`)

```
   SerialPort* sPort =NULL;
sPort = RodResourceManagerPIX::getSerialPort();
uint32_t txMask = BarrelConnectivity::getTxMaskFromRxMask(m_channels);
sPort->setTxMask(txMask);
Fei3Mod Fei3;
Fei3.setTxLink(sPort);

m_iblBocLink->enableTxChannels(txMask);
```

Vipin is for the Next to it there is the current of the lasers: if 0 then the lasers are off ViSet is the reference voltage of the VCSEL in the Optoboard Next to the ViSet current of th VVDC is kind of a supply voltage. It goes to 0 if we power cycle the crate. When we want to be safe: we need to switch off the VVDC. The voltage is in the OptoBoard.

[6] In principle UDP protocol can be used and few monitoring commands can use this protocol as well

5.16 VME Commands

There is a set of commands that can be used to access ROD counters, UART and other registers that are located in `$PIXELDAQ_ROOT/RodDaq/IblUtils/Vme`. The commands need to be issued from an SBC in order to work. Here is the list of some of the most used commands.

-
-
-

5.17 Packages

This sections provides brief description and documentation of rarely used or autmatic tools.

5.17.1 QuickStatus Web

Analyzes logs from the PIT and obtains run summary (LB, time etc.) by runquery. The latter runs under user okepka on lxplus (should be moved under pixeldaq)

- Web page: LINK
- Crontab jobs analyzing logs and producing a webpage run on lxpix1
- Location of the webpage: /data/tdaqfiles/QuickStatus/

Appendices

Appendix A

VNC

This appendix explains how to use a VNC server on lxplus to keep windows in SR1 and the PIT open which speeds up work logging back in. (this section was adopted from https://twiki.cern.ch/twiki/bin/view/Sandbox/JosephHaleySandbox)

Deviating from the instructions below it is recommended to try things in this order:

1. start a vncserver on lxplus.
2. From within the GPN (CERN network) open a VNC screen directly on lxplus.
3. From outside the GPN (eg. at home) open a VNC screen through an ssh tunnel.
4. Run the vncserver in a screen session with Kerberos token renewal.

1. ssh -Y lxplus.cern.ch
2. Note down the lxplus machine
3. Open a kerberos-renewal-screen: `pgrep -fl krenew -u ${USER} || AKLOG=/usr/bin/aklog krenew -b -t -- scre`
4. I actually use an alias in my .bashrc

   ```
   alias kscreen="pgrep -fl krenew -u ${USER} || AKLOG=/usr/bin/aklog krenew -b -t -- screen -D -m ; s
   ```

5. Open a vnc server as explained in the instructions below
6. Close the screen with ctrl+a crtl+d. (Don't exit the screen!)
7. Close the lxplus connection
8. In local you may want to create an lxplus ssh tunnel as explained in the instructions. I do that everytime I want to connect to the Remote Desktop

A.1 Starting a VNC server on an lxplus node

Log in to an lxplus node, but keep track of which one you are actually sent to (e.g. `lxplus0045.cern.ch`), you will need to connect this specific lxplus machine to use VNC.

```
ssh -Y <USER>@lxplus.cern.ch
```

Set up a screen session and set up kerberos and afs tickets so that the VNC server you start will keep running after you disconnect. (For more information on screen, see the manual pages or google.)

```
screen
k5reauth -x -f pagsh
aklog
bash
```

Start the VNC server with the geometry you want and the display number you want (e.g., :2). If display :2 is already in use, you will have to pick another number. It doesn't matter which display number you choose, but you must remember which one you used in order to connect later on. (I find the geometry 1880x970 convenient for 1920x1080 screens because you can see the whole desktop without going to fullscreen mode.).

If this is the first time you have started a VNC server: You will need to choose a password. It can be whatever you want, it doesn't have to be the same as your CERN nice password, but it should not be too weak. By default, the VNC server will start a really basic window manager (twm). You want it to start a more useful window manager like KDE (you could also start gnome, but I use KDE), so do the following:

Stop the VNC sever you just started:

```
vncserver -kill :2
```

Open the file `~/.vnc/xstartup` and comment out the line with `twm &` and add the line `startkde &` , so it looks like:

```
#twm &
startkde &
```

Now start the VNC server again:

```
vncserver -geometry 1880x970 :2
```

Now detach the screen session with: CTRL+a CTRL+d.

This will keep the screen session running in the background so the VNC server will not die when you log out of lupus. If you did this right, you will go back to the terminal you were in before you started screen and it will say [detached]. Make sure you remember the node you are on and the VNC display number you are using and log out of lxplus. You can now close this terminal.

A.2 Connecting to a VNC server

Make an ssh tunnel from the VNC port on your local computer to the VNC server you started on lxplus. The lxplus node is the one you noted down above and the port number is 590X, where X is the display number from above. If you started the VNC server on `lxplus0074` with display number :3, then you make the tunnel with:

```
ssh -L 5903:localhost:5903 -N -f -l <USER> lxplus074.cern.ch
```

Now start a VNC client on your local computer and connect to `localhost:5902`. On MacOSX the VNC client is built-in and you can simply do

```
open vnc://lxplus074.cern.ch:5903
```

or if you set up a ssh tunnel

```
open vnc://localhost:5903
```

Or you can use the system browser:

1. Go to system preferences - Sharing and select Screen Sharing
2. Setup a password (if you want)
3. To connect to the lxplus machine you can use Safari and put the url mentioned above in the url box

If you don't have a VNC client, you will need to install one. VNC Viewer, TightVNC and Vinagre are popular options. You will need to enter your VNC password and then a window should come up with the VNC desktop that is running on lxplus. Enjoy! (Or see below for troubleshooting.)

A.3 Troubleshooting

If you are not able to connect, it could be that the afs permissions have expired in the screen session where the VNC server is running, so it cannot read and write to you directory. You should be able to fix this with the following:

Log in to the lxplus node that is running your VNC server. For example, if you started it on lxplus0045, then:

`ssh ${USER}@lxplus0045.cern.ch`

Connect to the screen session in which you are running the VNC server:

`screen -x`

You should now be back in the screen session where you started the VNC server. If you do an ls you will see that your afs permissions are now gone. To get them back:

`k5reauth -x -f pagsh`

`aklog`

Now everything should work again.

Detach the screen session with: CTRL+a CTRL+d

You can now exit out of this window.

It could also be that the VNC server was killed and no longer exists. Although VNC servers should be able to run forever, it seems that sometimes they get killed on lxplus, maybe for maintenance purposes. If you go to the node that should be running your VNC server and it's not running, you will just need to start a new one. It could be that the VNC server is still running, but the screen session it was started in is not. I'm not sure how this happens, but it does and I don't know how to fix it. The only thing I know to do is kill the VNC server and start a new one in a new screen session. (If anyone knows a better way, please let me know.)