PET UJ Report nr 1/2018

J-PET Data Analysis with Framework software version 6.1

Magdalena Skurzok, Michał Silarski, Krzysztof Kacprzak

Instytut Fizyki, Uniwersytet Jagiellonski, Poland

e-mail: `michal.silarski@uj.edu.pl`, `magdalena.skurzok@uj.edu.pl`, `k.kacprzak@student.uj.edu.pl`

This report includes useful information concerning installation, basic steps of analysis and GitHub repository, and its main purpose is to guide fresh Framework users through the first steps of software usage and give basis for writing independent analysis of J-PET experimental data. Updated for Framework version 6.1, January 2018.

Contents

1. Framework data structure

Before getting to use the Framework software, it is a good idea to review (or learn from the basis) about the J-PET experiment, since there are some **JPet* Classes** in Framework, that represent elements, that detector consists of and physical phenomenons, that take place during the measurements.

1.1 J-PET Detector

The J-PET Detector consists of slots (two photomultipliers, one scintilator) arranged in layers (3 at the moment), as shown in Fig. 1. During data analysis with Framework, you will use C++ classes, which description follows.

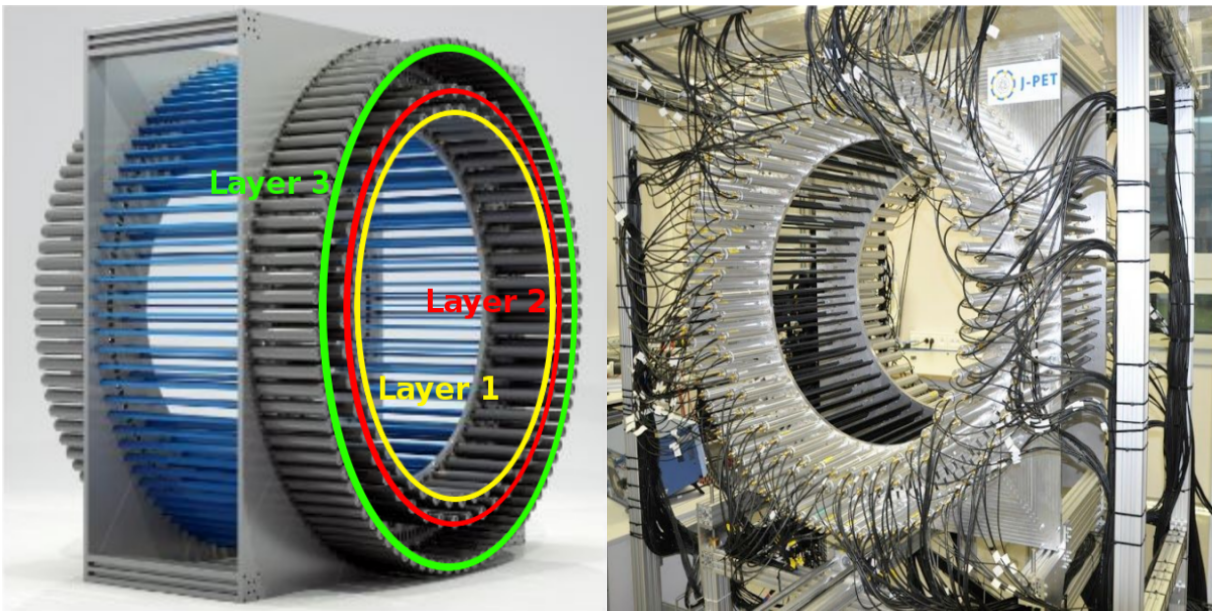

Figure 1: J-PET detector scheme with marked layers (left panel) and photo of J-PET detector setup in laboratory (right panel).

1.2 JPetLayer

The representation of a Layer consisting of Barrel Slots. The numbering of the Layers start from 1 and is ordered from the one with the shortest radius to the one with the longest (see left panel of Fig. 1). The properties of each JPetLayer Object are: **ID, Name, Radius, Active/inactive status**. In the current setup **Layer 1 and Layer 2** consist of **48 slots** and **Layer 3** consists of **96 slots**.

1.3 JPetBarrelSlot

The representation of Slot, that consists of Scintillator with Photomultipliers attached to the ends. JPetBarrelSlot obejcts contain information about: **ID (in Layer), ID in Frame (general numbering), Name, Active/inactive status, Angle (Theta)** that describes position in Layer. Along with the information about Layer radius and hit position, one can calculate i.e. position of hit in respect to detector center (0,0,0).

1.4 JPetScin

The representation of Scintillators, that register energy deposition of photons coming from the source (whatever source it is). Subpage on PetWiki holding details about Scintilators:
`http://koza.if.uj.edu.pl/petwiki/index.php/List_of_scintillators_for_big_barrel`.
Each JPetScin object holds information about: **ID, Sizes (dimensions), Barrel Slot it belongs to**.

1.5 JPetPM

The representation of Photomultipliers, that measure Signals arriving from Scintillators.
`http://koza.if.uj.edu.pl/petwiki/index.php/List_of_photomultipliers_for_big_barrel`.
Each JPetPM object has properties such as: **ID, Side (A or B)** - info that can be easily used to distinguish PM to pair them, **High Voltage gains, settings and options, FEB (Front End Board) it is connected to, Barrel Slot it belongs to**.

1.6 Remarks so far

While using Framework there is possibility to easily obtain information about connections between objects, by using 'getters' functions. So for example if in code we have available an object of JPetPM class, we are able to:

- get radius of Layer that this PM belongs to
  `float radius = pm.getBarrelSlot().getLayer().getRadius();`
- get the ID of Scintillator, that PM is connected to
  `int scinID = pm.getScin().getID();`
- check whether PM is connected to active FEB
  `bool isFEBactive = pm.getFEB().isActive();`

Next we are describing classes, that represent physical phenomenons, that take place during the measurement.

1.7 JPetEvent

In simplification, J-PET Detector registers interactions of photons with scintillating material. Those photons originate from some physical phenomenon, lets call it an Event. The source of this Event can be different - whether it is decaying ortho-positronium or something less exciting. One can illustrate an example Event as such (Fig. 2):
In first case, our Event is marked in that picture with number 2. We assume that whatever it is, it emits photons - in illustrated example three photons marked with solid line arrows originating from point number 2. This phenomenon is represented as JPetEvent object, that holds information about: `Hits` that construct this Event, `Type` - so was it an Event with 3 photons? Maybe with 2? Or one, that can be described as photon from deexcitation? The types of events can be various and will probably be changed in the future to describe accurately the gathered experimental data.

1.8 JPetHit

It is basically the phenomenon of photon from the previous subsection depositing energy in some Scintillator. The deposition is marked with yellow circle in the scheme presented in Fig. 3. After the particle hit, energy is deposited in the Scintillator and it propagates in every direction - also towards the Photomultipliers (marked with arrows). Every JPetHit object contains information about: **Arrival Time [ps], Energy, Position in Scintillator (X,Y,Z), Two Signals** that construct the hit, **Time difference** between arrival times of two Physical Signals.

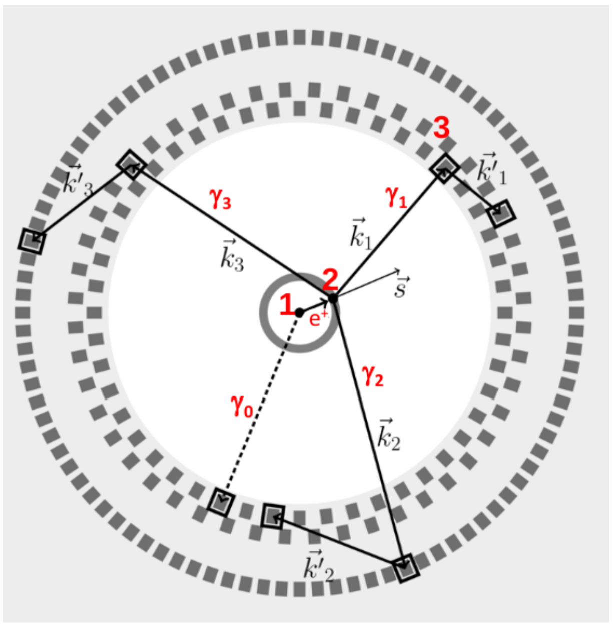

Figure 2: Schematic view of the cross section of the J-PET detector with marked ortho-Positronium decay. Ortho-Positronium decay point is marked with number 2, while outgoing photons with γ_1, γ_2 and γ_3.

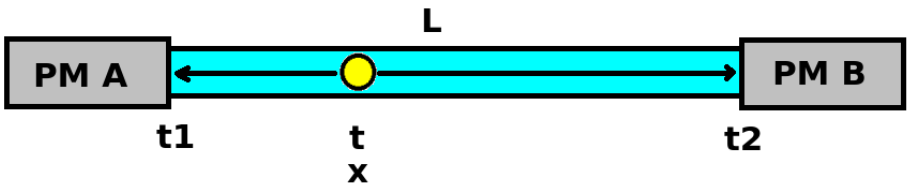

Figure 3: Schematic view of one scintillator strip with two photomultipliers on both sides. Yellow circle denotes hit of photon.

1.9 JPetPhysSig

The representation of reconstructed Signal, that appear after its arrival into the Photomultiplier. It is measured by PM, that gives information about: **Arrival Time [ps], Photoelectrons**. It is not settled up to now, what is understood as arrival time of the reconstructed signal - it is up to the user to decide. JPetPhysSig is marked with dense-pointed curve in Fig. 4.

1.10 JPetRawSig

The collection of 1-8 points, that are representation of Signal Channels. It is a group of 8 points - 4 red and 4 green shown in Fig. 4. Each JPetRawSig object hold info about: **Number of points** that construct it, **Signal Channel Points** with division to Leading Edge and Trailing Edge points.

1.11 JPetSigCh

The representation of single Signal, that was registered on PM channel. It can be on one of 4 thresholds and be type of Leading Edge or Trailing Edge (that information is provided by electronics boards). Such points are represented in Fig. 4 with red and green points, thresholds with dashed lines, and the reconstructed Physical Signal is a dense-pointed curve.

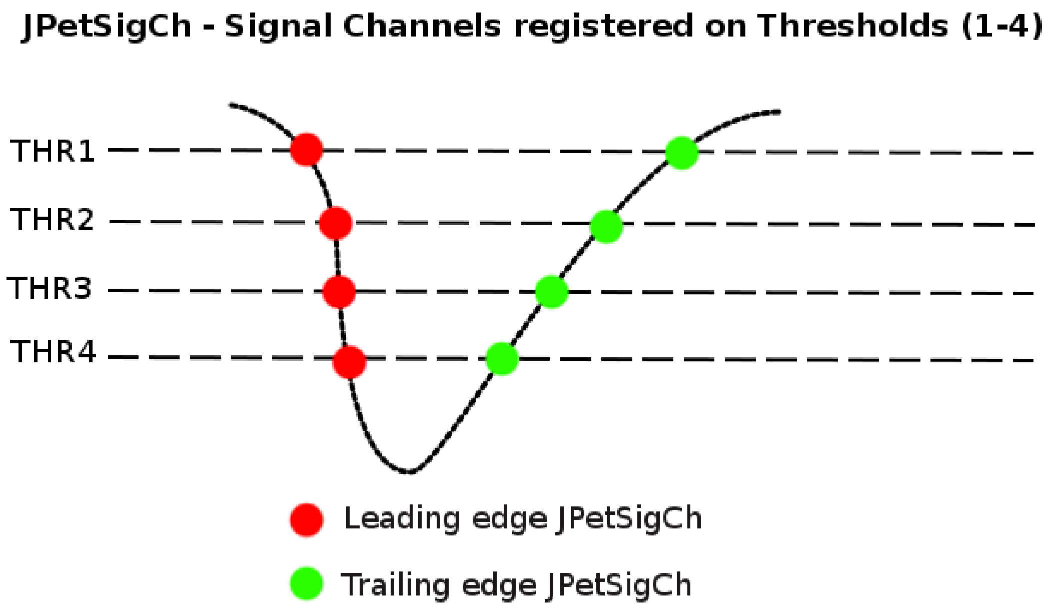

Figure 4: Schematic view of reconstructed Physical Signal (dense-pointed curve). 8 points denote representation of Signal Channel - 4 thresholds and type of Leading Edge and Trailing Edge (4 red and 4 green points, respectively).

Each JPetSigCh object has information about: **arrival time, Threshold number and value, Photomultiplier** it belongs to, **FEB and TRB** it belongs to and type of `Edge (Leading or Trailing)`.

1.12 JPetTimeWindow

The measurement is conducted in real time, and consists of sequential periods, that are called Time Windows. For example in Run 1 the Time Window was equal to 666,7 microseconds. There is very little sense in analysing sets of JPetSigCh form different Time Windows, that are represented by JPetTimeWindow Class. Each JPetTimeWindow is a collection of JPetSigCh Objects, that come from all PM on all Barrel Slots on all Layers.

1.13 Final remarks

When performing analysis, one has to perform tasks, that construct objects from the simplest (JPetSigCh) to the most complex (JPetEvent or even JPetLOR, not described here). The aim of the work of the whole J-PET group is to work out examples and methods, that will allow this construction in common way, with the conservation of order of creating the objects - which is presented in Fig. 5. It is the reverse order comparing to this presented in this Report.

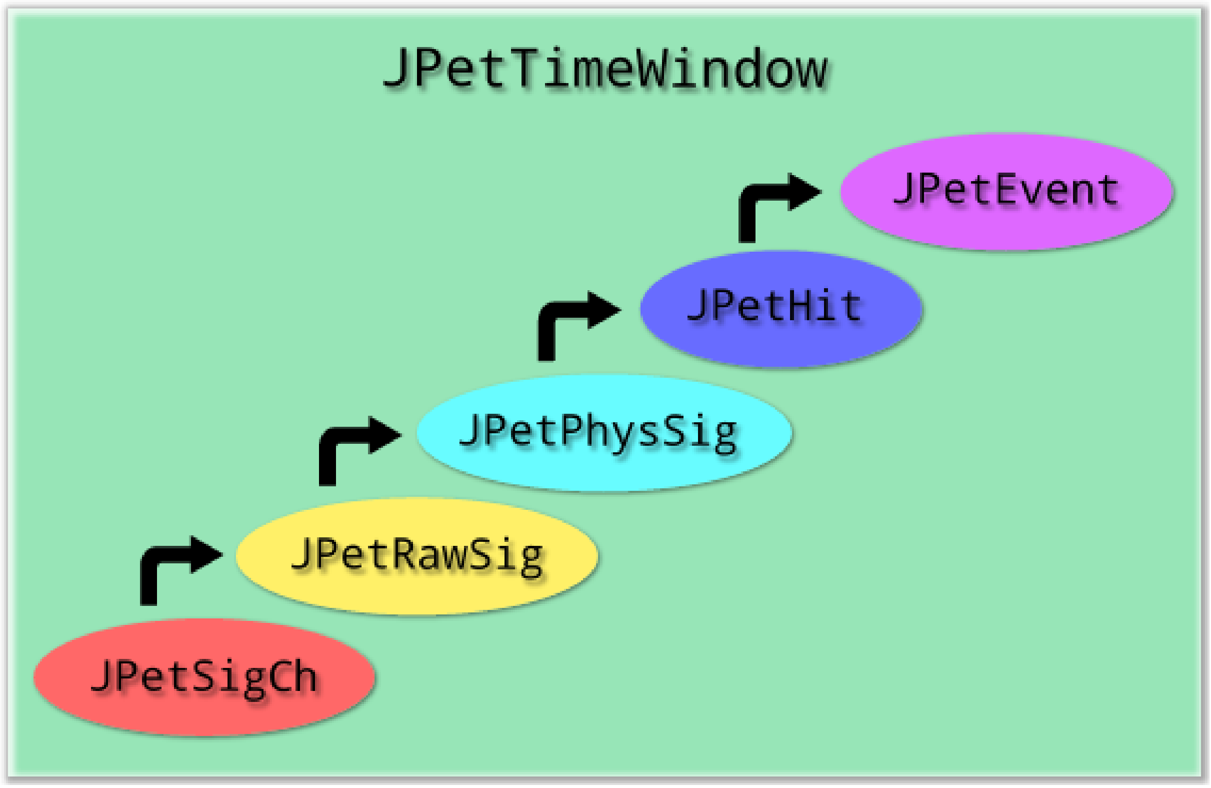

Figure 5: J-PET objects within one Time Window.

2. Framework installation

Installation process of Framework is proven to be challenging for first-time users, but do not give up! All needed information is posted on PetWiki `http://koza.if.uj.edu.pl/petwiki/index.php/Installing_the_J-PET_Framework_on_Ubuntu`. The best way is to follow the instructions step by step and stay optimistic. In case of problems, you can always ask for help fellow Framework ~~victim~~ user or post a support request on Redmine forum `http://sphinx.if.uj.edu.pl/redmine/projects/j-pet-framework/boards`.

2.1 Requirements

J-PET Framework can be installed at your own computer or at J-PET server. Basically, for installation one need:

- Linux Ubuntu operating system - recommended versions 14.04 and 16.04
- g++ compiler - required version 4.9
- ROOT Data Analysis Framework - required version 5 (recommended 5.34.26) (`https://root.cern.ch/`)

Always refer to INSTALL file, to check required libraries. The installation is also shown in tutorial movie
`http://koza.if.uj.edu.pl/~alek/workshop/jpet_framework_tutorial_screencast.m4v`

2.2 Using j-pet-server

j-pet-server is equipped with all required software for individual Framework installation. You need to create user account (contact Eryk Czerwiński). To logon to server from outside the Institute, you need a personal VPN access, ask your supervisor how to obtain it from Departments IT staff.

- Install `openvpn` on Ubuntu (or or any other Debian):

  ```
  sudo apt-get install openvpn
  ```

- With your private key, use software to connect to VPN:

  ```
  sudo openvpn <yourName>.conf
  ```

  (`<yourName>.conf` is the name of your private configuration file)

- provide you `sudo` password and you private VPN key

- after successful connection, in another terminal you can connect with j-pet-server with `ssh`

  ```
  ssh yourUserName@serverIP
  ```

2.3 GitHub repositories

J-PET Framework software is maintained within public repositories of GitHub:

- Core: `https://github.com/JPETTomography/j-pet-framework`
- Examples: `https://github.com/JPETTomography/j-pet-framework-examples`

The most useful way to obtain Framework and Examples code, is with Git version control system. If you are not acquainted with Git, there are many useful tutorials for beginner users of version control systems, i.e. `http://rogerdudler.github.io/git-guide/`. To obtain Examples and Framework at the same time, just type in terminal the command:

```
git clone --recursive https://github.com/JPETTomography/j-pet-framework-examples.git
```

3. What and where and maybe... how?

3.1 Installation result

After cloning with Git and installing, J-PET Framework Examples are located in folder **j-pet-framework-examples**. This folder contains, among others:

- **examples**:

  `LargeBarrelAnalysis`, which is developed to become an official reconstruction procedure for all J-PET data. It includes modules (Tasks) dedicated for analysis of data gathered with large barrel detector. Description of modules in Sec. 4.3.

  NewAnalysisTemplate - place to start your analysis. This example links modules from `LargeBarrelAnalysis`, so goal for user is to add new Tasks, that analyse data further, after completion of all previous, basic procedures.

  TimeCalibration and VelocityCalibration - examples, that are used for obtaining calibrations from specific measurements.

- **j-pet-framework** directory - core of Framework software.

- **scripts** - here you can find useful ROOT scripts, that load Framework shared-object library at starting ROOT software (i.e. `rootlogon.C`).
- **DOXYGEN documentation**, if generated, can be found in build folder in `html` or `latex` directories. Method of generating the documentation is described in Sec. 3.2,

3.2 Documentation

The code documentation can be generated in **build** directory with Doxygen package with commend:

```
make documentation
```

or inside j-pet-framework directory (where `Doxyfile` is located)

```
doxygen
```

Look for `index.html` file, that is available in the `j-pet-framework/html/` directory.

4. Analysis step by step

This section includes useful information concerning the J-PET data analysis. The steps of analysis are presented based on `LargeBarrelAnalysis` example.

4.1 Obtaining data files

The data from measurements with J-PET detector is recorded on servers and transported to tapes eventually for storage. Information about types of files and their location can be found on PetWiki in **Documents/Reports** section and in **Archived Data** section, in case if the files were moved to tapes. The right person to ask for data access is Eryk Czerwiński, and other J-PET collaborators for sure store single files in personal workspaces. Raw data files are in `hld` format (unpacked files) and have usually size of 2 GB. The file name only will not provide information about type and purpose of measurement, so it is up to the user to get the knowledge what is she/he about to analyse with Framework software. There are many types of measurements done during the Runs, i.e. **calibration runs - with collimator or reference detector, runs with bare source, runs with annihilation chamber of different sizes**.

4.2 Calibration files

During the execution of Framework tasks, the user need to provide proper calibration files, that comply with the data file. Refer to PetWiki `http://koza.if.uj.edu.pl/petwiki/index.php/Default_settings_and_parameters_used_in_the_analyses` to know, which calibration file to use for specific Run data file. It is possible to use improper calibration for certain data file, so always check if you are performing reasonable things. The files themselves can be downloaded from mentioned website, but are also downloaded during building of Framework Examples from `sphinx` server (at least some of them, that are crucial).

Currently, types of calibrations/configuration are following:

- necessary - without them program will not be performed:
  - configuration `XML` for Unpacker module
  - detector configuration `JSON`
- optional, but without using or misusing those files prevents the obtained results to be reasonable:
  - signal time calibration constants [ASCII]
  - correction for trailing edge channel times OR TOT correction OR stretcher [ROOT]
  - effective velocity of light in scintillators [ASCII]
  - threshold values - for now not used, so can be abandoned [ASCII]

4.3 Analysis modules

Each directory with Framework analysis contains modules responsible for proper tasks and `main.cpp` file with included modules. Every Framework module consists of `init()`, `exec()` and `terminate()` methods. The `init()` and `terminate()` method are executed only once per module run, obviously at the beginning and the end respectively. One can i.e. make there histograms declaration, which next are filled in `exec()` method, that runs separately for each Time Slot within the data file. Basically the analysis modules correspond to proper methods, that are transforming simpler data structure to a more complex one, as it was show in Figure 5. The program starts with `hld` and transforms it to sequence of `ROOT` files with different extensions. The output from one module is an input for the next.

4.3.1 `LargeBarrelAnalysis` modules

The modules of considered `LargeBarrelAnalysis` example (located in directory: `j-pet-framework-examples/LargeBarrelAnalysis`) are following:

- Unpacker - reading `hld` file and making it usable for `ROOT` format, uses configuration `XML` and setup `JSON`
- TimeWindowCreator - process unpacked hld file into a tree of JPetTimeWindow objects (output file `filename.tslot.calib.root`). Uses time calibration constants and threshold values files, provided by user.
- SignalFinder - creates Raw Signals (output file `filename.raw.sig.root`)
- SignalTransformer - creates Reco & Phys Signals (output file `filename.phys.sig.root`)
- HitFinder - creates hits from physical signals (output file `filename.hits.root`). Users effective velocity of light file, provided by user.
- EventFinder - creates Events as group of Hits (output file `filename.unk.evt.root`)
- EventCategorizer - categorizes Events (output file `filename.cat.evt.root`)

4.4 Analysis Run

In case of `LargeBarrelAnalysis` example described in previous Section, you can run analysis in directory `build/LargeBarrelAnalysis` in a manner illustrated by an example:

```
   ./LargeBarrelAnalysis.x -t hld -f dabc_16218140613.hld -l detectorSetup.json -i
44 -c TOTConfig.root -p conf_trb3.xml -o outputDir/ -u userParams.json -r 0 100000
```

General way of executing the program with options is:

```
   ./<program_name>.x -t <opt_t> -f <opt_f> -l <opt_l> -i <opt_i> -c <opt_c> -p <opt_p>
-o <opt_o> -u <opt_u> -r <opt_r>
```

where: `t, f, i, l, p, c, o, u, r` are options corresponding to:

- `opt_t` - data file format (usually `hld` or `root`)
- `opt_f` - path to data file `path/filename` with the same extension as declared with `-t` option
- `opt_l` - `JSON` file with detector setup
- `opt_i` - run number, but in practice a *magic number* that is a first key in the used `JSON` file
- `opt_c` - `ROOT` file with TOT corrections
- `opt_p` - `XML` file with configuration for Unpacker task
- `opt_o` - path to folder, where you want the resulting files to be written. If not specified, the output files will be created in the same directory as input file
- `opt_u` - `JSON` file with user options. This is the best way to declare parameters used by different modules, without the need of recompiling whole code. The parameters, that can be provided depend on version of modules, but generally follow such template:
  ```
  {
  "TimeCalibLoader_ConfigFile_string":"dummyTimeCalib.txt",
  "ThresholdLoader_ConfigFile_string":"dummyThresholds.txt",
  "SignalFinder_EdgeMaxTime_float":"20000",
  "SignalFinder_LeadTrailMaxTime_float":"300000",
  "HitFinder_TimeWindowWidth_float":"25000",
  "EventFinder_EventTime_float":"5000"
  }
  ```
- `opt_r` - range of analysed events, two number denoting entry index - begin and end (i.e. `1230 2340`). Option useful for quick tests, when it is enough to limit run to analysis of several Time Slots.

4.5 Place to start - `NewAnalysisTemplate`

This section presents simple example of analysis run. Similar examples are also shown in tutorial movie *jpet_framework_tutorial_screencast.m4v* available at:
`http://koza.if.uj.edu.pl/~alek/workshop/`.

Let's start!
First enter directory `j-pet-framework-examples/NewAnalysisTemplate`. There you can find basic files: `CMakeList.txt`, `README` and `main.cpp`. Modify `main.cpp` to fit your needs: include or exclude modules linked from `LargeBarrelAnalysis` i.e. by commenting them out or removing; add your custom task.

Listing 1: main.cpp

```
#include <JPetManager/JPetManager.h>
#include "../LargeBarrelAnalysis/TimeWindowCreator.h"
//#include "../LargeBarrelAnalysis/SignalFinder.h"
#include "MyCustomTask.h"
using namespace std;

int main(int argc, const char* argv[]) {
  JPetManager& manager = JPetManager::getManager();
  manager.registerTask<TimeWindowCreator>("TimeWindowCreator");
  //manager.registerTask<SignalFinder>("SignalFinder");
  manager.registerTask<MyCustomTask>("MyCustomTask");

  manager.useTask("TimeWindowCreator", "hld", "tslot.calib");
  //manager.useTask("SignalFinder", "tslot.calib", "raw.sig");
  manager.useTask("MyCustomTask", "tslot.calib", "my.sig");

  manager.run(argc, argv);
}
```

Class `MyCustomTask` should extend `JPetUserTask`, in this example it takes as an input the result of task `TimeWindowCreator`. If needed, another folder with separate analysis can be added. If your directory structure looks like this:

- j-pet-framework-examples
  build/
  LargeBarrelAnalysis/
  NewAnalysisTemplate/
  j-pet-framework/
  CMakeLists.txt
  ...

then add new folder (i.e. `MyFirstAnalysis`) inside `j-pet-framework-examples` with proper `main.cpp` file and append the `CMakeLists.txt` in the top directory with

```
add_subdirectory(MyFirstAnalysis)
```

To compile, go to `build/` directory and run

```
cmake ..

make
```

All the examples, including yours, shall be built. The executable file is in
`j-pet-framework-examples/build/MyFirstAnalysis`
directory, and it is possible to run it as it was demonstrated above (Sec. 4.4)
As a result of the analysis we obtain output files with names corresponding to original file name, with extensions appropriate to module definition. So each task - one output file. The result of a `LargeBarrelAnalysis` program performed on a file named i.e. `dabc_16218140613.hld` shall be:

```
dabc_16218140613.tslot.calib.root
dabc_16218140613.raw.sig.root
dabc_16218140613.phys.sig.root
dabc_16218140613.hits.root
dabc_16218140613.unk.evt.root
dabc_16218140613.cat.evt.root
```

5. Contact persons

In case of problems with Framework (installation, analysis, etc.) one can contact following experts:

- **Eryk Czerwiński** eryk.czerwinski@uj.edu.pl - J-PET servers and data management
- **Aleksander Gajos** alek.gajos@gmail.com - J-PET Framework development
- **Wojciech Krzemień** wojciech.krzemien@ncbj.gov.pl - J-PET Framework development and design
- **Szymon Niedźwiecki** wictusn@gmail.com - general and detailed knowledge of J-PET experiment
- **Marcin Zieliński** m.zielinski83@gmail.com - J-PET wiki pages
- **Magdalena Skurzok, Michał Silarski** magdalena.skurzok@uj.edu.pl, michal.silarski@uj.edu.pl - detector calibration procedures
- **Krzysztof Kacprzak** k.kacprzak@student.uj.edu.pl - development of J-PET Framework examples