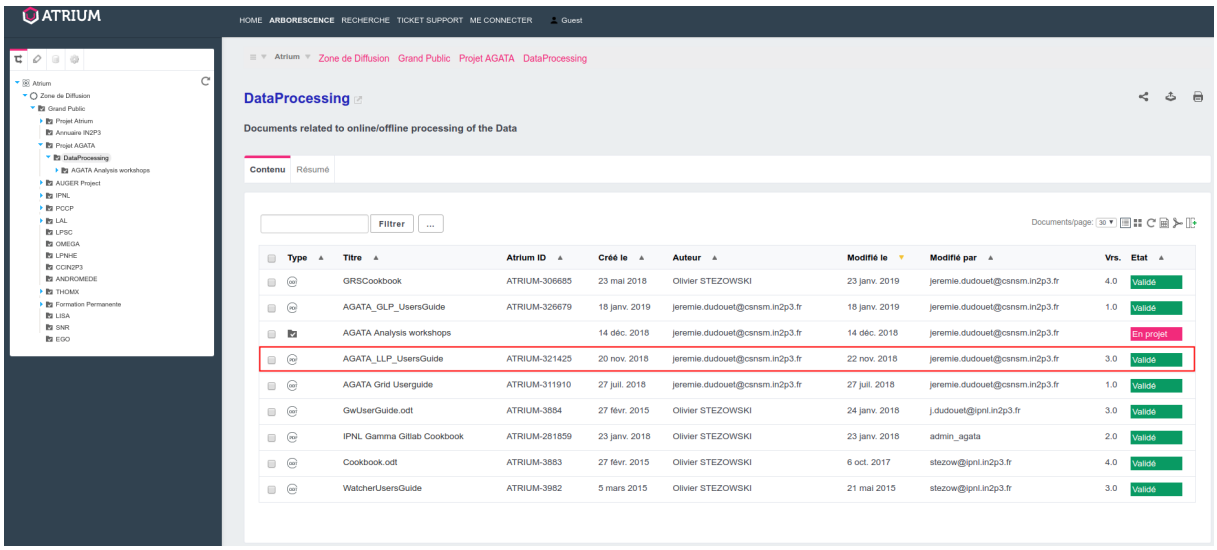
AGATA Data Analysis User's guide for Local Level Processing

AGATA Data Analysis Team

September 20, 2019

This document provides a guide to help the users analyzing the AGATA data produced at the local level processing i.e. before any building of events. It includes energy calibrations, time alignments, cross talk corrections and any other corrections to improve the quality of the data. Criteria for bad events rejection are also highlighted.

The last version of this document, provided by the AGATA Data Analysis Team, can be found on (ATRIUM).



Contents

People involved in this document:

D. Bazzacco, A. Boston, E. Clement, N. Dosmes, J. Dudouet, A. Gadea, F. Holloway, L. Hongjie, A. Korichi, N. Lalovic, E. Legay, J. Ljungvall, C. Michelagnoli, R. Perez, D. Ralet, M. Siciliano and O. Stezowski

NOTE: Please for any comment/question/suggestion contact agata{at}ipnl.in2p3.fr

1 Some general statements

1.1 AGATA Forum

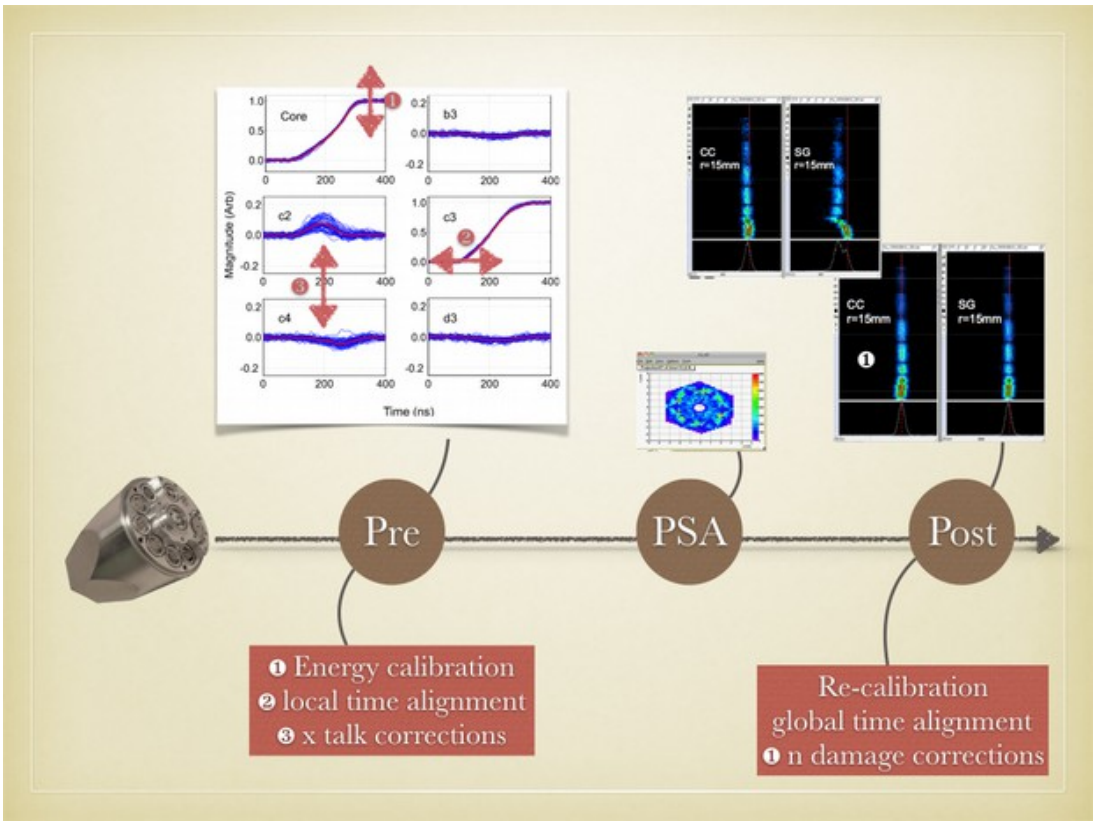
The AGATA Forum http://agata.in2p3.fr/forum is a dedicated place for any question, issue, or discussion concerning the AGATA data analysis.

1.2 NARVAL, actors and the AGAPRO package

The AGATA DAQ box is based on NARVAL [Ref, see also the section **'the ADF component' in the GammaWare User's Guide**]. As a consequence of such a framework, the AGATA data is processed through consecutive calls of ACTORS, each actor being in charge to delivering useful data for the next one in the chain. Even if NARVAL is itself written in ADA, it can bind other languages (C, C++, ...). However, the choice of the AGATA collaboration has been to develop algorithms in the C++ language. The most relevant actors are part of a package names AGAPRO [see **How to install the AGAPRO package in the Cookbook**].

1.3 The AGATA local level Processing

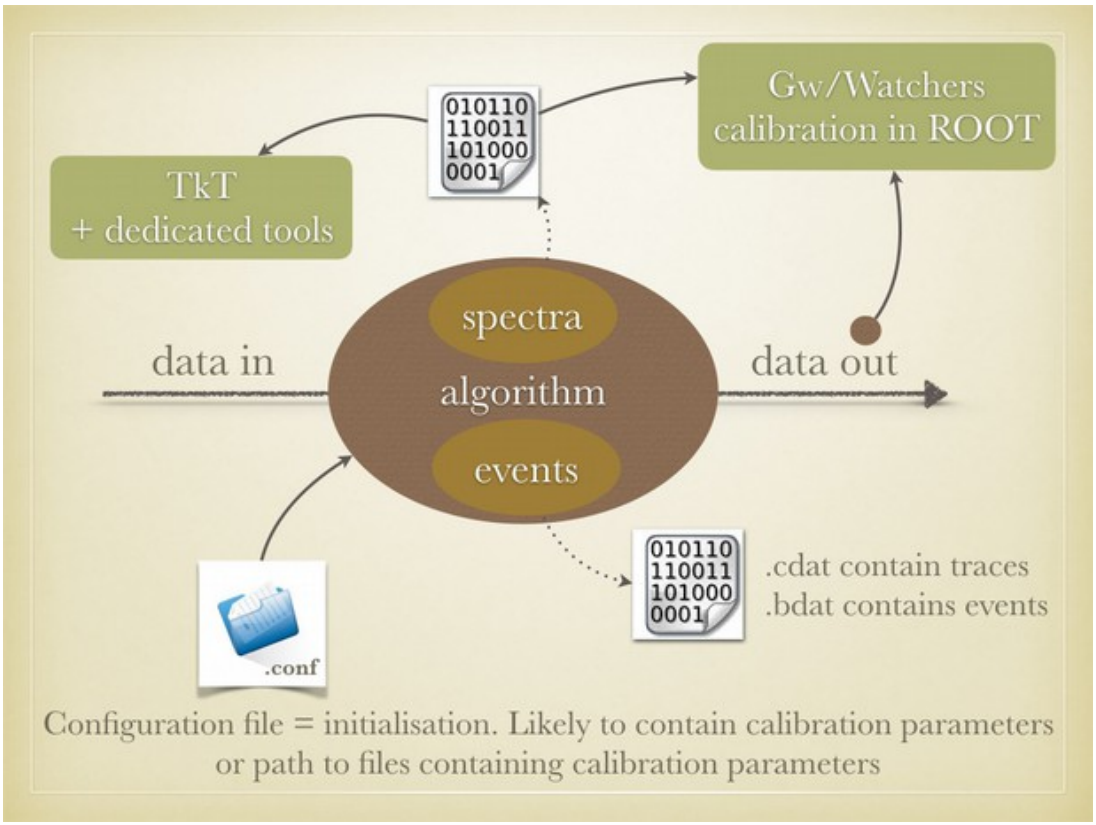
At the local level processing, three actors, also called filters in NARVAL's terminology, process the data flow. Those actors, Preprocessing, PSA (Pulse Shape Analysis) and PostPSA[1] are in charge respectively to prepare traces (energy calibration / time alignments), to determine hits from traces and to apply additional corrections/filters to prepare the data to be merged. At this stage one achieves the global level processing which is mainly based on the gamma-ray tracking. Such a chain is illustrated in the following picture:



Before any processing, an actor is initialized first using a configuration file which has

[1]Depending on the data flow processed, this filter may not be mandatory.

in principle the *.conf* extension. Such a file contains all the parameters required by the algorithm to work properly. This is illustrated in the following picture:



NOTE: To manage all the configuration files needed by all the actors, a python script, called *gen_conf.py* , is provided.

In addition, to determine the best parameters for a given data set, specific works (or actions) are required which are based on spectra and/or on more complex inputs (traces, events, energy) at different levels in the processing of the data flow. Several mechanisms are available to do this. Traces/events can be read from the data stream (in the ADF format) out of any actors and thus can be used to build spectra. This is the way the Watchers work. For historical reasons, spectra, traces/events can be produced also directly by the actor itself[2] and saved on disk together with the ADF files.

The spectra produced by the actors are saved in a specific format that can be read by TkT or by the GammaWare package. Specific treatments are then respectively realized through dedicated command line programs, available in the AGAPRO package, or through graphical user's interfaces in the GW/Watchers.

1.4 Replay of Data / Directories organization

The data produced by an experiment are stored in a single (unix) directory itself composed of several sub-directories one per run. For each run, one can find the data produced (Data sub-directory) at different level in the data flow as well as the files used to configure (Conf sub-directory) the different actors.

```
/agatadisks/eXXX/eXXX/run_0001.dat.14-04-17_11h12m53s
|-- Conf
|-- Data
|-- gen_conf.py
```

[2]Depending on the configuration of the processing chain at compilation and from the different *.conf* files

The two sub-directories are themselves divided in sub-directories, one per Germanium crystal for the local level processing and several ones for the ancillaries and the global level processing:

```
/agatadisks/eXXX/eXXX/run_0001.dat.14-04-17_11h12m53s
| -- Conf
| | -- 00A
| | -- 00B
| | -- ...
| | -- 14C
| | -- Builder
| | -- Global
| | -- Merger
| -- Data
| | -- 00A
| | -- 00B
| | -- ...
| | -- 14C
| | -- Builder
| | -- Global
| | -- Merger
```

Here is also a snapshot of a typical content of the bottom directories:

```
| -- Conf
| | -- 14C
| | | -- BasicAFC.conf
| | | -- BasicAFP.conf
| | | -- CrystalProducer.conf
| | | -- CrystalProducerATCA.conf
| | | -- PSAFilter.conf
| | | -- PostPSAFilter.conf
| | | -- PreprocessingFilter.conf
| | | -- PreprocessingFilterPSA.conf
| | -- Builder
| | | -- BasicAFC.conf
| | | -- CrystalPositionLookUpTable
| | | -- EventBuilder.conf
| | | -- TrackingFilter.conf
| | -- Merger
| | | -- BasicAFC.conf
| | | -- CrystalPositionLookUpTable
| | | -- EventMerger.conf
| | | -- TrackingFilter.conf
```

1.5 Replay of Data/Emulators

In the process to optimize the parameters to get the best quality as possible for your data, it is required to apply one or several times the same algorithm, or actor, using different configuration files. This could be done using NARVAL itself, on site or out site. Another possibility is to use what is called emulators i.e. other frameworks in charge of organizing / running the processing of data by many different actors.

1.5.1 Narval (and Narval standalone)

How to replay with NARVAL on site / out site ... To be written

1.5.2 Femul

FEMUL (Flat EMULator) is able to run complex topologies including DISPATCHER i.e. actors having several input lines and one output line. There are two such dispatchers, *EventBuilder* and *EventMerger*: the former being in charge of building AGATA events (crystals in coincidence from the PSA_XXXX.adf and resulting in Builder_XXXX.adf files) and the later allows one to build coincidences with ancillaries (resulting in merged_XXXX.adf files). All actors are part of the AGAPRO package. The procedure to get/compile/install this program is given in the Cookbook [see **How to install the femul emulator in the Cookbook**].

The topology (list of detectors and actors to apply on the data flow) is built using an ascii *Topology.conf* file and the way to run is[3]:

```
femul Topology.conf
```

Depending on the task you would like to perform, you may need to adapt the topology given in *Topology.conf*. The most commonly actors used are:

- Producer actors (to start a NARVAL chain):
  - CrystalProducerATCA → To start a replay from traces (*.cdat) files
  - BasicAFP → To start a replay from ADF files (Basic Agata Frame Producer)
- Filter actors (to apply calibrations/algorithms on the data flow):
  - PreprocessingFilterPSA → To apply energy and time calibrations/corrections
  - PSAFilter → To execute the Pulse Shape Analysis algorithm
  - PostPSAFilter → For final corrections after PSA (neutron damage, re-calibrations...)
  - TrackingFilterOFT → To process the tracking algorithm
- Dispatcher actors (to merge several NARVAL chains in one):
  - EventBuilder → To build a AGATA event within a specific timing window
  - EventMerger → To build a global event between AGATA and an ancillary detector in a specific timing window
- Consumer actors (to end a NARVAL chain):
  - BasicAFC → To end a replay and write ADF files on disk (Basic Agata Frame Consumer)
  - None → To end a replay without writing the output files (used when only spectra files are used in calibration procedures)

Here are three different examples of such topology file:

The first one shows how to read traces (fevent_mezzdata.cdat.*) from many crystals, run up to the PSA and then dump hits in ADF files. (eg: PSA_XXXX.adf):

```
LOOP CRY 00B 00C 01B 01C 04B 04C 05A 05B 05C 06A 06B 06C

Chain 4   CRY
Producer  CrystalProducerATCA
Filter    PreprocessingFilterPSA
Filter    PSAFilter
Consumer  BasicAFC
ENDLOOP
```

The second example shows how to read psa hits (psa*.adf) from many crystals, perform some post PSA task, build AGATA events, apply tracking and then dump tracked gamma-rays in ADF files (Tracked*.adf):

```
LOOP CRY 00B 00C 01B 01C 04B 04C 05A 05B 05C 06A 06B 06C

Chain 3     CRY
Producer    BasicAFP
Filter      PostPSAFilter
Dispatcher  EventBuilder
ENDLOOP

Chain 3     Global/
Builder     EventBuilder
Filter      TrackingFilter
Consumer    BasicAFC
```

[3]femul -h to get some help from the command line

The last example shows how to read psa hits (psa*.adf) from many crystals, perform post PSA, build AGATA events, merge them with some ancillary data before applying tracking and then dump tracked gamma-rays in an ADF file:

```
LOOP CRY 00B 00C 01B 01C 04B 04C 05A 05B 05C 06A 06B 06C

Chain 3       CRY
Producer      BasicAFP
Filter        PostPSAFilter
Dispatcher    EventBuilder
ENDLOOP

Chain 3       Builder/
Builder       EventBuilder
Consumer      BasicAFC
Dispatcher    EventMerger

Chain 2       ancillary/
Producer      BasicAFP
Dispatcher    EventMerger

Chain 3       Merger/
Builder       EventMerger
Filter        TrackingFilterOFT
Consumer      BasicAFC
```

To replay data, one should create a top destination directory containing the same structure as the one produced online. The *Conf* sub-directory and the *gen_conf.py* should be copied from the initial one, and the original Data should be linked to avoid copies of large amount of data. The *gen_conf.py* is then used to adapt the configuration files to your current directory. It will also produce an Out sub-directory where the files produced by the replay will be stored. Here is an example to prepare a replay folder for run_xxxx, where the raw data are located in /agatadisks/:

```
$ mkdir Replay
$ cd Replay
$ mkdir run_xxxx
$ cd run_xxxx
$ cp -r /agatadisks/run_xxxx/Conf ./
$ cp /agatadisks/run_xxxx/gen_conf.py ./
# (Here edit the gen_conf.py as explain in the following)
$ ln -s /agatadisks/run_xxxx/Data Data
$ python gen_conf.py
```

NOTE: The command: python "gen_conf.py -h" show a help message

When copying the *gen_conf.py* , take care of changing ONLINE into OFFLINE and NARVAL into femul.

Here is the beginning of the *gen_conf.py* script showing some parameters to be changed for the replay you would like to perform.

```
"""
Script to generate the configuration files for the replay of AGATA data using the
multi-process distributed system Narval or the single-process emulator femul.
The script is divided in a few sections, some of which are specific to the actual
analysis (and therefore are likely to be changed by the user) while some others are
normally not to be touched:
0) Type of analysis and replacement macros (in the style of the shell) used to parametrize
the commands listed in 2).
1) The structure of the actual analysis is defined by the variables PROGTYPE and CONFTYPE and
by the dictionaries Topology and Actors. The dictionary ExtraFiles contains a list of files,
which are needed by the analysis but are not generated by this script (e.g. calibrations,
mappings, ...); these files can be copied from a previous analysis if the script is started
with the option -o or --old (python gen_conf.py -h to get the list of accepted options)
2) The command lines to be written in the conf files of the actors defined in Actors{}.
Uncomment/comment/modify the command lines and their parameters
3) A small database defining the position and the PSA signal basis of the germanium crystals.
This part should not need to be changed.

To get a list of command line arguments, launch the script as:   gen_conf.py -h
To get a list of keywords accepted by the vaious actors:         femul -k
"""
##############################################################################################
###################  0  Type of analysis and replacement symbols  ############################
```

```
##############################################################################################

PROGTYPE='femul'    # NARVAL or femul   (to choose between os.getcwd() and '' for CWD)
CONFTYPE='OFFLINE'  # ONLINE or OFFLINE (used just to exclude the ReadDataDir line in the Producers)

MACROS={            # various replacements for symbols defined in 2).
'$CONFDIR'      : 'Conf',                    # this will be prefixed by CWD/
'$READDIR'      : 'Data',                    # this will be prefixed by CWD/; if ONLINE this will not be
    written
'$SAVEDIR'      : 'Out',                     # this will be prefixed by CWD/; if ONLINE this will be
    replaced by $READDIR
'$ANCILLARY'    : 'Ancillary',               # this will be prefixed by CWD/
'$GLOBAL'       : 'Global',                  # this will be prefixed by CWD/
'$PSABASE'      : '/agatadisks/bases/ADL',   # standard place at LNL/Linux
'$CRYSTAL_ID'   : "",                        # the actual value is defined in GeDataBase
'$SIGNAL_BASIS' : "",                        # the actual value is defined in GeDataBase
'$CRYSTAL'      : "",                        # the actual value taken from Topology['CRYSTAL']
}

##############################################################################################
###################  1  Structure of analysis  ###############################################
##############################################################################################

Topology={    # The directories to be generated in Conf, Data and Out
'CRYSTAL'   : "00A 00B 00C 01A 01B 01C 02A 02B 02C 03A 03B 03C 04A 04B 04C 05A 05B 05C",
'ANCILLARY' : "Ancillary",
'BUILDER'   : "Builder",
'MERGER'    : "Merger",
}

# The name of the used actors must correspond to one of the tuples defined in the following section.
# This requirement creates a problem for BasicAFP and BasicAFC when they are used in chains of different
    type
# (e.g. after PSA and after Tracking) and one wants to define chain-specific names for their input/output
    files.
# The solution is to suffix the name of the chain-type (e.g. _CRYSTAL or _GLOBAL or any other), to the
    defining tuple.
# This suffix will be silently removed from the actual name of the generated configuration files.

Actors={      # These are the xxxx.conf files to be generated
'CRYSTAL'   : "CrystalProducer BasicAFP_CRYSTAL PreprocessingFilter PSAFilter PostPSAFilter
    BasicAFC_CRYSTAL",
'ANCILLARY' : "BasicAFP_ANCYLLARY AncillaryFilter BasicAFC_ANCYLLARY",
'BUILDER'   : "BasicAFP_BUILDER EventBuilder BasicAFC_BUILDER",
'MERGER'    : "EventMerger TrackingFilter BasicAFC_MERGER",
}

ExtraFiles={  # If not already present, these files can be copied from a directory specified in the
    command line
'CRYSTAL'   : "CrystalProducerATCA.conf PreprocessingFilterPSA.conf xinv_1325-1340.cal xdir_1325-1340.cal",
'ANCILLARY' : "AncillaryFilter.conf",
'BUILDER'   : "CrystalPositionLookUpTable",
'MERGER'    : "CrystalPositionLookUpTable",
}

##############################################################################################
###################  2  Tuples specifying the content of the configuration files  #############
##############################################################################################
```

In case of replay of the PSA actor, the ADL bases need to be downloaded from the AGATA DAQ box or from Cologne web-site. Contact agata{at}ipnl.in2p3.fr for download rights. The path to the ADL bases needs to be updated in the *gen_conf.py* script (in the $PSABASE variable).

The CrystalProducer actor needs also to be modified to specify to use the traces files as input. For this, uncomment the lines “InputDataFile” and “AllInputFiles”:

```
########################
CrystalProducer=(
"ActualClass            CrystalProducerATCA",
"CrystalID              $CRYSTAL_ID",
"ReadDataDir            $READDIR/$CRYSTAL",
"SaveDataDir            $SAVEDIR/$CRYSTAL",
"TraceLength            100",
"WriteDataMask          0",
"InputDataFile          event_mezzdata.cdat",
"AllInputFiles",
)
```

Finally, make sure that the option "NoMultiHist" is commented in the different actors of the *gen_conf.py* during the calibration procedure. This allow to produce the spectra files used in the different steps of this document.

2 The different actors processing the data flow

2.1 The Producer actor

This actor is in charge to start the NARVAL chain. In online mode, it gets the traces from the front end electronics and sends it to the data flow. In offline mode, it reads input files (adf or traces files) and sends them to the NARVAL chain.

2.2 The Preprocessing actor

This actor is in charge of preparing the data for the PSA algorithm. In order to do that job, two files which are named ***PreprocessingFilter.conf*** and ***PreprocessingFilterPSA.conf*** should contain the required information.

As any detector, any AGATA capsule should be calibrated in energy. Since it is also highly electrically segmented, cross talks have a significant importance and must be corrected. For details, see for instance these publications for such effects: B.Bruyneel et al., NIMA 608 (2009)

Be aware, there might be time delays between the different segments and the core signal. Any PSA code requires having all the signals from one capsule perfectly aligned and thus time alignment are also performed.

Here are snapshots of the two required configuration files.

- ***PreprocessingFilter.conf***:

```
ActualClass          PreprocessingFilterPSA
SaveDataDir          /global/path/to/Replay/Folder/run_xxxx/Out/00A
EnergyGain           4
XtalkFile            xinv_1325-1340.cal
WriteTraces          100
```

- ***PreprocessingFilterPSA.conf***:

```
segm         0   4600   500   0.300652   15   17.220
segm         1   4600   500   0.313630   15   17.873
...
core         0   4390   500   0.482203    0   20.
core         1   4390   500   1.735468    0   20.
tntf   2097152
```

NOTE: the content may be slightly different depending on the version of the PreprocessingFilter actor used.

For the second file, here is the meaning of the different columns:

- Type : segm/core
- Id : Segments(0-35) or Cores(0-1) Id
- Tfall : Pole zero / decay time [timestamp units]
- Trise : Shaping time / Risetime [timestamp units]
- egain : Slope of energy calibration (no offset at the preprocessing level) [keV/channel]
- emink : Energy threshold [keV]
- tmove : Shift to align in time the different segments [ns]

2.2.1 Energy calibration

Explanations and goals:

In digitizers, signals are processed to extract the amplitude using a trapezoidal filter. This value is written from the beginning of the chain into the data flow. This is the value which is used to calibrate the detectors in energy. Concerning the 36 segments, because

of the various cross talks, the calibration in energy is done using events in which only one and only one segment in a given crystal has fired.

The obtained calibration coefficients are to be set in the 5th col of the ***PreprocessingFilterPSA.conf*** file.

NOTE: All the different steps of the following procedures needs to be done for each crystal. The environment variable $CryId will be used in the following to refer to the current detector (ex: CryId=00A). We suppose for the given shell command that you are working in the Conf folder of the current detector.

Tools available:

** command line programs **

To use those tools, spectra directly produced by the actors at running time are required. In particular the ones contained in the file *Data/$CryId/Prod__4-38-32768-UI__Ampli.spec*[4]:

From the file name, we can deduce that this spectra file contains 4 libraries of 38 spectra written in 32768 unsigned integer bins, containing the amplitude spectra of segments and cores. The first library contains the raw amplitude from the trapezoidal filter. The second library contains events of segment multiplicity=1, with amplitude larger than the threshold and scaled by the EnergyGain (see conf file) factor. The calibration needs to be done on spectra from the second library. The two other libraries are not used in the energy calibration but can be necessary for x-talk corrections.

From these spectra files, a C program, called *RecalEnergy*[5], is able to find peaks and thus calibrate the different channels (core and segments) for one crystal. Here is the way to use it, with some options[6]

```
$ cd Conf/$CryId  # If not already done, go in the working directory
$ RecalEnergy -spe ../../Data/$CryId/Prod__4-38-32768-UI__Ampli.spec -sub 38 -num 38 -gain 2 -60Co \
  |tee recal.out
```

Options:

-spe filename	:	Name of spectrum to analyze (mandatory)
-sub nn	:	Analysis starts from spec nn (-sub 38 to analyze the second library)
-num nn	:	Number of spectra to analyze (-num 38 for 36 segments + 2 cores)
-lim min max	:	Limit the search to this range in channels
-dwa w h	:	Default fwhm and minimum amplitude for the peak search
-60Co	:	Define the source (default is 60Co, more than one source is allowed)
-gain val	:	Scaling factor for the slope

The gain factor needs to be coherent with the gain applied to the spectra, this information can be found in the file *CrystalProducer.conf*:

```
ProjeM1              10 2 (threshold and gain)
```

A way to apply this for several crystals $DetList is the following:

NOTE: You need to be in the global folder (where Out, Data, Conf are located)

```
$ for i in $DetList ;
  do RecalEnergy -spe Data/$i/Prod__4-38-32768-UI__Ampli.spec -sub 38 -num 38 -gain 2 -60Co \
  |tee Conf/$i/recal.out;
  done
```

[4]These spectra are produced by the CrystalProducer actor.

[5]See zPrograms directory in the agapro package

[6]For a fill list of options, RecalEnergy -h

In the following, the commands will be written for only one detector, but this loop can be use to obtain automatic procedures on a list of detectors.

Check carefully (rEnergy, FM05, Chi2...) the output of each file to be sure that the fits are good. If not, reprocess the concerned files using more restrictive parameters (-lim, -dwa options). Note than in case of dead segments (see followings), the automatic procedure might not work and need to be done manually.

Note that for very bad crystals, typically neutron damaged detectors, the default fwhm parameter needs to be increased (the default value is 10) to obtain a reasonable fit result.

Once the calibration seems good the results should be inserted in the configuration file (i.e. add the *recal.out* coefficients to the 5$^{\text{th}}$ column of *PreprocessingFilterPSA.conf*) so that the processing of the data should be done properly. A python script *colupdate.py* is provided to help with this[7]:

```
$ cp PreprocessingFilterPSA.conf PreprocessingFilterPSA.save # Just in case...
$ python colupdate.py PreprocessingFilterPSA.conf recal.out -c 4 13 -o PreprocessingFilterPSA.conf
```

- -c column you want to use from file1 and file2 starting from 0 (5$^{\text{th}}$column in *PreprocessingFilterPSA.conf* (4 starting from 0), 14$^{\text{th}}$column in recal.out (13 starting from 0)
- -o file output

TkT Calibration check: Once all crystals have been calibrated, we need to process a local level replay of the Preprocessing actor to check the calibrations. For this, we need to create a dedicated folder containing a link to the raw data, the Conf folder (containing the new *PreprocessingFilterPSA.conf* files), the *gen_conf.py* script adapted to offline femul replay (see section Replay) and a Topology file similar to this:

```
LOOP CRY 00B 00C 01B 01C 04B 04C 05A 05B 05C 06A 06B 06C

Chain 3   CRY
Producer  CrystalProducerATCA
Filter    PreprocessingFilterPSA
Consumer  None
ENDLOOP
```

The "None" consumer do not write any output file (we only need here the spectra files).

For preparing the following steps, we can modify in the *gen_conf.py* file the WriteDataMask value to 8 to write the "bdat" files used for x-talk corrections. If you don't plan to perform x-talk corrections, set this value to 0.

The following lines allows to process the replay:

NOTE: You need to be in the global folder (where Out, Data, Conf are located)

```
$ python gen_conf.py # (will generate the replay Conf and Out folders)
$ femul Topology.conf
```

Open the Preprocessing spectrum file: *Out/$CryId/Prep__2-40-16384-UI__Ener.spec* with TkT to check the energy calibrations (apply the gain factor written in *PreprocessingFilter.conf* file to obtain the correct energy).

[7] See zUseful directory in the agapro package

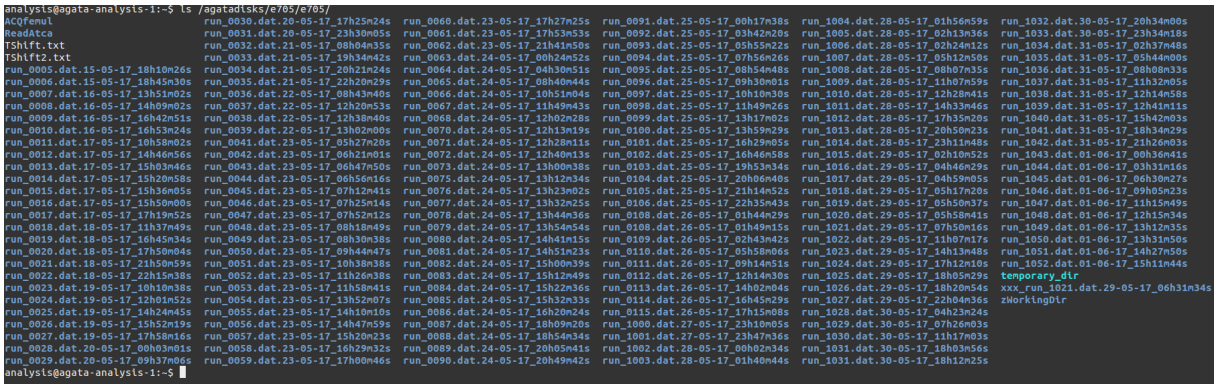
** Graphical method with GammaWare **

The SpectraViewer toolkit of the GammaWare Watchers gives the possibility to open/display the spectra created by the actors in the ROOT environment. The *RecalEnergy* (see previous paragraph) program has been implemented in this toolkit in order to allow for a direct visualization of the energy calibration procedure.

The SpectraViewer toolkit is loaded from the Watchers directory:

```
$ cd /path/to/gammasoftware/LYON/gw/demos/adf
$ root GANILLoadWatchers.C
```

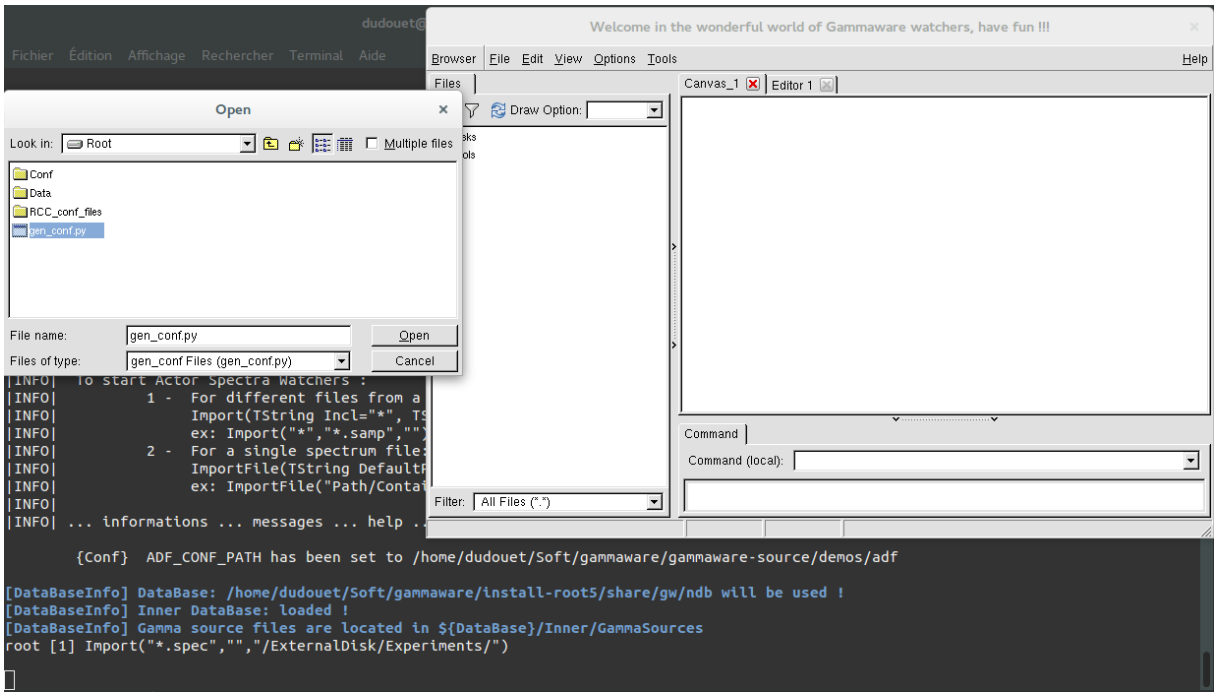
For the energy calibration tool, one needs to load the *Ampli.spec* producer files of each crystal. In this example, the runs are located in the directory "/agatadisk/e705/e705":



This is done using the Import(TString Include, TString Exclude, TString RunPath) method:

```
$ Import("Prod*Ampli.spec","","/agatadisk/e705/e705/run_1052.dat.01-06-17_15h11m44s/")
```

The *gen_conf.py* file must then be selected in the pop-up window. It will be used to get the crystal list and the different gain to apply to each spectra:



The ROOT TBrowser will then contain 4 folders :

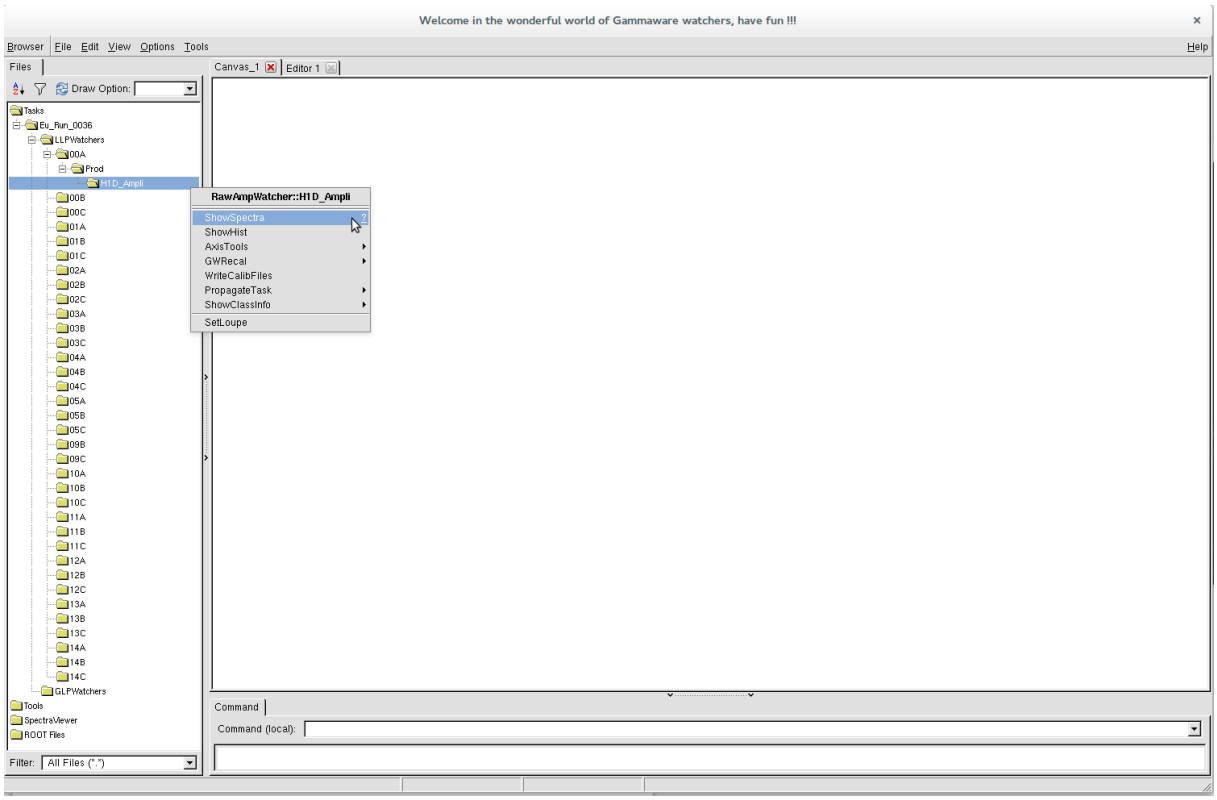
-Tasks : Used to apply operations on the spectra like energy calibrations
-Tools : Visualization Tools (not used here)
-SpectraViewer : Folder containing all the loaded actor's spectra
-ROOT Files : To get an access to the open ROOT files

To Start the calibration procedure for a given crystal, one first need to plot the crystal spectra (segments and core). For this, double left click on:

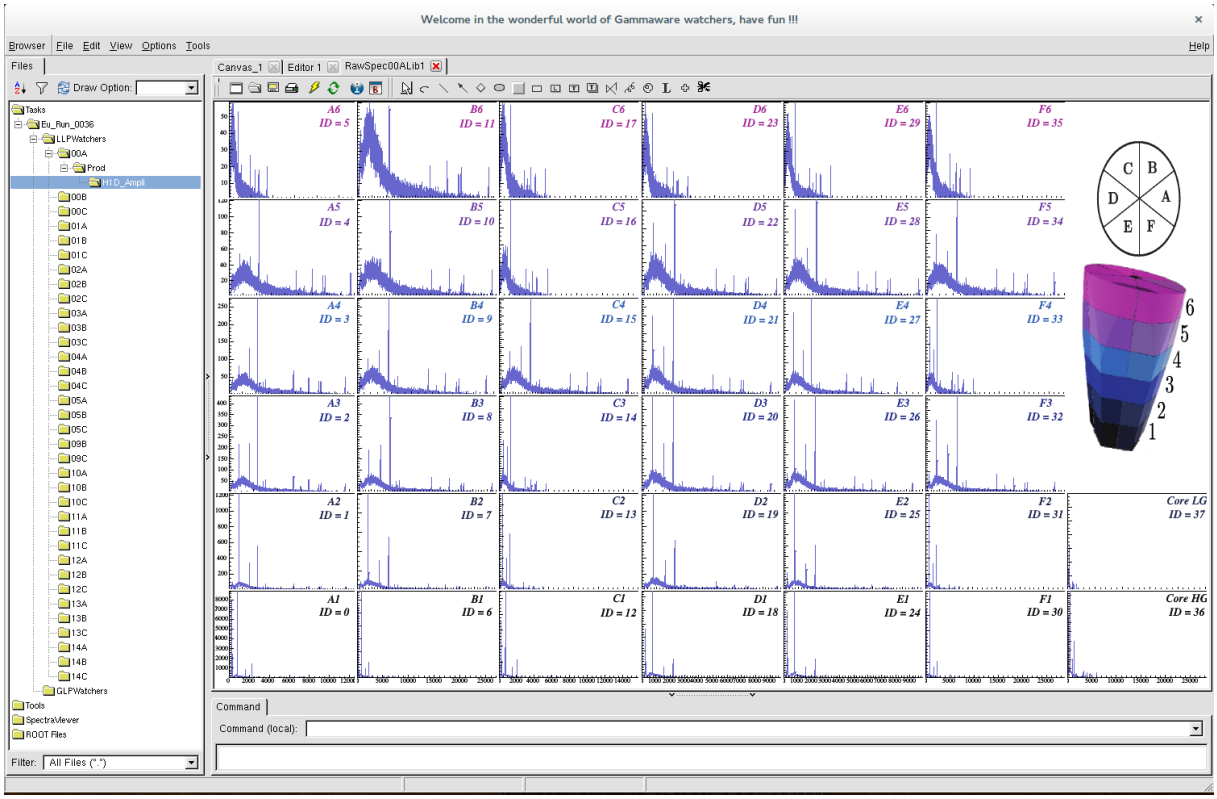
Tasks/RunName/LLPWatchers/CrystalName/Prod/H1D_Ampli

Right click on H1D_Ampli:

-ShowSpectra	: Plot the crystal map (segments and cores)
-ShowHist	: Plot the spectrum of a selected segment
-AxisTools	: Some tools to modify the range (and log scale) of the plotted pads.
-GWRecal	: Calibration tools (see below)
-WriteCalibFiles	: to store the calibrations in the *PreprocessingFilterPSA.conf* files
-PropagateTasks	: not used here
-ShowClassInfo	: not used here
-SetLoupe	: if set to true, allow to zoom on a selected pad



For the energy calibration, the second library (lib 1 starting from 0) is used (see explanations in the previous part). Use the ShowSpectra method to plot all the spectra in the crystal map. The axis tool, FullRangePerPad (*AxisTools/Range/FullRangePerPad*) can be used to define an individual automatic range to each pad.



To specify the calibration source, use:

GWRecal/Configure/TheoreticalPeaks/SetSource

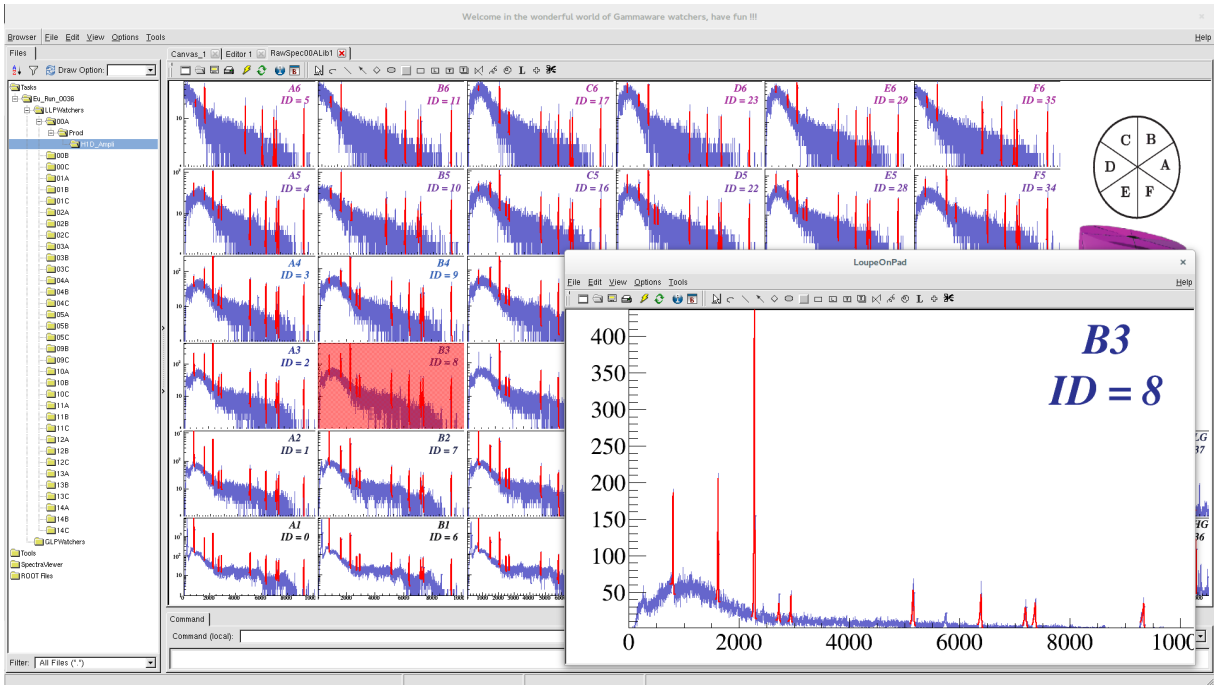
Defined sources: 22Na 40K 56Co 57Co 60Co 88Y 133Ba 134Cs 137Cs 152Eu 208Pb 226Ra 241Am (more than one source can be set, separated by a space).

The keyboard shortcut "s+s" can also be used (for Set Source)

To fit the whole crystal map with the default parameters, use:

GWRecal/Calibrate/FitAll ("Ctrl+f" shortcut)

The *SetLoupe* method can here be useful to carefully check the fit result on the different segments, when active, select a pad by a wheel click and then press the space key:



A window range can be define for all pads for the peak search using:

GWRecal/Configure/Channel/SetGlobalChannelLimits ("Ctrl + c + l" shortcut)

Or only for the selected pad using:

GWRecal/Configure/Channel/SetHistChannelLimit ("c + l" shortcut)

Peak limits (minimum FWHM and maximum) can also been defined for all pads:

GWRecal/Configure/PeakSearch/SetGlobalPeaksLimits ("Ctrl + p + l" shortcut)

Or only for the selected pad using:

GWRecal/Configure/PeakSearch/SetHistPeaksLimits ("p + l" shortcut)

When new parameters have been set for the peak search, the whole crystal map can be re-calibrated ("*Ctrl+f*"), or only the selected pad ("*f*"). Once all segments have been fine tuned, make a final fit of the whole map to print the final fit parameters. Fit results are also written on disk in the *FitResults/GWRecal* directory.

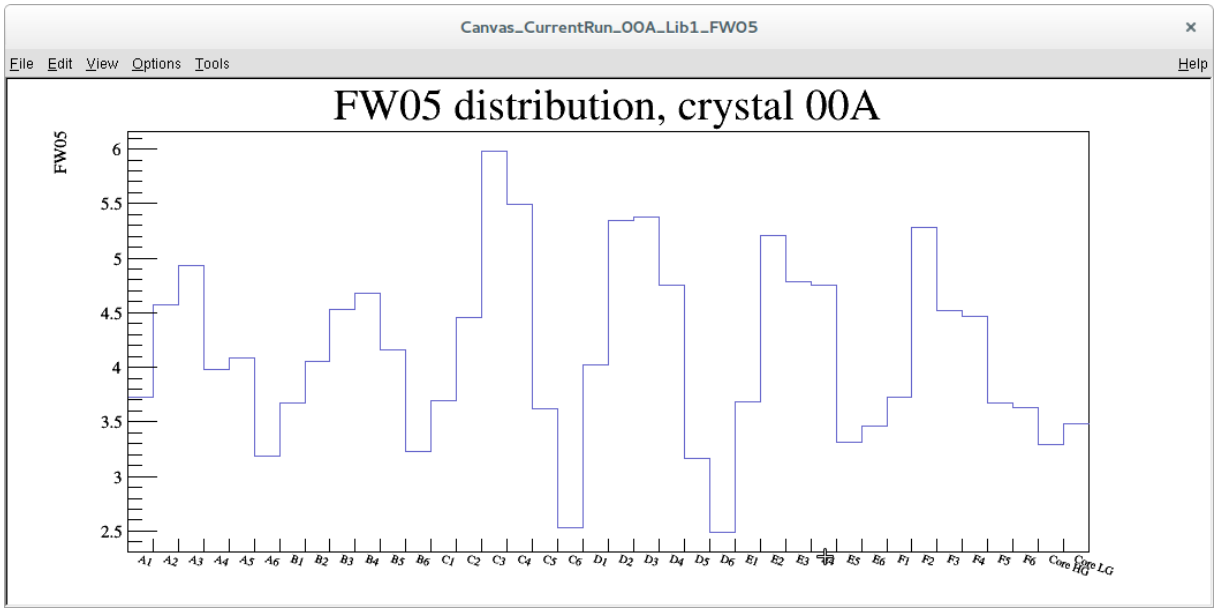
```
#  Spectra taken from file : /media/dudouet/ExternalDisk/Experiments/e705/Eu_Run_0036/Data/00A/Prod__4-38-32768-UI__Ampli.spec
#
#  lib  #spec #pks #ok   rEnergy     FW05     FW01      Area  Position  Width   Ampli    WTML    WTMR  slope*gain rChi2%
#
    1      0   28  10   1408.20    3.723    9.316      2042   9329.24   20.4      66   4.221   1.823    0.301890   3.86
    1      1   19  10   1408.93    4.572   13.721      1009   8975.26   12.5      25  11.978   1.990    0.313958  18.84
    1      2   19  10   1408.39    4.933   13.459      1422   9634.25   22.2      32   6.471   1.823    0.292371   7.09
    1      3   19  10   1408.38    3.980   10.808      1358   8803.02   16.6      42   6.298   1.823    0.319975   3.75
    1      4   10  10   1408.27    4.085   10.717       778   8739.85   18.8      24   5.239   1.823    0.322263   8.70
    1      5    8   7   1408.41    3.185    7.905       384   8608.14   16.4      16   4.080   1.823    0.327229   9.78
    1      6   24  10   1408.32    3.669    8.979      1919   9340.70   21.0      64   3.860   1.823    0.301546   2.37
    1      7   19  10   1408.38    4.049    7.381       725   8810.94   25.3      27   1.823   1.823    0.319688  10.41
    1      8   20  10   1409.18    4.530   13.186      1375   9337.56   14.6      34  10.145   1.823    0.301831  31.17
    1      9   12  10   1407.71    4.678    8.526      1049   9416.49   31.3      31   1.823   1.823    0.298988   4.28
    1     10   12   8   1408.08    4.157    7.577       593   9127.03   26.9      21   1.823   1.823    0.308553  13.62
    1     11    8   7   1408.35    3.232    7.620       372   9048.41   18.9      15   3.366   1.823    0.311292   8.45
    1     12   29  10   1408.46    3.698    9.453      1753   8517.95   17.7      62   4.630   1.823    0.330703   6.42
    1     13   20  10   1408.98    4.461   12.172       874   8878.49   18.5      24   6.471   1.823    0.317392  30.38
    1     14   18  10   1409.01    5.976   17.490      1691   9438.52   18.8      31  10.628   1.823    0.298566  25.96
    1     15   18  10   1407.95    5.492   10.014      1055   9145.88   35.7      28   1.823   1.824    0.307887  15.39
    1     16   14  10   1407.94    3.621    6.599       512   9430.52   24.3      20   1.823   1.823    0.298593   9.98
    1     17   11   8   1408.39    2.533    6.187       329   9084.28   14.1      16   3.830   1.823    0.310072   8.02
    1     18   22  10   1408.37    4.024   10.337      1231   9039.84   20.2      38   4.738   1.823    0.311592  10.29
    1     19   18  10   1407.93    5.344    9.741       512   8941.51   33.9      14   1.823   1.823    0.314921  20.93
    1     20   18  10   1408.52    5.377   14.457       939   9217.08   24.3      21   5.951   1.823    0.305632  20.48
    1     21   17  10   1407.31    4.748    8.653       626   8477.21   28.6      21   1.823   1.823    0.332022  11.43
    1     22   11  10   1408.22    3.162    8.002       416   9179.04   16.7      16   4.430   1.823    0.306833   4.20
    1     23    8   7   1407.90    2.483    5.322       193   9441.83   16.4      10   2.536   1.823    0.298225   7.77
    1     24   29  10   1408.47    3.684    9.510      1223   9197.60   18.6      40   4.842   1.823    0.306270  10.56
    1     25   19  10   1407.50    5.207    9.564       596   8675.48   32.1      17   1.823   1.851    0.324478  15.92
    1     26   20  10   1408.32    4.789   12.565      1120   8373.87   21.2      31   5.239   1.823    0.336362  28.35
    1     27   18  10   1407.93    4.754   12.921      1291   9263.10   20.9      32   6.322   1.823    0.303988   9.36
    1     28   15  10   1408.05    3.318    7.994       683   9161.08   19.1      26   3.634   1.823    0.307398   3.33
    1     29   13   8   1407.92    3.463    7.435       391   9628.30   23.3      14   2.546   1.823    0.292454   1.88
    1     30   31  10   1408.19    3.724    9.666      1768   9867.09   19.9      53   4.971   1.823    0.285431   2.71
    1     31   17  10   1408.19    5.282   14.543      1171   9186.03   21.9      26   6.830   1.823    0.306594  26.38
    1     32   17  10   1407.79    4.516    8.230      1099   9106.31   29.2      35   1.823   1.823    0.309191   4.54
    1     33   15  10   1407.74    4.470    8.147      1039   9751.79   31.0      32   1.823   1.823    0.288714   6.03
    1     34   15   9   1408.20    3.668    9.377       813   9244.93   19.1      27   4.633   1.823    0.304643   9.90
    1     35   12   7   1407.41    3.630    6.617       393   9217.33   23.8      16   1.823   1.823    0.305383  10.92
    1     36   52  10   1407.99    3.296    6.772     40858   5931.57   13.8    2590   2.298   1.823    0.474745   1.11
    1     37   23  10   1408.06    3.481    7.502     40894   1652.52    4.0    8602   2.572   1.823    1.704130 274.27
Calibration File write in: FitResults/GWRecal/CurrentRun/00A_Amplitude_Lib1.fit
```

For any reminder on the possible commands and keyboard shortcuts:

GWRecal/PrintHelp ("p+h")

Once a crystal map has been correctly fitted, a "PlotResults" tool allows to print the different parameters versus the segment ID, or versus other parameters :

GWRecal/PlotResults/PlotCalibResults ("p+c")



A detailed explanation of this PlotResult tool can be shown using:

GWRecal/PlotResults/PrintPlotHelp

Finally, a global fit of the whole crystals can be done in a single command, using the PropagateTask method on the "LLPWatchers" folder. A detailed help on this tool can be displayed by a right click on LLPWatchers folder and:

PropagateTask/PrintHelp

The PropagateTask tool can be used using:

PropagateTask/PropagateTask

For example, the command:

"ShowSpectra Lib=1 ; Recal Source=152Eu PL=20,10 ;
Plot Exp=FW05:spec Opt=same ; all"

will plot the first library of all segments of all crystals. Spectra will then be fitted with the Recal code using an ^{152}Eu source and limiting the peak search to peaks FWHM <20 and amplitude >10. FWHM of the reference peak for each crystal will finally been plotted on a same Canvas.

To store the fit results in the *PreprocessingFilterPSA.conf* file, you can use the *coluplate.py* script (see the previous section) using the files written in the FitResults folder, or use the WriteCalibFiles option (right click on a crystal), and giving the path of the Conf folder (it will be applied for the whole crystals)

2.2.2 Pole-zero/shaping-time adjustments

Explanations and goals:

The **trapezoidal filter** is applied in the pre-processing actor. We have here the possibility to change the shaping time and tune the decay time of the exponent for the experiment (depending on the rate/crystal).

To distinguish the meaning of pole-zero and shaping-time, in the calibration scripts:

- *pole-zero* is related to the *decay-time*
- *shaping-time* to the *rise-time*

In order to set the pole-zero of the cores (for the segments one should leave them like they are) it is necessary to fit the *exponential of the long traces* (this is done on site by the local team when a new crystal is added in the topology).

NOTE: this part is done on site at the beginning of an experiment and should not be modified afterwards.

Access to the following account-machine from the AGATA server:

```
$ ssh psa-tests@scgw3
$ cat config_crystal/crystal_${CryId}.rc
```

```
...
all risetime  500
all baseline  4
all polezero  4600
CC  channel   1
CC  threshold 100
CC  polezero  4606
...
```

These coefficients for pole zero and rise should be integrated in the *PreprocessingFilterPSA.conf*.

2.2.3 Crosstalk corrections

Explanations and goals:

Crosstalk is present in any segmented detectors. Through couplings, the collection of the signal in one segment modifies the signal collected in the neighboring ones. One consequence is a shift in energy (applying a calibration based on events with just one segment fired) which increases as a function of the number of segment fired in one crystal. There are different crosstalk in segmented Germanium crystals, in particular differential and proportional. To have more information on such effect in different segmented Germanium detectors, you can read the following papers (non-exhautive list):

Vetter NIMA 452 (2000), Swensson NIMA 540 (2005), B.Rossé NIMA 565 (2006), B. Bruyneel NIMA 608 (2009)

Concerning the AGATA capsule, it is possible use the crosstalk correction procedure to recover up to one broken or missing segment per crystal (by using the fact that sum of the energies of the segments should be equal to the core energy).

Tools available:

** TkT and command line programs **

In order to perform this correction, we will use the files with the extension *.bdat (**event_energy.bdat**) produced by the CrystalProducer if the 4th bin in the WriteDataMask instruction of *CrystalProducer.conf* is set ("WriteDataMask 8"). The program used to perform this is called *SortCapsule*. Xtalk corrections needs to be done using a 60Co source.

1. First of all, it is necessary to write the preprocessing calibration file in a new format, use the following command to prepare the file in the proper format for the next steps:

```
$ cd Conf/$CryId          # Go in the working directory if not already done
$ rm -f ecalF1.cal        # if file was already existing
$ for i in {1..38} ; do cat PreprocessingFilterPSA.conf | head -n $i |awk -v var=$(( $i - 1 )) \
  '{print "0\t" var "\t2\t0\t" $5}' | tail -n 1 >> ecalF1.cal ;
  done ; cat ecalF1.cal;
```

The ecalF1.cal file should be similar to:

```
0 0 2 0 0.301769
0 1 2 0 0.313686
...
0 36  2 0 0.484332
0 37  2 0 1.740927
```

2. Verify the calibration coefficients:

```
$ xTalkSort -ifile ../../Out/$CryId/event_energy.bdat.0000 -ecalF1 ecalF1.cal -egain 5
```

This will generate the following files for each crystal:

```
-proj__3-38-32768-UI__raw.spec raw projections     :  [0]=Tot   [1]=SG@F1   [2]=CC@F1
-spec__3-38-16384-UI__cal.spec calibrated spectra  :  [0]=Tot   [1]=SG@F1   [2]=CC@F1
-ssum__2-12-16384-UI__cal.spec sumEnergy[segFold]  :  [0]=SG    [1]=CC
```

with [N] the library number.

The two last files can be read using TkT to check the segments energies. Energies needs to be good for fold 1, but note that the position of the peaks in the first set of spectra in ssum__2-12-16384-UI__cal.spec depends on the number of fired segments (Fold). No such dependency for the second set (core).

3. Call *xTalkSort* to sort and analyze the AGATA events dumped into event_energy.bdat.0000 by producing the cross-talk spectra for full energy release of 1332.5 keV in one segment (F1)

```
$ xTalkSort -ifile ../../Out/$CryId/event_energy.bdat.0000 -ecalF1 ecalF1.cal -egain 5 -specXT \
  -trigewin 1325 1340
```

with:

- ecalF1 : file with energy calibration coefficients for singles extracted above
- egain : gain factor for energy spectra
- specXT : generate cross talk spectra from Mseg=1
- trigewin : energy window on trigger channel

4. Once the different spectra have been created, the *RecalEnergy* program can be used to build the direct Xtalk matrix:

```
$ RecalEnergy -spe xspe__36-37-16384-UI__cal.spec -num 1332 -ener 1332.5 -gain 5 -offs 1000 \
  -Xtalk 37 |tee 1325-1340.txt
```

with:

- Xtalk 37 : Use the 37^{th}spectrum for the core in the Xtalk calculation
- offs 1000 : channel offset to subtract to the position of the peaks

5. Transform this file to the proper format for direct cross talk coefficients:

```
$ grep -v "^#" 1325-1340.txt |grep -v "^ *36 " |cut -b15-102 --complement |tee xdir_1325-1340.cal
```

6. Invert the cross talk coefficients and rename it:

```
$ xTalkInvert -f xdir_1325-1340.cal ; mv xdir_1325-1340.cal.inv xinv_1325-1340.cal
```

The inverse matrix is given the name of xdir_1325-1340.cal.inv. The traditional name of this file was called xinv_1325-1350.cal. Up to you to rename the file or keep the given name.

7. Call again *xTalkSort* using the Xtalk inverted matrix to check the corrected spectra:

```
$ xTalkSort -ifile ../../Out/$CryId/event_energy.bdat.0000 -ecalF1 ecalF1.cal -egain 5 \
  -recalXT xinv_1325-1340.cal
```

This will produce the spectra files:

```
-proj__3-38-32768-UI__raw.spec raw projections       :  [0]=Tot   [1]=SG@F1   [2]=CC@F1
-spec__3-38-16384-UI__cal.spec calibrated spectra    :  [0]=Tot   [1]=SG@F1   [2]=CC@F1
-spec__3-38-16384-UI__adj.spec adjusted spectra      :  [0]=Tot   [1]=SG@F1   [2]=CC@F1
-ssum__2-12-16384-UI__cal.spec sumEnergy[segFold]    :  [0]=SG    [1]=CC
-ssum__2-12-16384-UI__adj.spec sumAdjusted[segFold]  :  [0]=SG    [1]=CC
```

with [N] the library number.

Check with TkT the spectra in spec__3-38-16384-UI__adj.spec and ssum__2-12-16384-UI__adj.spec. In particular the shift in position of the sum energy of the segments as a function of the segment fold should have disappeared.

Specific case of "lost" or "broken" segments:

The part extends the treatment to the case of one **dead** segment which can be "**lost**" or "**broken**".

A **broken** segment is the result of a problem at the FET level with the consequence that the charge of the segment is not collected but flows to the neighbors. The salient effect of a **broken** segment is the presence of ghost peaks in the neighbors and of a strong step-like tail in the spectrum of the core. The ghost peaks and the left step can be seen as enhanced cross-talk.

The case of a **lost** segment is when the detector works normally but the signal is not present in the data due, e.g., to a broken wire or a faulty digitizer channel. In this case there are no ghost peaks.

It is worth remarking that segments with **unstable** gain could be transformed into (and treated as) **lost** segments by setting their energy calibration to zero. Of course this is possible only if all other segments in the detector work correctly.

To treat **dead** segments, we have to find a way to quantify the amount of missing energy. For **broken** segments, we have also to generate a specific set of cross-talk correction coefficients capable of removing the ghost peaks from the affected neighbors. Two different ways of treating this are possible

1. In the calibration file adapted for Xtalk calculations (*ecalF1.cal*) **and** in the *PreprocessignFilterPSA.conf*, the calibration parameter of the **dead** segment **must to be set to 0**.

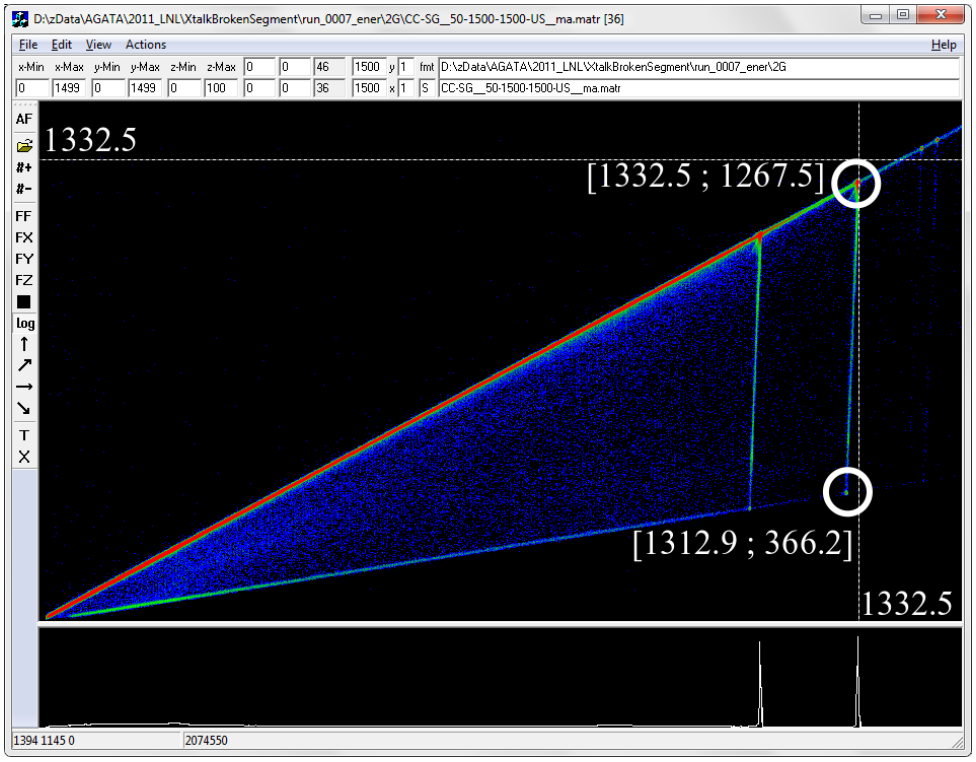
   NOTE: For the command lines in the following, it is supposed that you are in the Conf folder of the concerned detector named in the following: "$CryId"

2. To determine the energy released in the missing segment:

   ```
   $ xTalkSort -ifile ../../Out/$CryId/event_energy.bdat.0000 -ecalF1 ecalF1.cal -matCCSG
   ```

   Open with *Mat.exe* the file CC-SG__50-1500-1500-US__ma.matr. Matrix #36 shows the correlation between the energy seen by the core and the sum-energy of all segments.

   For a **broken** segment it should look like the following:



The main diagonal is not at 45° (as usual when the sum-energy is over all segments) and there is a lower limit line for the lost in the sum of segments (because part of the energy released in the **broken** segment is "collected" in its close neighbors, generating the "ghost peaks") and also a small loss of core energy (the slightly tilted lines coming down from the peaks which would be perfectly vertical if the segment would be disconnected).

The coefficients to manage the broken segment (**here and also in PreprocessingFilterPSA**) are:

- **deadXsg** = 1267.5/1332.5 = 0.95122
  → **slope of main diagonal**
- **deadXcc** = (1332.5-1312.9)/(1267.5-366.2) = 0.021746
  → **1/slope of core loss**

For a **lost** segment:

the main diagonal has practically the same slope but the lower line is not present and the lines out of the peaks are just vertical. The coefficient to manage the lost segment are:

- **deadXsg** = 0.95122
  → **slope of main diagonal**
- **deadXcc** = 0
  → **no core loss**

3. We can verify these values from the core-segment correlation matrices produced after giving the index of the dead segment and the two coefficients:

   For a **lost** segment (case of dead segment id = 4 [A5]):

   We will use in the following the environment variable $DeadSeg=4

```
$ xTalkSort -ifile ../../Out/$CryId/event_energy.bdat.0000 -ecalF1 ecalF1.cal \
  -deadSeg $DeadSeg 0.95122 0. -matCCSG
```

The off diagonal stuff in sCC-eSumSG has disappeared by construction because the missing energy has been assigned to the lost segment. Indeed A5 is now present with the correct energy and we are essentially back to the normal situation and we can produce the cross-talk matrix as:

```
$ xTalkSort -ifile ../../Out/$CryId/event_energy.bdat.0000 -ecalF1 ecalF1.cal \
  -deadSeg $DeadSeg 0.95122 0. -matx1
```

and the cross talk correction coefficients as in the normal case:

```
$ xTalkMake -f xSG__36-36-100-1536-US__ij.matr
```

Test the result (see step 7 of standard Xtalk corrections):

```
$ xTalkSort -ifile ../../Out/$CryId/event_energy.bdat.0000 -ecalF1 ecalF1.cal \
  -deadSeg $DeadSeg 0.95122 0. -recalXT xinv_1325-1340.cal
```

For a **broken** segment (case of dead segment id = 4 [A5]):

```
$ xTalkSort -ifile ../../Out/$CryId/event_energy.bdat.0000 -ecalF1 ecalF1.cal \
  -deadSeg $DeadSeg 0.95122 0.021746 -matCCSG
```

Also for this case the off diagonal stuff in sCC-eSumSG has disappeared and the matrix of the broken segment is no more empty. However, the assigned energy is much too small due to the fact that part of it is still in the neighbors. The slope of the diagonal line in the two matrices is:

- **slopeDeadSG** = 897.5/1332.5 = 0.6734
  → **(matrix #4) needed for xTalkMake**

To correct the remaining effects we need to determine the proper xTalk coefficients. The procedure described in the first part of this document is not suited as it looks only for slightly negative-energy peaks, missing completely the positive-energy ghosts. The program manages this case from the fact that the third parameter of the command line switch "-deadSeg" is positive:

```
$ xTalkSort -ifile ../../Out/$CryId/event_energy.bdat.0000 -ecalF1 ecalF1.cal -egain 5 \
  -deadSeg $DeadSeg 0.95122 0.021746 -matx1
```

In the generated file : xSG__36-36-1000-100-US__ij.matr, like for the standard xTalk matrices the x axis (second-last index) is the spectrum of the "affected segment" while the y axis (last index) is the spectrum of the "affecting (net-charge) segment". In this case the 100 channels-long Y axis is a [1300-1350] keV window on the core energy (0.5 kev/chan), while the 1000 channels-long x-axis is the spectrum of the affected segment for the energy range [-15 185]-keV (at a gain of 5 channels/keV). The matrix is incremented only if one segment sees more than 60% (>800 keV) of the core energy and that segment is considered as the affecting one. Have a look to the matrices [4][seg] to see the location of the ghost peaks in close neighbours of A5. Then, calculate the cross-talk correction coefficients by modifying the diagonal element of the broken segment with:

```
$ xTalkMake -f xSG__36-36-1000-100-US__ij.matr -dxy 1000 100 -gate 55 75 -xcal -15 5 \
  -egam 1332.5 -dval $DeadSeg 0.6734
```

To be consistent with the Xtalk file names, rename the matrices:

```
$ mv xdir_0055-0075.cal xdir_1325-1340.cal
$ mv xinv_0055-0075.cal xinv_1325-1340.cal
```

Finally, test the result with (see step 7 of standard Xtalk corrections):

```
$ xTalkSort -ifile ../../Out/$CryId/event_energy.bdat.0000 -ecalF1 ecalF1.cal \
  -deadSeg $DeadSeg 0.95122 0.021746 -recalXT xinv_1325-1340.cal
```

4. In the *gen_conf.py* , define in the PreprocessingFilter and PSA actors the dead segments as in the example bellow with 00A as a broken segment, and 11A as a lost one:

```
#########################
PreprocessingFilter=(
"ActualClass           PreprocessingFilterPSA", # name of the used daughter class
"SaveDataDir           $SAVEDIR/$CRYSTAL",      # normally Out/01A...
"EnergyGain            4",                      # channels/keV of the calibrated energy spectra
"XtalkFile             xinv_1325-1340.cal",     # cross talk correction coeffs for the energies
"WriteTraces           100" ,                   # number of traces written
#### command lines to be produced only for the specified crystals
{
'00A' : ("DeadSegment    29 0.950 0.049"), # case of a broken segment
'11A' : ("DeadSegment    18 0.947 0."),    # case of a lost segment
}
)
#########################
PSAFilter=(
"ActualClass           PSAFilterGridSearch",    # name of the used daughter class
"BasisFile             $SIGNAL_BASIS",          # this is generated from the GeDataBase structure
"SaveDataDir           $SAVEDIR/$CRYSTAL",      # normally Out/Data(online)
"EnergyGain            4",                      # channels/keV of the calibrated energy spectra
"XtalkFile             xdir_1325-1340.cal",     # cross talk correction coeffs for traces
"Threads               5 300",                  # number of threads, events/thread
"GridSearchType        Adaptive" ,              # SegCenter, Adaptive, CoarseOnly or Full;
#### command lines to be produced only for the specified crystals
{
'00A' : ("DeadSegment    29"),
'11A' : ("DeadSegment    18"),
}
)
```

Check also that the files: xinv_1325-1340.cal xdir_1325-1340.cal are written in the "ExtraFiles" list of the *gen_conf.py* :

```
ExtraFiles={
'CRYSTAL'    : "CrystalProducerATCA.conf PreprocessingFilterPSA.conf xinv_1325-1340.cal
     xdir_1325-1340.cal",
}
```

2.2.4 Time alignment

Explanations and goals:

The Preprocessing has another very important "ingredient", i.e. the possibility to aligned in time (based on Straight line fit or Digital CFD of the signal rise-time) the traces of the different segments to the core one.

This operation is very important to obtain good performances of the PSA algorithm. In other words, the PSA tends to 'process' all the waveforms as beginning at the same time and therefore, if there is any mis-match in time, the reconstructed interaction points are prone to be unreliable. A bad time alignment can result in clusterisation patterns at the output of the PSA.

Tools available:

** command line programs **

The spectra to be used are in the *Prep__6-40-1000-UI__TT.spec*[8] (before was *Prep__2-*

[8]This set of spectra is produced by the Preprocessing filter.

40-1000-UI__TT1.spec). Library 1 (starting from 0) corresponds to timing before applying any shift. It must be used for time calibration. Library 3, after time shift, can be used to check the quality of the time alignment.

1. Time alignment of segments:

   The command lines to be used per crystal is the following:

```
$ cd Conf/$CryId  # If not already done, go in the working directory
$ RecalEnergy -spe ../../Out/$CryId/Prep__6-40-1000-UI__TT.spec -sub 40 -num 36 -T 500 \
  |tee shift_TT.out
```

   Options:

   -sub 40 : Second library
   -num 36 : 36 segments
   -T 500 : we want all the peaks at this position

   Add this coefficients to the 7th column of *PreprocessingFilterPSA.conf* using the *colupdate.py* script:

```
$ cp PreprocessingFilterPSA.conf PreprocessingFilterPSA.save2 # Just in case...
$ colupdate.py PreprocessingFilterPSA.conf shift_TT.out -c 6 13 \
  -o PreprocessingFilterPSA.conf
```

2. Time alignment of the core:

   The time alignment of the core is done using the spectrum file produced by the PSA actor name "Psa__40-1000-UI__Tzero.spec" (Here we don't care about what the PSA actor is doing, we only use this spectrum file). A replay from traces to PSA is thus required, taking into account the new time calibration of the segments.

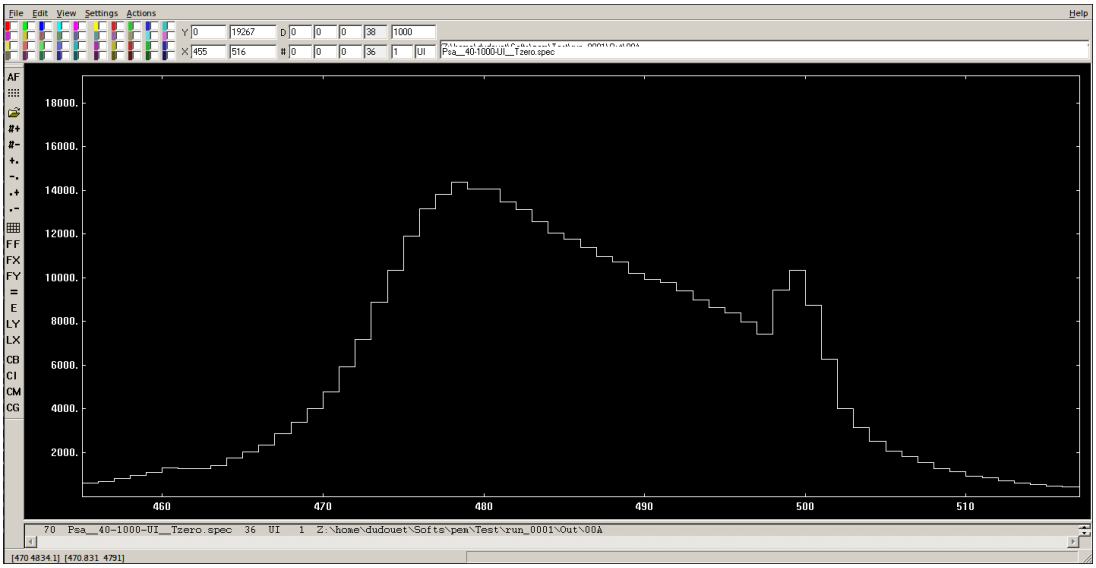
   NOTE: Be careful to have in the *gen_conf.py* file, the option "NoMultiHist" **commented** in the PSA actor (otherwise the spectra files will not be produce).

   This kind of topology can be used:

```
LOOP CRY 00B 00C 01B 01C 04B 04C 05A 05B 05C 06A 06B 06C

Chain 4   CRY
Producer  CrystalProducerATCA
Filter    PreprocessingFilterPSA
Filter    PSAFilter
Consumer  None
ENDLOOP
```

   The statistic needed here is not very important, few minutes should be enough. To align the timing of the core, look at the histogram n° 36 of the file Psa__40-1000-UI__Tzero.spec as shown below:



A well align core must contain only one peak, centered at channel 500. In the example above, a broad peak is observed with a peak position at 478. A shift of 22 channels is thus needed. Apply this shift to the timing value of the two cores set in the *PreprocessingFilterPSA.conf* file:

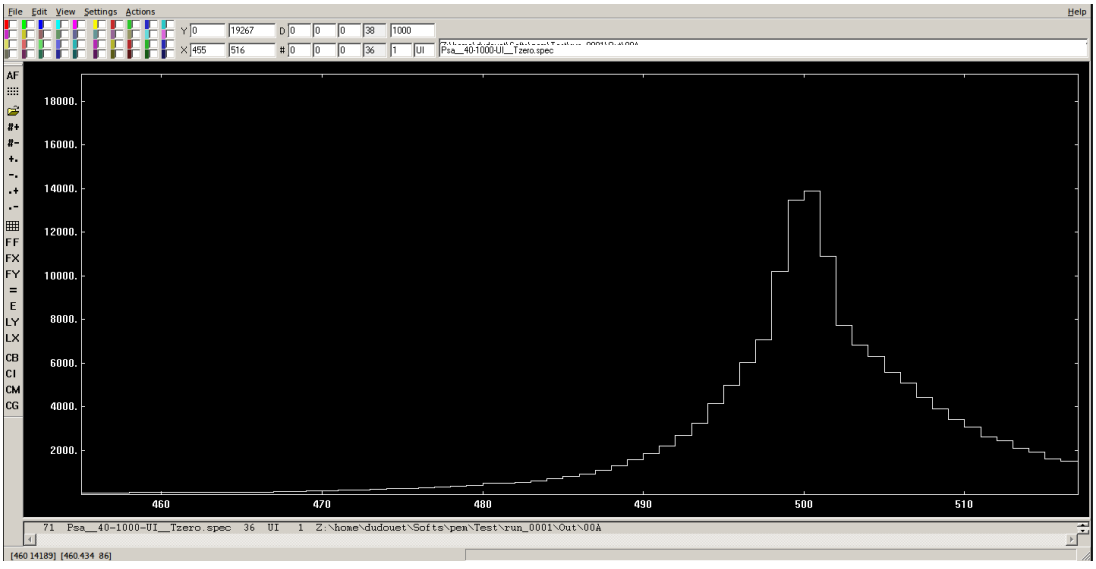
Before core alignment:

```
...
core          0    4390    500    0.482203    0       5.
core          1    4390    500    1.735468    0       5.
tntf    2097152
```

After core alignment:

```
...
core          0    4390    500    0.482203    0       27.
core          1    4390    500    1.735468    0       27.
tntf    2097152
```

Once the core has been aligned, start a new short replay to check the result, it should be similar to:



Finally, reprocess a replay to check the quality of the time alignment, looking at the spectra file "*Prep__6-40-1000-UI__TT.spec*" (library 3)

2.3 The Pulse Shape Analysis (PSA) actor

The PSA actor is used to extract from the signals shapes the position of each interaction point with a 5mm precision. No specific calibration are needed for this step. But the quality of the PSA is highly dependent on the good calibrations at the Preprocessing level.

2.4 The PostPSA actor

The PostPSA filter actor allows to make all the final operations on the local level data (neutron damage correction, final energy calibration with an offset and core time alignment).

2.4.1 neutron damage corrections

The damages caused by interactions between neutrons and the Germanium detectors deteriorate the gamma spectra quality. The typical effect of this neutron damages is a left tail on the peaks. This detector deterioration is increasing along time and its correction is mandatory to obtain a satisfying energy resolution.

This corrections is done in three steps:

1. First, a recalibration of the segments and core can be performed before the neutron damage correction.
2. The program named "*SortPsaHits*" is then used to estimate the neutron damages and correct the energies.
3. After the neutron damage correction, a final recalibration is processed to correct from possible shifts induced by the neutron damage correction.

Neutron damage correction procedure:

0. First, it is necessary to do a replay of a ^{60}Co source run from the traces including the "*WritePsaHits*" option in the PSA actor of the *gen_conf.py* file:

   NOTE: It is necessary to have configured the correct path to the ADL signal bases in the *gen_conf.py* (see the Replay section)

```
#########################
PSAFilter=(
"ActualClass            PSAFilterGridSearch",    # name of the used daughter class
"BasisFile              $SIGNAL_BASIS",          # this is generated from the GeDataBase structure
"SaveDataDir            $SAVEDIR/$CRYSTAL",      # normally Out/Data(online)
"EnergyGain             4",                      # channels/keV of the calibrated energy spectra
"XtalkFile              xdir_1325-1340.cal",     # cross talk correction coeffs for traces
"Threads                5 300",                  # number of threads, events/thread
"GridSearchType         Adaptive" ,              # SegCenter, Adaptive, CoarseOnly or Full;
"WritePsaHits" ,                                 # writes the hits in binary
)
```

   This will produce the spectra files: *Psa__0-16-F__Hits.fdat*, needed to determine the coefficients of the neutron damage corrections.

   This kind of topology can be used:

```
LOOP CRY 00B 00C 01B 01C 04B 04C 05A 05B 05C 06A 06B 06C

Chain 4   CRY
Producer  CrystalProducerATCA
Filter    PreprocessingFilterPSA
Filter    PSAFilter
Consumer  BasicAFC
ENDLOOP
```

The use of the BasicAFC consumer here will write the *psa*.adf* files that will be used for faster replays in the following.

In your calibration folder, create a new symbolic link "Out" linked to the output of this new replay

1. Prepare a dummy trapping file:

```
$ cd Conf/$CryId          # Go in the working directory if not already done
$ rm -f Trapping.cal      # If already existing
$ for i in {0..35} ; do echo -e "$i\t1.\t1.\t999999.9\t999999.9\t1.\t1." >> Trapping.cal ;
  done ; cat Trapping.cal
```

This file should be like following:

```
0 1.  1.  999999.9  999999.9  1.  1.
1 1.  1.  999999.9  999999.9  1.  1.
...
...
34  1.  1.  999999.9  999999.9  1.  1.
35  1.  1.  999999.9  999999.9  1.  1.
```

It can be understood as follow:

- column 0: segment id
- column 1: extra gain on the segment before correction
- column 2: extra gain on the core before correction
- column 3: electron-trapping correction (lambdaE)
- column 4: hole-trapping correction (lambdaH)
- column 5: extra gain on the segment after correction
- column 6: extra gain on the core after correction

2. The program *SortPSAHits* is necessary to sort PSA hits in a specific format to determine neutron damage correction parameters:

```
$ SortPsaHits -f ../../Out/$CryId/Psa__0-16-F__Hits.fdat -best 1300 1350 -bpar 1 10000 0 \
  |tee sort_hit.log
```

3. Update the dummy trapping file:

```
$ tail -n 39 sort_hit.log |head -n 36 > sort_hit_nohead.log;
$ awk 'FNR==NR{a[NR]=$4;next}{$4=a[FNR]}1'  sort_hit_nohead.log Trapping.cal \
  |awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' > Trapping_tmp.cal;
$ awk 'FNR==NR{a[NR]=$5;next}{$5=a[FNR]}1'  sort_hit_nohead.log Trapping_tmp.cal \
  |awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' |tee Trapping.cal;
```

The columns corresponding to neutron damage correction should now be updated in the Trapping.cal files like following:

```
 0         1.         1.       51.6       13.4         1.         1.
 1         1.         1.     1482.1       11.1         1.         1.
...
...
34         1.         1.      326.8       11.1         1.         1.
35         1.         1.      385.5       11.1         1.         1.
```

4. Generate of the file *Pso__2-4-40-2048-UI__Ener.spec* for the recalibration of the segment before neutron correction:

```
$ SortPsaHits -f ../../Out/$CryId/Psa__0-16-F__Hits.fdat -gain 5 -offs 5000 -fcal Trapping.cal
```

This *Pso__2-4-40-2048-UI__Ener.spec* file contains two spectra libraries of 4 sub libraries of 40 spectra:

- Family 0: Segments
  - Library 0: original spectra
  - Library 1: original spectra + recal before neutron damage correction
  - Library 2: spectra after neutron damage correction
  - Library 3: neutron correction + final recalibration
- Family 1: Core
  - Library 0: ...

5. If necessary, recalibration of the segments and cores before neutron correction (the search peaks parameters "-dwa" probably needs to be adapted):

   - segments recalibration (Family 0, Library 0):

```
$ RecalEnergy -spe Pso__2-4-40-2048-UI__Ener.spec -num 36 -sub 0 -gain 5 -offs -5000 \
  -noTR -dwa 30 5 | tail -n 36 |tee log_sg_pre.cal
```

   - cores recalibration for each segment (Family 1, Library 0):

```
$ RecalEnergy -spe Pso__2-4-40-2048-UI__Ener.spec -num 36 -sub 160 -gain 5 -offs -5000 \
  -noTR -dwa 30 5 | tail -n 36 |tee log_cc_pre.cal
```

6. Insert the pre-trapping recalibration parameters in the trapping file:

   - segments:

```
$ awk 'FNR==NR{a[NR]=$14;next}{$2=a[FNR]}1' log_sg_pre.cal Trapping.cal \
  |awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' \
  |tee Trapping_tmp.cal
```

   - cores:

```
$ awk 'FNR==NR{a[NR]=$14;next}{$3=a[FNR]}1' log_cc_pre.cal Trapping_tmp.cal \
  |awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' \
  |tee Trapping.cal
```

   NOTE: Check carefully that the good peaks have been fitted, the coefficients should be all around 1. with a ∼‰ precision

   The columns corresponding to pre-calibration should now be updated in the *Trapping.cal* files like following:

```
 0   0.999311   1.000675      51.6       13.4         1.         1.
 1   1.000594   1.000559    1482.1       11.1         1.         1.
...
...
34   0.999558   1.000857     326.8       11.1         1.         1.
35   0.999976   1.000779     385.5       11.1         1.         1.
```

7. Apply the recalibration:

```
$ SortPsaHits -f ../../Out/$CryId/Psa__0-16-F__Hits.fdat -gain 5 -offs 5000 -fcal Trapping.cal
```

   This will re-generate the file *Pso__2-4-40-2048-UI__Ener.spec* for the next step

8. Post trapping recalibration of the segments and cores after neutron correction (the search peaks parameters "-dwa" probably needs to be adapted):

   - segments recalibration (Family 0, Library 3):

```
$ RecalEnergy -spe Pso__2-4-40-2048-UI__Ener.spec -num 36 -sub 80 -gain 5 -offs -5000 \
  -dwa 30 5 | tail -n 36 |tee log_sg_post.cal
```

- cores recalibration for each segment (Family 1, Library 3):

```
$ RecalEnergy -spe Pso__2-4-40-2048-UI__Ener.spec -num 36 -sub 240 -gain 5 -offs -5000 \
  -dwa 30 5 | tail -n 36 |tee log_cc_post.cal
```

NOTE: As we are here working on the recalibration after trapping, the spectra number offsets has been adapted: 80(Family 0, library 2) for segments, and 240(Family 1, library 2) for cores.

9. Insert the post-trapping recalibration parameters in the trapping file:

   - segments:

```
$ awk 'FNR==NR{a[NR]=$14;next}{$6=a[FNR]}1'  log_sg_post.cal Trapping.cal \
  |awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' \
  |tee Trapping_tmp.cal
```

   - cores:

```
$ awk 'FNR==NR{a[NR]=$14;next}{$7=a[FNR]}1'  log_cc_post.cal Trapping_tmp.cal \
  |awk '{printf "%2s %10s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' \
  |tee Trapping.cal
```

   The columns corresponding to post-calibration should now be updated in the *Trapping.cal* files like following:

```
0   0.999311   1.000675       51.6       13.4   0.998467   0.998754
1   1.000594   1.000559     1482.1       11.1   0.999063   0.999600
...
...
34   0.999558   1.000857      326.8       11.1   0.999425   0.999294
35   0.999976   1.000779      385.5       11.1   0.999054   0.999287
```

10. Re-generation of the file *Pso__2-4-40-2048-UI__Ener.spec* for final checks (be careful, a gain of 5 and an offset of 5000 is applied, the 1333 peak will thus be located at channel 1665):

```
$ SortPsaHits -f ../../Out/$CryId/Psa__0-16-F__Hits.fdat -gain 5 -offs 5000 -fcal Trapping.cal
```

    Finally, the *gen_conf.py* needs to be adapted in the PostPSA actor part, as follow (example for crystal 00A):

    NOTE: If this line is already present, with other keywords, **remove them**, they will be added lated and **must not be present for the moment**.

```
#########################
PostPSAFilter=(
"ActualClass           PostPSAFilter",          # name of the used daughter class
"SaveDataDir           $SAVEDIR/$CRYSTAL",      # normally Out/Data(online)
"EnergyGain            4",                      # channels/keV of the calibrated energy spectra
#### command lines to be produced only for the specified crystals
{
'00A' : ("TrappingFile Trapping.cal"),
}
)
```

    Check also that the file: *Trapping.cal* is written in the "ExtraFiles" list of the *gen_conf.py* :

```
ExtraFiles={
'CRYSTAL'    : "CrystalProducerATCA.conf PreprocessingFilterPSA.conf xinv_1325-1340.cal
     xdir_1325-1340.cal Trapping.cal",
}
```

2.4.2 Final energy re-calibrations (with offsets)

In all the previous steps, energy calibrations were always linear. But the PostPSA actor allows to apply a final segment and energy recalibration, including an offset and if necessary higher order of calibrations. This step needs thus to be done on data taken with an **^{152}Eu source** to have a good estimation of the non-linearity of the detectors. A very large statistics is thus needed for the backward segments. If you don't have such a dataset, use a ^{60}Co source,, by replacing in the following the "-152Eu" option by "-60Co".

0. The first step is to perform a replay of an ^{152}Eu run, from the traces, including the PostPSAFilter actor with neutron damage correction applied, and **without recalibration (RecalEnergy2) and ForceSegmentToCore** options.

   For an easier an faster work in the following, this replay can be done in two steps:

   (a) Replay from traces to PSA (without the post PSA) and store the *psa*.adf* files using the BasicAFC consumer:

   You need to update the symbolic link "Data" in order to link to the ^{152}Eu data folder.

```
$ mv Out Out_save             # If you want to save the last replay from 60Co
$ rm Data
$ ln -s /path/to/152Eu/run/Data Data
$ python gen_conf.py
```

   This kind if topology can be used:

```
LOOP CRY 00B 00C 01B 01C 04B 04C 05A 05B 05C 06A 06B 06C

Chain 4   CRY
Producer  CrystalProducerATCA
Filter    PreprocessingFilterPSA
Filter    PSAFilter
Consumer  BasicAFC
ENDLOOP
```

   A large statistics is better, this replay can be long, but it will allow to no more perform the PSA later. The next replays will then be very fast.

   (b) Replay of the PostPSA only, using as input the *psa*.adf* files that have been produced at the previous step. For this, you need to move the Out folder in which the *psa*.adf* have been written in a new folder ("Data_Local" for example), and update the symbolic link "Data" to this folder, in such a way that the input files of the replay will be the *psa*.adf* that have been just produced:

```
$ mv Out Data_Local
$ rm Data
$ ln -s Data_Local Data
$ python gen_conf.py
```

   This kind if topology can be used:

```
LOOP CRY 00B 00C 01B 01C 04B 04C 05A 05B 05C 06A 06B 06C

Chain 3   CRY
Producer  BasicAFP
Filter    PostPSAFilter
Consumer  None
ENDLOOP
```

   This will produce the *Post__5-40-16384-UI__Ener.spec* spectra filename

1. Segments recalibration:

- case 1 : for detectors with *Trapping.cal* files **present** (should be the standard procedure):

```
$ RecalEnergy -spe ../../Out/$CryId/Post__5-40-16384-UI__Ener.spec -sub 40 -num 36 -gain 4 \
  -poly1 -152Eu |tee recal.log
```

- case 2 : if for any reason the *Trapping.cal* file **is not present**:

```
$ RecalEnergy -spe ../../Out/$CryId/Post__5-40-16384-UI__Ener.spec -sub 0 -num 36 -gain 4 \
  -poly1 -152Eu |tee recal.log
```

  This difference is due to the fact that the segment energy corrected from the trapping file is filled in the spectra of the second library of the file ("-sub 40"). But if there is no neutron damage correction (new detectors), these spectra are empty. Spectra of the first library ("-sub 0") need thus to be taken for the re-calibration.

2. Extract the re-calibration parameters:

```
$ tail -n 36 recal.log |tee recal_nohead.log ;
$ awk -F' '  '{printf "segm %2.2s  %6.3f  %.6f \n",$1,$16,$17}' recal_nohead.log \
  |tee RecalEnergy2.cal
```

3. Adapt the *gen_conf.py* as follow:

```
#########################
PostPSAFilter=(
"ActualClass            PostPSAFilter",          # name of the used daughter class
"SaveDataDir            $SAVEDIR/$CRYSTAL",      # normally Out/Data(online)
"EnergyGain             4",                      # channels/keV of the calibrated energy spectra
#### command lines to be produced only for the specified crystals
{
'00A' : ("TrappingFile Trapping.cal", "RecalEnergy2 RecalEnergy2.cal"),
}
)
```

   Check also that the file: *RecalEnergy2.cal* is written in the "ExtraFiles" list of the *gen_conf.py* :

```
ExtraFiles={
'CRYSTAL'    : "CrystalProducerATCA.conf PreprocessingFilterPSA.conf xinv_1325-1340.cal
     xdir_1325-1340.cal Trapping.cal RecalEnergy2.cal",
}
```

4. Core recalibration:

- case 1 : for detectors with *Trapping.cal* files **present** (same as for segments):

```
$ RecalEnergy -spe ../../Out/$CryId/Post__5-40-16384-UI__Ener.spec -sub 79 -num 1 -gain 4 \
  -poly1 -152Eu| tee recalCore.log
```

- case 2 : for detectors **without** *Trapping.cal* files:

```
$ RecalEnergy -spe ../../Out/$CryId/Post__5-40-16384-UI__Ener.spec -sub 39 -num 1 -gain 4 \
  -poly1 -152Eu| tee recalCore.log
```

5. Extract the re-calibration parameters:

```
$ tail -n 1 recalCore.log |tee recalCore_nohead.log ;
$ awk -F' '  '{printf "\"RecalCC  %6.3f  %.6f\"\n",$16,$17}' recalCore_nohead.log
```

   The output of the last command should be equivalent to: "RecalCC -0.115 1.000401" and needs to be copied in the *gen_conf.py* as follow:

```
#########################
PostPSAFilter=(
"ActualClass           PostPSAFilter",          # name of the used daughter class
"SaveDataDir           $SAVEDIR/$CRYSTAL",      # normally Out/Data(online)
"EnergyGain            4",                      # channels/keV of the calibrated energy spectra
#### command lines to be produced only for the specified crystals
{
'00A' : ("RecalCC -0.094  1.000651","TrappingFile Trapping.cal", "RecalEnergy2 RecalEnergy2.cal"),
}
)
```

6. Check the results:

   To check that the recalibration has been well applied. A new replay is necessary with the modified *gen_conf.py* file. The file *Post__5-40-16384-UI__Ener.spec* can be analyzed with TkT. The different libraries are:

   1. segment energies without correction
   2. segment energies after trapping corrections
   3. fraction of the core energy without correction
   4. fraction of the core energy after trapping corrections
   5. **segment energies after trapping corrections and re-calibration.**

2.4.3 Force Segments to Core

The final step at the post PSA level is to decide of the use, or not, of the ForceSegmentsToCore option. If the core energy resolution is good and the counting rates not too important, it is better to use it. This option is defined for the desired detectors in the *gen_conf.py* , as follow:

```
#########################
PostPSAFilter=(
"ActualClass           PostPSAFilter",          # name of the used daughter class
"SaveDataDir           $SAVEDIR/$CRYSTAL",      # normally Out/Data(online)
"EnergyGain            4",                      # channels/keV of the calibrated energy spectra
#### command lines to be produced only for the specified crystals
{
'00A' : ("RecalCC -0.094  1.000651","TrappingFile Trapping.cal", "RecalEnergy2
    RecalEnergy2.cal","ForceSegmentsToCore"),
}
)
```

2.4.4 Global time alignments

The final time alignment of the cores is done using the output of the tracking. For this, this king of topology should be used:

```
LOOP CRY 00B 00C 01B 01C 04B 04C 05A 05B 05C 06A 06B 06C

Chain 3     CRY
Producer    BasicAFP
Filter      PostPSAFilter
Dispatcher  EventBuilder
ENDLOOP

Chain 3     Builder/
Builder     EventBuilder
Filter      TrackingFilterOFT
Consumer    BasicAFC
```

Then, it is necessary to define in the *gen_conf.py* file the mapping of the detectors in the TrackingFilter actor as follow:

```
########################
TrackingFilter=(
"ActualClass           PTrackingFilterOFT",     # name of the used daugther class
"SaveDataDir           $SAVEDIR/$MERGER",       # Out/Merger
"EnergyGain            4",                      # channels/keV of the calibrated energy spectra
"OftParams             0.05 0.02 0.8",          # minprobtrack minprobsing sigma_thet (0==default)
"SourcePosition        0 0 0",                  # source pos with respect to the center of AGATA
"NumGeDets  24"                                 # for printing of TkT spectra
"SpecMap  0  0",                                # detID --> specID
"SpecMap  1  1",
"SpecMap  2  2",
"SpecMap  6  3",
...
...
"SpecMap 39 21",
"SpecMap 40 22",
"SpecMap 41 23",
)
```

NOTE: 1: You need to be in the global folder (where Out, Data, Conf are located)
NOTE: 2: **Be careful, the ordering of the SpecMap is very important !**

1. Fit the time spectra:

   The tacking actor produces a spectra file named: *Track__N-N-1000-UI__TT.spec*, with N the number of detectors defined in the TrackingFilter part of the *gen_conf.py* file (in the following example, N=24). This file is used for the global time alignment of the cores. The RecalEnergy code is here again used:

   ```
   $ N=24; NN=$(( $N*$N ));
   $ RecalEnergy -spe Out/Global/Track__${N}-${N}-1000-UI__TT.spec -T 500 -num ${NN} |tee recalT.dat;
   $ tail -n ${NN} recalT.dat |tee recalT_nohead.dat;
   ```

   NOTE: If the EventMerger actor has been used in the replay, the file will be located in "Out/Merger/" instead of "Out/Global/"

2. Apply the *SolveTT.py* script (this script is located in the agapro/zUseful folder):

   ```
   $ solveTT.py -f recalT_nohead.dat -n ${N} -c 13 -p 500 |tee solveTT_tmp.dat
   ```

   The end of the output of this script should be similar to:

   ```
   Shifts that minimize Chi2
     -0.220
      3.565
      3.712
      1.187
      3.751
   ...
   ...
      3.505
      1.677
     -3.947
      1.248
     -0.931

   Initial:    Average of 506 nonzero values is -499.98900    Chi2 = 692.93223
   Corrected:  Average of 506 nonzero values is -499.98900    Chi2 = 17.45832
   ```

3. Extract the time shift values:

   ```
   $ tail -n $(( $N+3 )) solveTT_tmp.dat |tee solveTT.dat |head -n ${N} |tee solveTT.dat;
   $ awk -F' '  '{printf "\"TimeShiftCC  %7.3f \"\n",$1}' solveTT.dat
   ```

   The output of this last command give the time shifts to apply in the PostPSA actor for each crystal in the *gen_conf.py* file as follow:

```
#########################
PostPSAFilter=(
"ActualClass            PostPSAFilter",          # name of the used daughter class
"SaveDataDir            $SAVEDIR/$CRYSTAL",      # normally Out/Data(online)
"EnergyGain             4",                      # channels/keV of the calibrated energy spectra
#### command lines to be produced only for the specified crystals
{
'00A' : ("TimeShiftCC  -10.859" ,"RecalCC -0.094  1.000651","TrappingFile Trapping.cal",
     "RecalEnergy2 RecalEnergy2.cal","ForceSegmentsToCore"),
}
)
```

4. After a final global replay, check that the time alignment is correct. For this, open the spectra file *Out/Global/Track__$N-$N-1000-UI__TT.spec* (in the grid mode). The first set of N spectra will represent the timing between det 0 and the N-1 other... etc. All should be align at 500.

A List of commands

You can find bellow a list of commands for applying various steps of calibration for a list of detectors:

A.1 PreprocessingFilter

A.1.1 Energy calibration

```
#Define the crystal list
DET="00A 00B 00C 01A 01B 01C 02A 02B 02C"

### Commands to be applied in the replay folder:
### ==> where Conf, Data and Out dir, and gen_conf.py file are located

## Define Data dir (Out or Data as a function of spectra produced online or from replay)
DataDir=Data

## Define the source (-60Co or -152Eu) ==> 60Co is better for preprocessing level
Source=-60Co

for i in $DET; do cd Conf/$i;
RecalEnergy -spe ../../$DataDir/$i/Prod__4-38-32768-UI__Ampli.spec -sub 38 -num 38 -gain 2 \
-dwa 15 5 $Source |tee recal.out;
echo "ECalib for DET $i: press a key to save and continue"; read;
cp PreprocessingFilterPSA.conf PreprocessingFilterPSA.save_energy # Just in case...
colupdate.py PreprocessingFilterPSA.conf recal.out -c 4 13 -o PreprocessingFilterPSA.conf
cd ../..; done;
```

A.1.2 Time alignment

```
#Define the crystal list
DET="00A 00B 00C 01A 01B 01C 02A 02B 02C"

### Commands to be applied in the replay folder:
### ==> where Conf, Data and Out dir, and gen_conf.py file are located

## Define Data dir (Out or Data as a function of spectra produced online or from replay)
DataDir=Data

## Time alignment of the segments
for i in $DET;do cd Conf/$i;
RecalEnergy -spe ../../$DataDir/$i/Prep__6-40-1000-UI__TT.spec -sub 40 -num 36 -T 500 |tee shift_TT.out;
echo "ECalib for DET $i: press a key to save and continue"; read;
cp PreprocessingFilterPSA.conf PreprocessingFilterPSA.save_time # Just in case...
colupdate.py PreprocessingFilterPSA.conf shift_TT.out -c 6 13 -o PreprocessingFilterPSA.conf;
cd ../../; done;

# For the time alignment of the cores, see the graphical explanations from the documentation
```

A.2 PostPSAFilter

A.2.1 Neutron damage correction:

```
#Define the crystal list
DET="00A 00B 00C 01A 01B 01C 02A 02B 02C"

### Commands to be applied in the replay folder:
### ==> where Conf, Data and Out dir, and gen_conf.py file are located

## Define Data dir (Out or Data as a function of spectra produced online or from replay)
DataDir=Out

## Create dummy empty file
for i in $DET; do cd Conf/$i;
rm -f Trapping.cal
for i in {0..35} ; do echo -e "$i\t1.\t1.\t999999.9\t999999.9\t1.\t1." >> Trapping.cal; done
cat Trapping.cal; cd ../.. ; done;

## Determine neutron damage parameters
for i in $DET; do cd Conf/$i;
SortPsaHits -f ../../$DataDir/$i/Psa__0-16-F__Hits.fdat -best 1300 1350 -bpar 1 10000 0 \
|tee sort_hit.log
tail -n 39 sort_hit.log |head -n 36 > sort_hit_nohead.log;
awk 'FNR==NR{a[NR]=$4;next}{$4=a[FNR]}1' sort_hit_nohead.log Trapping.cal \
|awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' > Trapping_tmp.cal;
awk 'FNR==NR{a[NR]=$5;next}{$5=a[FNR]}1' sort_hit_nohead.log Trapping_tmp.cal \
|awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' |tee Trapping.cal;
cd ../.. ; done;

## Generate output spectra
for i in $DET;do cd Conf/$i;
SortPsaHits -f ../../$DataDir/$i/Psa__0-16-F__Hits.fdat -gain 5 -offs 5000 -fcal Trapping.cal
cd ../..;done;

## Recalibration BEFORE neutron damage correction
## ==> Not necessary if no REALLY bad calibration (more than 15keV shifted)
for i in $DET;do cd Conf/$i;
#segments
RecalEnergy -spe Pso__2-4-40-2048-UI__Ener.spec -num 36 -sub 0 -gain 5 -offs -5000 -noTR -dwa 30 2 \
|tail -n 36 |tee log_sg_pre.cal
#cores
RecalEnergy -spe Pso__2-4-40-2048-UI__Ener.spec -num 36 -sub 160 -gain 5 -offs -5000 -noTR -dwa 30 2 \
|tail -n 36 |tee log_cc_pre.cal
#update results
awk 'FNR==NR{a[NR]=$14;next}{$2=a[FNR]}1' log_sg_post.cal Trapping.cal \
|awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' |tee Trapping_tmp.cal
awk 'FNR==NR{a[NR]=$14;next}{$3=a[FNR]}1' log_cc_post.cal Trapping_tmp.cal \
|awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' |tee Trapping.cal
cd ../..;done;

# re-generate output spectra
for i in $DET;do cd Conf/$i;
SortPsaHits -f ../../Data/$i/Psa__0-16-F__Hits.fdat -gain 5 -offs 5000 -fcal Trapping.cal
cd ../..;done;

## Recalibration AFTER neutron damage correction
## ==> Necessary for damaged detectors
for i in $DET;do cd Conf/$i;
#segments
RecalEnergy -spe Pso__2-4-40-2048-UI__Ener.spec -num 36 -sub 80 -gain 5 -offs -5000 -dwa 20 2 \
|tail -n 36 |tee log_sg_post.cal
#cores
RecalEnergy -spe Pso__2-4-40-2048-UI__Ener.spec -num 36 -sub 240 -gain 5 -offs -5000 -dwa 20 2 \
| tail -n 36 |tee log_cc_post.cal
read
#update results
awk 'FNR==NR{a[NR]=$14;next}{$6=a[FNR]}1' log_sg_post.cal Trapping.cal \
|awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' |tee Trapping_tmp.cal
awk 'FNR==NR{a[NR]=$14;next}{$7=a[FNR]}1' log_cc_post.cal Trapping_tmp.cal \
|awk '{printf "%2s %10s %10s %10s %10s %10s %10s \n", $1,$2,$3,$4,$5,$6,$7}' |tee Trapping.cal
cd ../..;done;

# re-generate output spectra
for i in $DET;do cd Conf/$i;
SortPsaHits -f ../../Data/$i/Psa__0-16-F__Hits.fdat -gain 5 -offs 5000 -fcal Trapping.cal
cd ../..;done;
```

```
### Check calibs
### This script allows to check the full Trapping files,
### to help to spot potential errors in the calibration procedure

for i in $DET;do cd Conf/$i;
echo -e "\n\033[36mDet: $i\n\033[0m"
for line in {1..36} ; do
toto=`head -$line Trapping.cal |tail -1`;
id=`echo $toto |awk -F " " '{print $1}'`;
a=`echo $toto |awk -F " " '{print $2}'`;
diff=`printf "%.6f\n" $(echo "sqrt(($a - 1.)^2)" | bc -l)`
if ((`bc <<< "$diff<0.001"`)); then color="\033[32m";
elif ((`bc <<< "$diff<0.01"`)); then color="\033[33m";
else color="\033[31m";fi;a="$color$a\033[0m";
b=`echo $toto |awk -F " " '{print $3}'`;
diff=`printf "%.6f\n" $(echo "sqrt(($b - 1.)^2)" | bc -l)`
if ((`bc <<< "$diff<0.001"`)); then color="\033[32m";
elif ((`bc <<< "$diff<0.01"`)); then color="\033[33m";
else color="\033[31m";fi;b="$color$b\033[0m";
n1=`printf "%9.1f\n" $(echo $toto |awk -F " " '{print $4}')`;
n2=`printf "%9.1f\n" $(echo $toto |awk -F " " '{print $5}')`;
c=`echo $toto |awk -F " " '{print $6}'`;
diff=`printf "%.6f\n" $(echo "sqrt(($c - 1.)^2)" | bc -l)`
if ((`bc <<< "$diff<0.001"`)); then color="\033[32m";
elif ((`bc <<< "$diff<0.01"`)); then color="\033[33m";
else color="\033[31m";fi;c="$color$c\033[0m";
d=`echo $toto |awk -F " " '{print $7}'`;
diff=`printf "%.6f\n" $(echo "sqrt(($d - 1.)^2)" | bc -l)`
if ((`bc <<< "$diff<0.001"`)); then color="\033[32m";
elif ((`bc <<< "$diff<0.01"`)); then color="\033[33m";
else color="\033[31m";fi;d="$color$d\033[0m";
#echo -e "$a"
echo -e "$id\t$a\t$b\t$n1\t$n2\t$c\t$d"
done ;
cd ../..;done;
```

A.2.2 Final energy recalibration (RecalEnergy2)

```
#Define the crystal list
DET="00A 00B 00C 01A 01B 01C 02A 02B 02C"

### Commands to be applied in the replay folder:
### ==> where Conf, Data and Out dir, and gen_conf.py file are located

## Define Data dir (Out or Data as a function of spectra produced online or from replay)
DataDir=Out

## Define the source (-60Co or -152Eu) ==> 152Eu is better for final recalibrations
Source=-152Eu

## Segments recalibration (if no trapping file, see the documentation):
for i in $DET;do cd Conf/$i;
RecalEnergy -spe ../../$DataDir/$i/Post__5-40-16384-UI__Ener.spec -sub 40 -num 36 -gain 4 -poly1 \
-dwa 30 2 $Source |tee recal.log
echo "ECalib for DET $i: press a key to continue"; read;
tail -n 36 recal.log |tee recal_nohead.log ;
awk -F' '  '{printf "segm %2.2s %6.3f %.6f \n",$1,$16,$17}' recal_nohead.log |tee RecalEnergy2.cal
cd ../..;done;

## core recalibration (if no trapping file, see the documentation):
rm -f recalCore.log
for i in $DET;do cd Conf/$i;
RecalEnergy -spe ../../$DataDir/$i/Post__5-40-16384-UI__Ener.spec -sub 79 -num 1 -gain 4 -poly1 \
-dwa 40 5 $Source |tee recalCore.log;
tail -n 1 recalCore.log > recalCore_nohead.log ;
cd ../..;
awk -v var=$i -F' '  '{printf "%s: \"RecalCC %6.3f %.6f\"\n",var,$16,$17}' \
Conf/$i/recalCore_nohead.log >> recalCore.log ;
done; cat recalCore.log ;

## Quick Check of the FWHM:
for i in $DET;do cd Conf/$i;
tail -n 1 recalCore.log > recalCore_nohead.log ;
awk -v var=$i -F' '  '{printf "%s: \"FWHM %6.3f\"\n",var,$6}' recalCore_nohead.log
cd ../..;done;
```

```
### Check calibs
### This script allows to check the full Trapping files,
### to help to spot potential errors in the calibration procedure

for i in $DET;do cd Conf/$i;
echo -e "\n\033[36mDet: $i\n\033[0m"
for line in {1..36} ; do
toto=`head -$line RecalEnergy2.cal |tail -1`;
id=`echo $toto |awk -F " " '{print $2}'`;
offset=`echo $toto |awk -F " " '{print $3}'`;
offsetabs=$(echo "sqrt(($offset)^2)" | bc -l)
if ((`bc <<< "$offsetabs<1"`)); then color="\033[32m";
elif ((`bc <<< "$offsetabs<5"`)); then color="\033[33m";
else color="\033[31m";fi;offset="$color$offset\033[0m";
gain=`echo $toto |awk -F " " '{print $4}'`;
diff=`printf "%.6f\n" $(echo "sqrt(($gain - 1.)^2)" | bc -l)`
if ((`bc <<< "$diff<0.001"`)); then color="\033[32m";
elif ((`bc <<< "$diff<0.01"`)); then color="\033[33m";
else color="\033[31m";fi;gain="$color$gain\033[0m";
echo -e "segm $id\t$offset\t$gain"
done ;
cd ../..;done
```

A.2.3 Global time alignment

```
## N define the number of detectors:
## ==> must be identical as the value defined in the Tracking actor of the gen_conf.py
N=38;

## Define Data dir (Out/Merger or Out/Global as a function of the topology used)
DataDir=Out/Merger

NN=$(( $N*$N )); \
RecalEnergy -spe $DataDir/Track__${N}-${N}-1000-UI__TT.spec -T 500 -num ${NN} |tee recalT.dat; \
tail -n ${NN} recalT.dat |tee recalT_nohead.dat; \
solveTT.py -f recalT_nohead.dat -n ${N} -c 13 -p 500 |tee solveTT_tmp.dat; \
tail -n $(( $N+3 )) solveTT_tmp.dat |tee solveTT.dat |head -n ${N} |tee solveTT.dat; \
awk -F' ' '{printf "\"TimeShiftCC %7.3f \"\n",$1}' solveTT.dat;
```