

J-PET 2019

**J-PET Data Analysis with Framework
software version 8.0**

Krzysztof Kacprzak

Instytut Fizyki, Uniwersytet Jagiellonski, Poland

e-mail: k.kacprzak@alumni.uj.edu.pl



This guide is an introduction to Framework software prepared for data analysis in J-PET experiment. It contains description of installation, software design, steps of creating a new analysis. The main purpose of this paper is to guide fresh Framework users through the first steps and provide reference source. Manual prepared for 8th version of Framework, October 2019.

Previous version of this Guide by Magdalena Skurzok, Michał Silarski and Krzysztof Kacprzak can be found on PetWiki:

- Framework version 5.0
- Framework verison 6.1
- Framework verison 7.0

Contents

1	Introduction	4
2	Structure of Framework	5
2.1	Detector objects	5
2.2	Data objects	6
2.3	Units used in reconstruction procedures	9
3	Framework installation	10
3.1	Requirements	10
3.2	Installing Unpacker2	11
3.3	Installing Framework	11
3.4	Building example procedures	11
3.5	Using j-pet-servers	12
3.6	GitHub repositories	12
4	What and where and maybe... how?	13
4.1	Outcome of the build	13
4.2	Note about submodule	13
4.3	Note about Monte Carlo	13
4.4	Documentation	14
4.5	Merging multiple files	14
5	Analysis step by step	15
5.1	Obtaining data files	15
5.2	Calibration files	15
5.3	Analysis modules	15
5.4	Analysis Run	16
5.5	Place to start - NewAnalysisTemplate	17
6	Contacts and useful links	18

1. Introduction

J-PET experiment in its design combines several technologies, that together required a set of dedicated solutions, in order to prove itself as a functioning prototype. The purpose of analyzing the acquired data is fulfilled by software project - J-PET Framework. As the name suggests, it is a library of methods created specifically for this experiment, that can be used for creation of all sort of procedures, that would be useful in advancing the research. In this document, the user of mentioned programs can find the technical references and descriptions of nearly all pieces of this Framework.

The manual starts with an outlay of the structure and design, continues with documentation about installation and software maintenance, description of available tools and finishes with an example of some custom analysis. This version of the manual was prepared for the release of 8th version of Framework, published in October 2019.

2. Structure of Framework

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 detector elements, that detector consists of and physical phenomenons, that take place during the measurements.

2.1 Detector objects

2.1.1 JPetFrame

The J-PET Detector consists of slots (two photomultipliers, one scintillator) arranged in layers (3 at the moment), as shown in Fig. 1. It is represented by `JPetFrame` class.

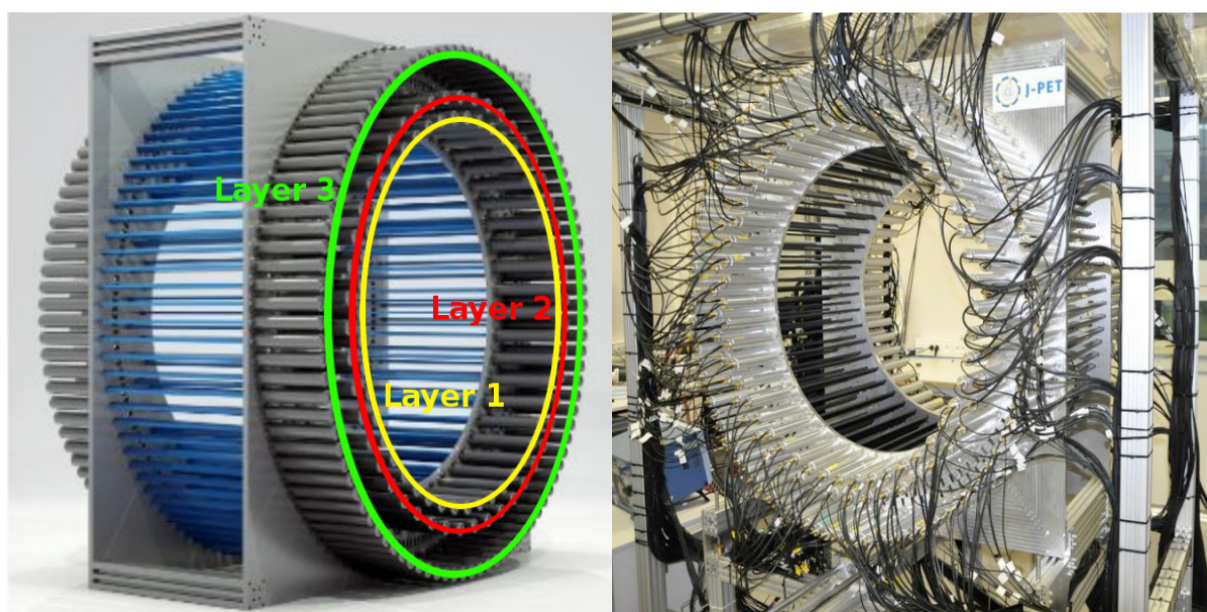


Figure 1: J-PET detector scheme with marked layers (left panel) and photo of J-PET detector setup in laboratory (right panel).

2.1.2 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.

2.1.3 JPetBarrelSlot

The representation of Slot, that consists of Scintillator with Photomultipliers attached to the ends. `JPetBarrelSlot` objects 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 $(0, 0, 0)$.

2.1.4 JPetScin

The representation of Scintillators, that capture incoming photons from the source (whatever source it is). Refer to a page of PetWiki with details about Scintillators. Each JPetScin object holds information about: ID, Sizes (dimensions), Barrel Slot it belongs to.

2.1.5 JPetPM

The representation of Photomultipliers, that measure Signals arriving from Scintillators. . Each JPetPM object has properties such as: ID, Side (A or B) - info that can be easily used to pair signals on the opposite PMs, High Voltage gains, settings and options, FEB (Front End Board) that it is connected to, Barrel Slot it belongs to.

2.1.6 Remarks so far

While using Framework there is possibility to easily obtain information about connections between objects, by using `getter` functions. So for example, if in the 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 BarrelSlot
`bool isBSActive = pm.getBarrelSlot().isActive();`

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

2.2 Data objects

2.2.1 JPetEvent

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 updated in the future to describe accurately the gathered experimental data.

2.2.2 JPetLOR

A LOR, meaning Line of Response, is a simple container with 2 hits, designed to describe a straight line between them, which can be used for medical imaging purposes. Each LOR holds information about: time [ps], time difference between hits and access to contained JPetHit objects. Idea of LOR is presented in Fig. 3.

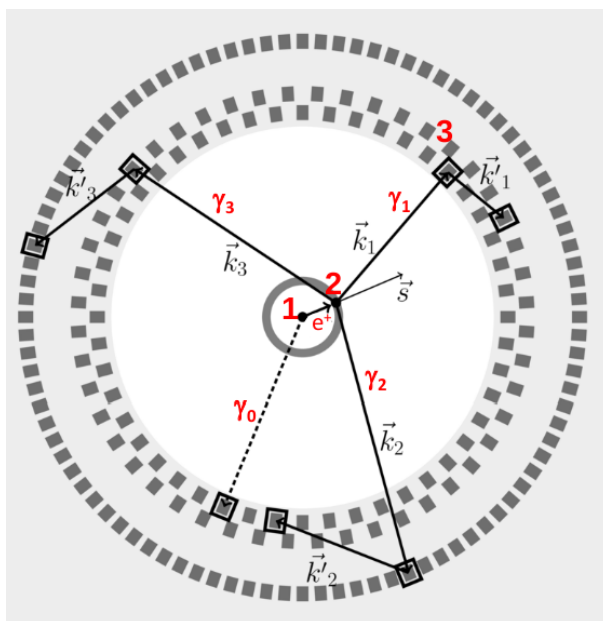


Figure 2: Schematic view of the cross section of the J-PET detector with marked ortho-Positronium decay. Ortho-Positronium decay point is marked with number 2, while outgoing photons with γ_1 , γ_2 and γ_3 .

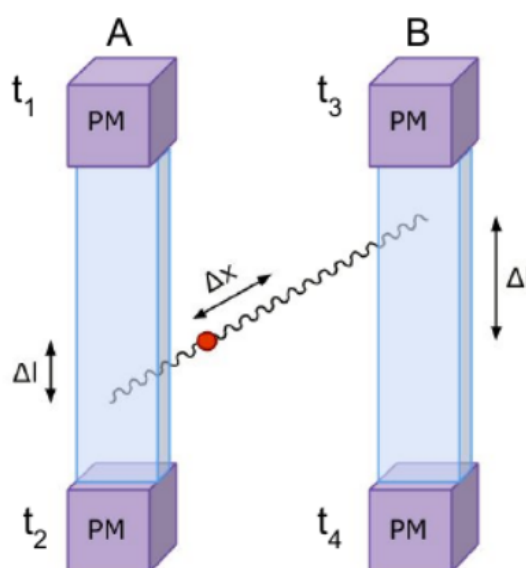


Figure 3: Idea of LOR construction with two hits in the pair of scintillators.

2.2.3 JPetHit

When photon (i.e. from the previous example) deposits energy in a Scintillator, it propagates in every direction - also towards the Photomultipliers at the ends. The deposition position is marked with yellow circle in the scheme presented in Fig. 4. Every JPetHit object contains information about: Arrival Time [ps], Energy, Position in the Scintillator (X, Y, Z), the

two Signals that construct the hit, Time difference between arrival times of two Physical Signals.

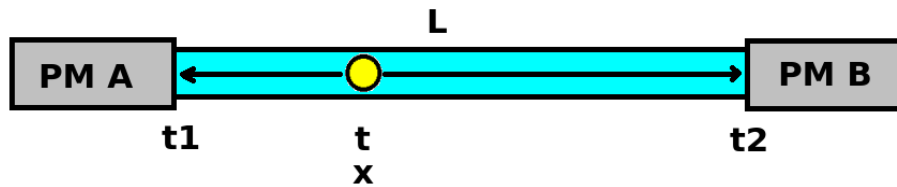


Figure 4: Schematic view of one scintillator strip with two photomultipliers on both sides. Yellow circle denotes hit of photon.

2.2.4 JPetPhysSig

The representation of reconstructed Signal, that arrives to the Photomultiplier. It contains information about: Arrival Time [ps], Number of Photoelectrons. Currently arrival time value is inherited from JPetRawSignal and method of estimating photoelectrons is not yet established. JPetPhysSig is marked with dense-pointed curve in Fig. 5.

2.2.5 JPetRawSig

The collection of 1-8 points, that are representation of Signal Channels. In the example in the Fig. 5 there is a group of 8 points - 4 red and 4 green shown for Leading Edge and Trailing Edge. Each JPetRawSig object hold info about: Number of points that construct it, Signal Channel Points with division to Leading Edge and Trailing Edge points.

2.2.6 JPetSigCh

The representation of a part a Signal, that was registered on a channel in a certain Photomultiplier. 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. 5 with red and green points, thresholds with dashed lines, and the reconstructed Physical Signal is a dense-pointed curve.

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

2.2.7 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 analyzing 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.

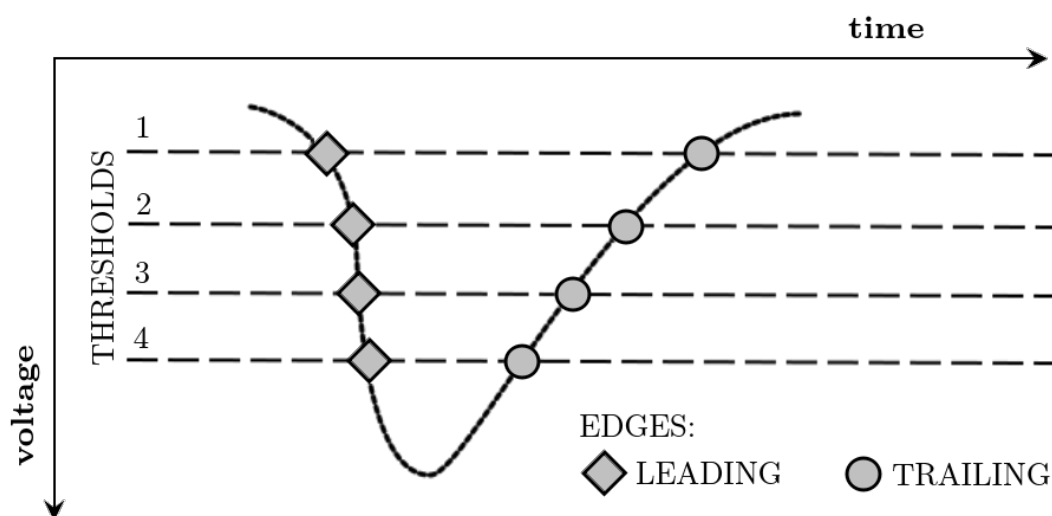


Figure 5: Schematic view of reconstructed Physical Signal (dense-pointed curve). 8 points denote representation of Signal Channel - 4 thresholds and type of Leading Edge and Trailing (4 square and 4 circle points, respectively).

2.2.8 Final remarks

When performing data reconstruction, one has to perform tasks, that construct objects from the simplest (JPetSigCh) to the most complex (JPetEvent or JPetLOR). 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. 6. It is the reverse order comparing to this presented in this Report.

2.3 Units used in reconstruction procedures

Everywhere in J-PET Framework software projects it is strongly advised to use the agreed set of units:

- voltage: mV
- time: ps
- distance: cm
- energy: keV

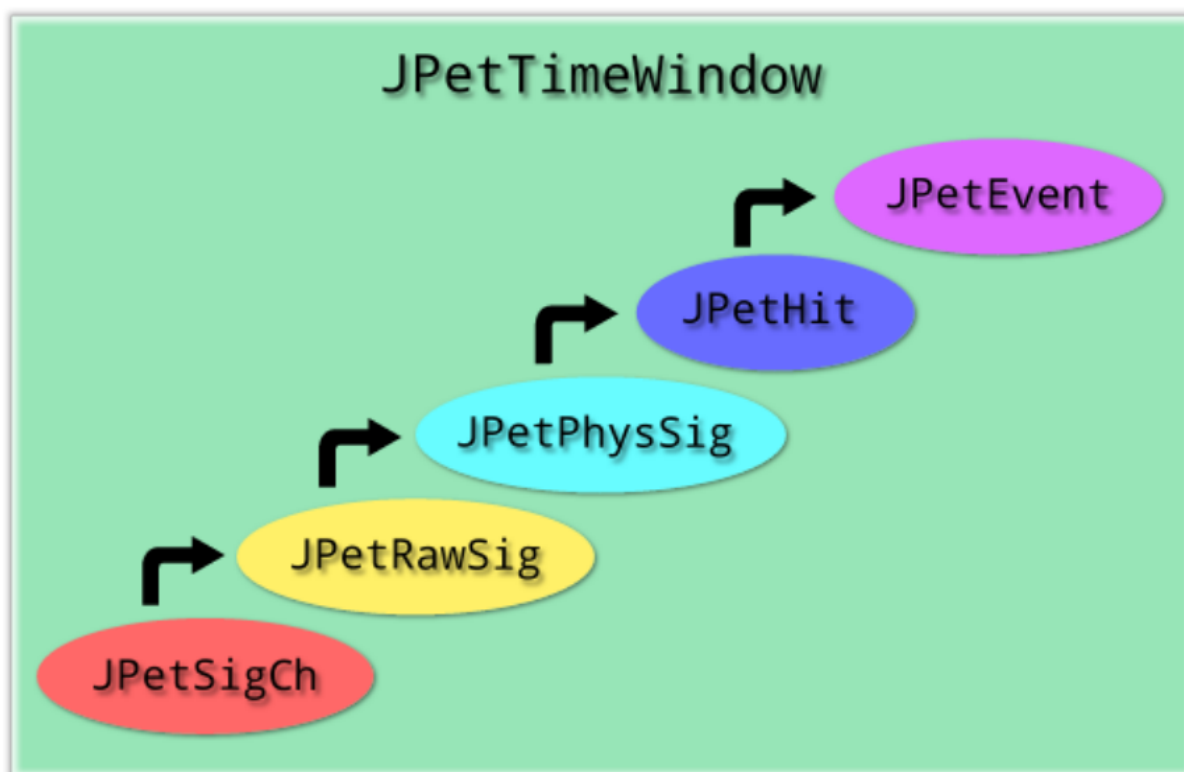


Figure 6: J-PET Objects within one Time Window.

3. Framework installation

Installation process of Framework is proven to be challenging for first-time users, but do not give up! 8th version of Framework requires performing build and installation of software with `cmake` in 3 steps:

1. Unpacker2
2. Framework library
3. Examples (with MLEM module)

The installation process is described below and all current information is posted on PetWiki. 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.

3.1 Requirements

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

- Linux Ubuntu operating system - tested and used with versions from 16.04 to 19.10.
- g++ compiler - version supporting C++11 standard, 5.0 or later
- cmake - minimal version 3.1.5

- BOOST libraries <https://www.boost.org/> - working from version 1.64
- ROOT Data Analysis Framework - version 6.10 or later (<https://root.cern.ch/>)

Always refer to `INSTALL.md` file, to check required libraries.

3.2 Installing Unpacker2

Installation of Unpacker project can be performed as described below.

```
git clone https://github.com/JPETTomography/Unpacker2.git
mkdir unpacker-build
mkdir unpacker-install
cd unpacker-build/
cmake -DCMAKE_INSTALL_PREFIX=../unpacker-install ../Unpacker2
make
make install
```

After successful installation, there should be a shared-object library `libUnpacker2.so` in set folder: `unpacker-install/lib`

Note that, if you choose to provide install path in a root directory (for example `/usr/local/`), you need to run install with admin access: `sudo make install`.

3.3 Installing Framework

This procedure is similar to installation of Unpacker2:

```
git clone https://github.com/JPETTomography/j-pet-framework.git
mkdir framework-build
mkdir framework-install
cd framework-build/
cmake -DCMAKE_INSTALL_PREFIX=../framework-install ../j-pet-framework/
```

During `cmake` the test files will be downloaded from the server.

```
make
make install
```

After successful installation, there should be a shared-object library `libJPetFramework.so` in set folder: `framework-install/lib` Note that, if you choose to provide install path in a root directory (for example `/usr/local/`), you need to run install with admin access: `sudo make install`.

3.4 Building example procedures

The repository `j-pet-framework-examples` contains several procedures, that are described in other section 4.1. Before installation with `cmake` is it required to set proper paths with `thisframework.sh` script:

```
source <path/to/framework>/bin/thisframework.sh
```

If you followed the described above example, then it is:

```
source framework-install/bin/thisframework.sh
```

Then follow the recipe:

```
git clone --recursive
  https://github.com/JPETTomography/j-pet-framework-examples.git
mkdir examples-build
cd examples-build
cmake ../j-pet-framework-examples/
make
```

How to use the executables is described in section 5.4.

3.5 Using j-pet-servers

Constantly growing computing infrastructure for J-PET experiment is administrated by Eryk Czerwiński, contact him in case of a need for access to the experimental data or server account creation. To logon to servers from outside the Institute, you need a personal VPN access, ask your supervisor how to obtain it from Departments IT staff. Using VPN access:

- Install `openvpn` on Ubuntu (version 2.4.3):
`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`:
`ssh yourUserName@serverIP`

3.6 GitHub repositories

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

- main library: <https://github.com/JPETTomography/j-pet-framework>
- examples: <https://github.com/JPETTomography/j-pet-framework-examples>
- Unpacker: <https://github.com/JPETTomography/Unpacker2>

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/>.

4. What and where and maybe... how?

4.1 Outcome of the build

After a successful build of the examples, as described in chapter 3, the folder contains the examples:

- `LargeBarrelAnalysis`, which is the current best attempt and a working version for universal reconstruction of experimental data gathered with the Large Barrel setup. Some of the modules are used in the other examples, those tasks are described in details in Sec. 5.3.
- 3 extensions of reconstruction procedures for a approach to data streaming: `CosmicAnalysis`, `Imaging` and `PhysicAnalysis`.
- calibration procedures:
 - `TimeCalibration` - a procedure analyses data from measurement with reference detector and produces files with synchronization constants of all scintillators in the detector.
 - `TimeCalibration_iter` - a iterative version of time calibration
 - `InterThresholdCalibration` - an analysis of time walk effect
 - `VelocityCalibration` - produces values of effective speed of light in scintillators
- `MCGeantAnalysis` - a task, that translates output data from J-PET Monte Carlo with Geant4 project into structures in Framework
- `ImageReconstruction` - an example using `j-pet-mlem` for imaging and sinogram creation
- `UserDataClassExample` - an example of a custom data class, that can be an extension of default `JPetEvent` or other structure
- `NewAnalysisTemplate` a place to start your own analysis. This example links the modules from `LargeBarrelAnalysis`, so a goal for a user is to add new `Tasks`, that analyze the data further, after completion of all previous procedures. It is advised to use for event categorization the tools and methods, that are available in i.e. `EventCategorizerTools` class from `LargeBarellAnalysis` example.

4.2 Note about submodule

If after executing `git clone` for obtaining project files with example procedures, the `j-pet-mlem` directory is empty, you have to obtain this submodule searately:

```
git submodule init
git submodule update
```

4.3 Note about Monte Carlo

The simulations with Mote Carlo methods are being developed as a separate project J-PET-Geant4. This is a piece of software using simulation package Geant4. The output of this program needs to be translated into J-PET Framework structures with the use of `MCGeantAnalysis`, that creates `JPetHits` and analyses them later the same way as main reconstruction procedures. The manual for working with this Monte Carlo simulator is available in here.

4.4 Documentation

The code documentation can be generated in `build` directory with Doxygen package with command:

```
make documentation
```

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

```
doxygen
```

Look for `index.html` file, that is available in the `html/` directory and open it with your favorite web browser.

4.5 Merging multiple files

The resulting files in ROOT format, created with J-PET Framework library, can be merged with customized version of `hadd` tool from this CERN software. In order to execute merging properly, the user needs to set proper variables in the `bash` environment, by invoking `thisframework.sh` script:

```
source <path/to/framework>/bin/thisframework.sh
```

Then merge desired files with the use of `hadd`, i.e:

```
hadd output.root file1.root file2.root file3.root
```

where `output.root` is naturally the result of merging. This will merge the content of the files, such as trees and histograms. It is advised not to merge data trees from large number of files - each file can have several Gb of size, so the resulting file would be enormous and merging will take a lot of time. Instead of this, only histograms can be merged, by passing proper flag: `-T`. For example, merging multiple files can look like this:

```
hadd -T output.root file*.root
```

5. Analysis step by step

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

5.1 Obtaining data files

The data from measurements with J-PET detector is recorded on servers and eventually transported to tapes 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 (uncompressed 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 analyze with Framework software. However the original filenames contain the time of the acquisition, following a rule

```
dabc_<year (two digits)><day of the year><hour><minute><second>.hld
```

There are many types of measurements done during the Runs, i.e. calibration runs - with collimator or reference detector, bare source, annihilation chamber of different sizes, imaging tests, phantoms etc.

5.2 Calibration files

During the execution of Framework tasks, the user need to provide proper calibration files, that comply with the data file. Refer to this `PetWiki` page 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 for Unpacker module (XML file)
 - detector configuration (JSON file)
- optional, but without using or misusing those files prevents the obtained results to be reasonable:
 - TDC non-linearity corrections (ROOT file)
 - TOT stretcher corrections (ROOT file)
 - signal time calibration offsets (ASCII file)
 - effective velocity of light in scintillators (ASCII file)

5.3 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()` methods are executed only once in each module. One can e.g. make there histograms, 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 procedures, that are transforming simpler data structures to a more complex ones, as it was show in Figure 6. The program starts with HLD file (it may be compressed with i.e. xz) and transforms it to sequence of ROOT files with different extensions. The output from one module is an input for the next.

5.3.1 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 with `JPetSigCh`. Uses file with time calibration offsets values and performs check for data corruption and proper flagging of signal channels.
- `SignalFinder` - creates Raw Signals with the matching procedure based on parameters of time windows.
- `SignalTransformer` - creates Reco & Phys Signals
- `HitFinder` - creates Hits from Physical signals, calculates Hit position with the use of values of velocities of light from proper calibration file.
- `EventFinder` - creates Events as group of Hits within a time window of a given value.
- `EventCategorizer` - categorizes Events with simple procedures, assigning one or several types of an event.

5.4 Analysis Run

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

```
./LargeBarrelAnalysis.x -t hld -f dabc_16218140613.hld
-l detectorSetup1.json -i 1 -p conf_trb3.xml -o outputDir/
-r 0 100000 -u userParams.json -b
```

General way of executing the program with options is:

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

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

- `opt_t` - input data file format (usually HLD, ROOT or compressed XZ, GZ, BZ2)
- `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, you can find it as a first key in the used JSON file
- `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_r` - range of analyzed events, two numbers 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.
- `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. Please see the `PARAMETERS.md` file for description of values/files to be set in JSON file. Contents of the file for `LargeBarrelAnalysis`
- `b` - optional, without any parameters, if used it displays information about current task and percentage of the progress.

5.5 Place to start - NewAnalysisTemplate

Let's start!

First enter directory `j-pet-framework-examples/NewAnalysisTemplate`. There you can find basic files: `CMakeList.txt`, `README.md` 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.

```
#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. 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
```

6. Contacts and useful links

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

- **Aleksander Gajos** (alek.gajos@gmail.com) - J-PET Framework development and PetWiki
- **Daria Kisielewska** (dk.dariakisielewska@gmail.com) - development of J-PET Monte Carlo simulations project
- **Szymon Niedźwiecki** (szymonniedzwiecki@googlemail.com) - both general and detailed knowledge of J-PET experiment
- **Kamil Rakoczy** (kamilrakoczy1@gmail.com) - code development and maintenance
- **Krzysztof Kacprzak** (k.kacprzak@alumni.uj.edu.pl) - development of J-PET Framework examples
- **Wojciech Krzemień** (wojciech.krzemien@ncbj.gov.pl) - Framework development and design
- **Eryk Czerwiński** (eryk.czerwinski@uj.edu.pl) - administration of servers and data access/management

Links:

- **J-PET Experiment homepage**
<http://koza.if.uj.edu.pl/pet/>
- **PetWiki** - all essential information, repository of publications, seminar presentations, current experiment and lab status
<http://koza.if.uj.edu.pl/petwiki/>
- **Redmine** - bug tracking, task management and discussion forum for Framework developers and users
<http://sphinx.if.uj.edu.pl/redmine/>
- **Online documentation generated with Doxygen**
<http://sphinx.if.uj.edu.pl/framework/doc/>