Modern African Nuclear DEtector Laboratory (MANDELA)

Digital Data Acquisition and Analysis

Lab Manual

(XIA-PIXIE16)

CONTENT

Dr. Kushal Kapoor

Post doctoral,

University of the Western Cape,

Cape Town, South Africa

MANDELAB Handbook

Summary of PIXIE-16 working

- For each channel that ***triggers the acquisition*** the channel identification, energy, time, and trace (if applicable) are grouped together and sorted according to the times at which they occur (*based on the Pixie16 clock*).
- The ***time sorted list of channels is then scanned*** and channels that fall within a certain time window are grouped together into events.
- ***The events are subsequently processed in two stages***:
  - The first stage operates channel by channel within the event and performs such operations as calibrations and raw parameter plotting and should not be altered.
  - The second level of processing is the experiment specific event processing and includes such activities **as correlations between implant events and decay events**. This section section should be altered according to experimental need.
- Below is the data analysis flow chart, it attempts to describe in general terms how the analysis proceeds.

Module Configuration

To configure the crates and modules data acqusition requires to read the *pxisys.ini, slot_def.set, and pixie.cfg* to configure module communication. CMake generates *pixie.cfg* based on user installed firmware. We use these three files (along with *default.set*) to boot the modules.

- Mixed Revision (i.e. RevF / RevD)
- Mixed Frequency (i.e. 100 MS/s and 250 MS/s)
- Mixed Bit Resolution (i.e. 12, 14, or 16)

For multi-crate systems pulser needs to set for syncronization.

Module Setup

Main DAQ program "poll2" handles all module communication. It includes functions to manage the channel and module parameters. DAQ utilities the programs based on XIA functions that allow

- Modification of module and channel parameters
- Capture and tuning of the channel baselines (VOFFSET)
- A toggle program that allows users to toggle specific bits of the CHANNEL_CSRA.

Poll2 uses ncurses, and provides a terminal based GUI for users. It has command history, tab completion, and scripting. We wanted the GUI to mimic use in a bash environment.

Dummy Interface

Interfacing uses the XIA API to talk to the modules. An experimental v4 branch that includes interface structure. It allows users to use the DAQ software without hardware attached. Unfortunately, there are some bugs that prevent Pixie16 modules from booting properly.

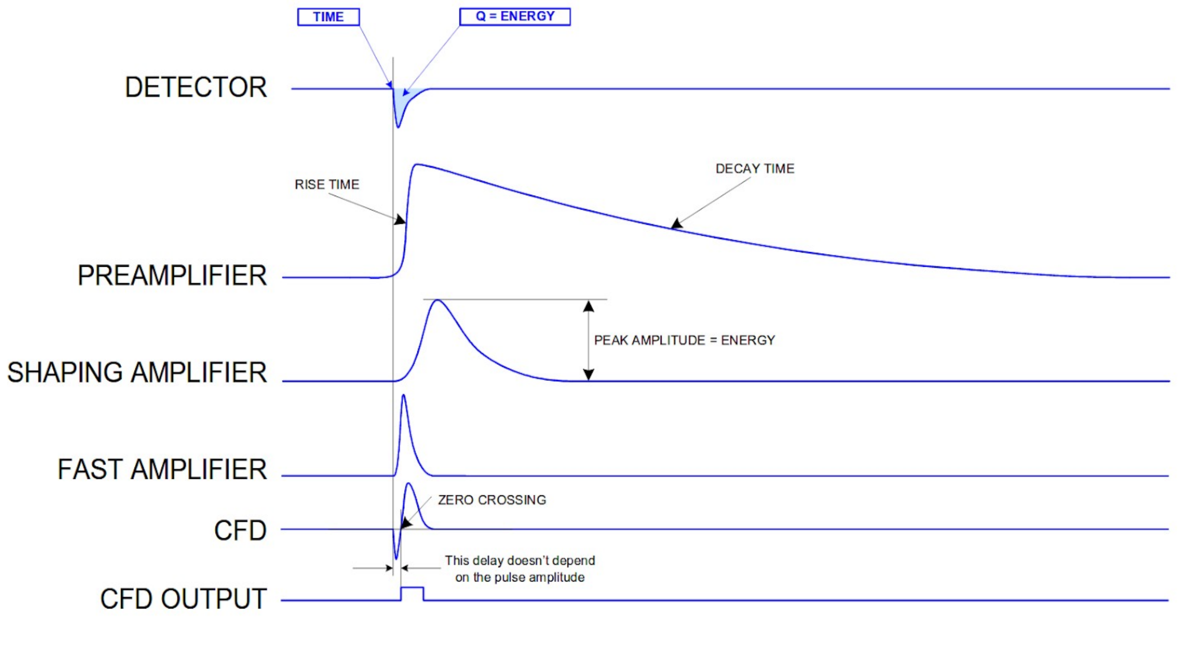


Fig. 1.3: Signals in the traditional analog chain

(Image from [CAEN]).

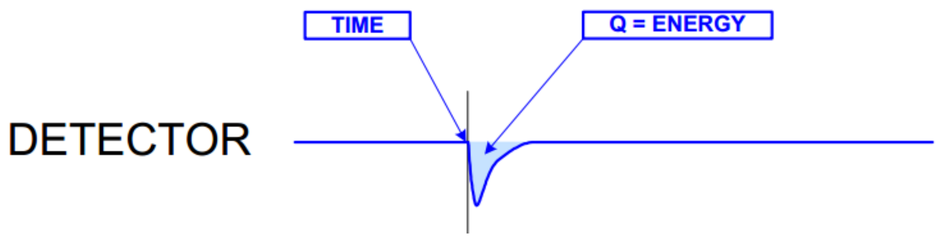
Fig. 1- Basics of signal processing.

Basic signal input and channel parameter settings

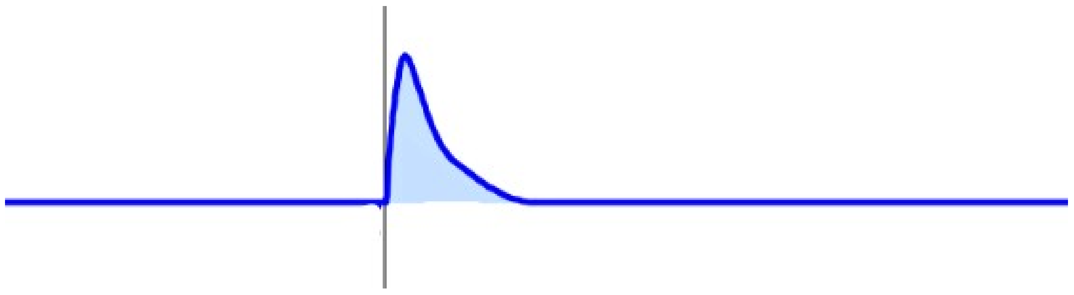
The following is a basic discussion on digital signal processing. Much is taken from the XIA PIXIE16 manual as well as the CAEN document on signal processing.

The traditional processing of analogue detector signals is illustrated in the figure below. Analog-to-digital-converters (ADC) measures the analog voltage value of the amplified signal and stores it as a number.

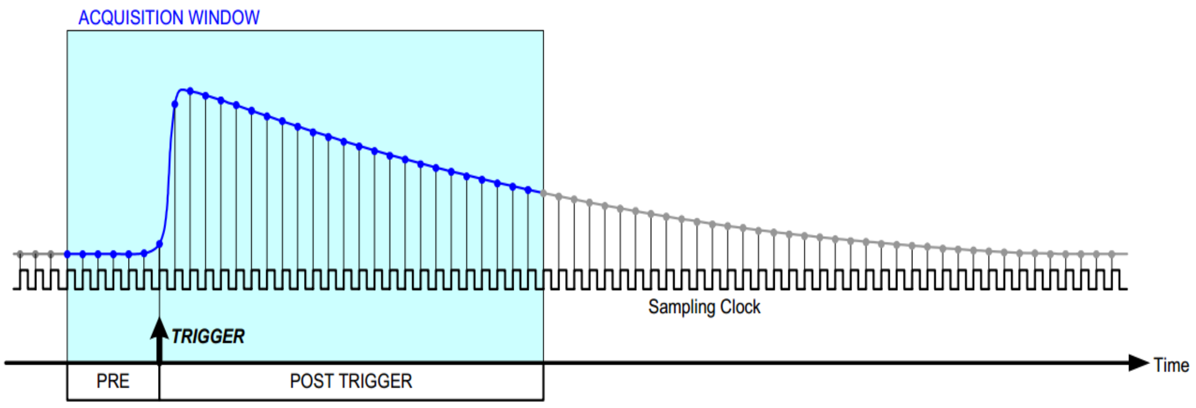
In the case of modern digital data acquisition systems, the detector output signal (top-most signal in the image above) is digitised by a Waveform Digitizer which operates similar to a digital oscilloscope.



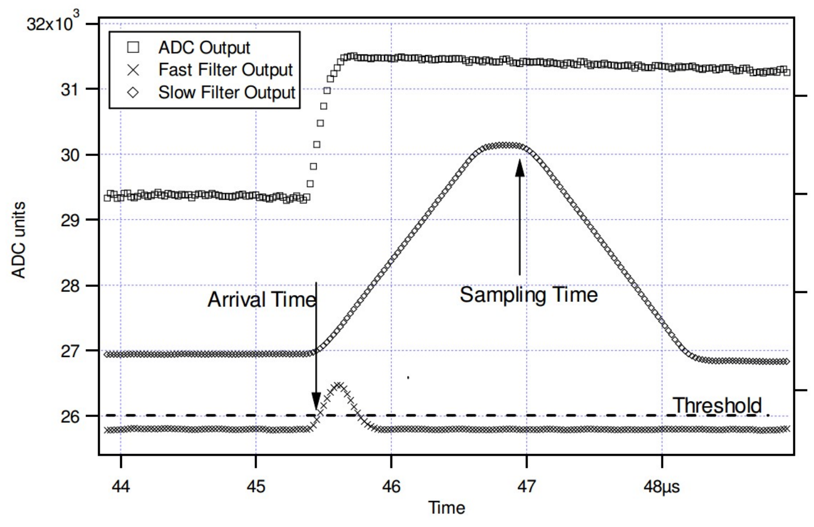
or



When a trigger occurs, a certain number of samples is captured and saved into one memory buffer. This can be seen from the image below



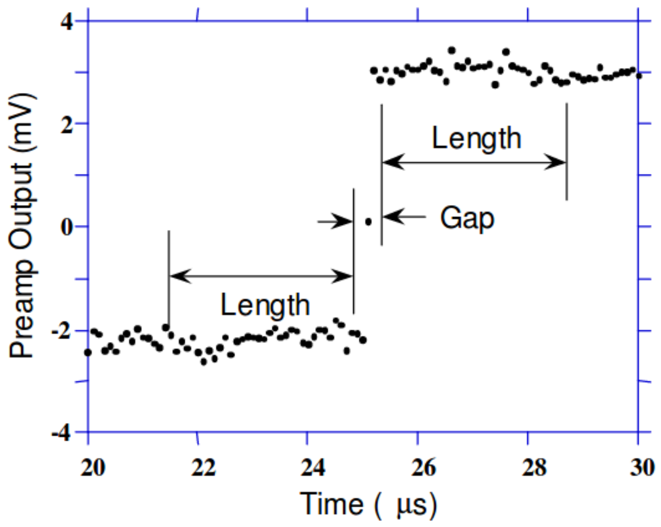
Once this **trace** of the signal is collected, further filtering and processing can be done at fast speeds. See the trace of the detector signal (ADC Output) in the figure below:



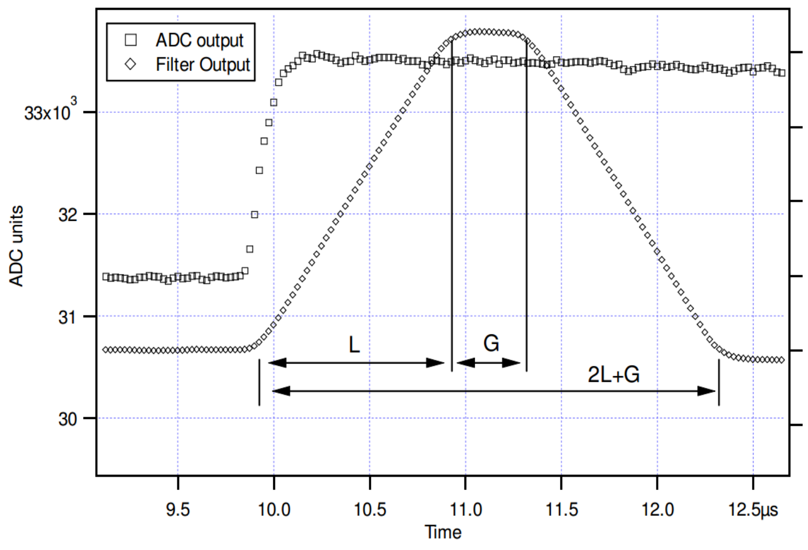
(Image from [XIA]).

In the above signal example, I consider a rise time (τ_r) of the input signal to be about 0.1 µs. The decay time (TAU, τ_d) in this example could be taken as about 25 µs.

For the slow energy filter, we can use the following as a guide to setting the filter parameters. The typical length of the ENERGY_FLATTOP (G or Gap) of the slow energy filter is taken as at least the same as the signal rise time to about 3 times that, i.e. 0.35µs. The ENERGY_RISETIME (L or Length) of the slow filter is taken as about 4 times the FLATTOP, i.e. 1.2 µs.

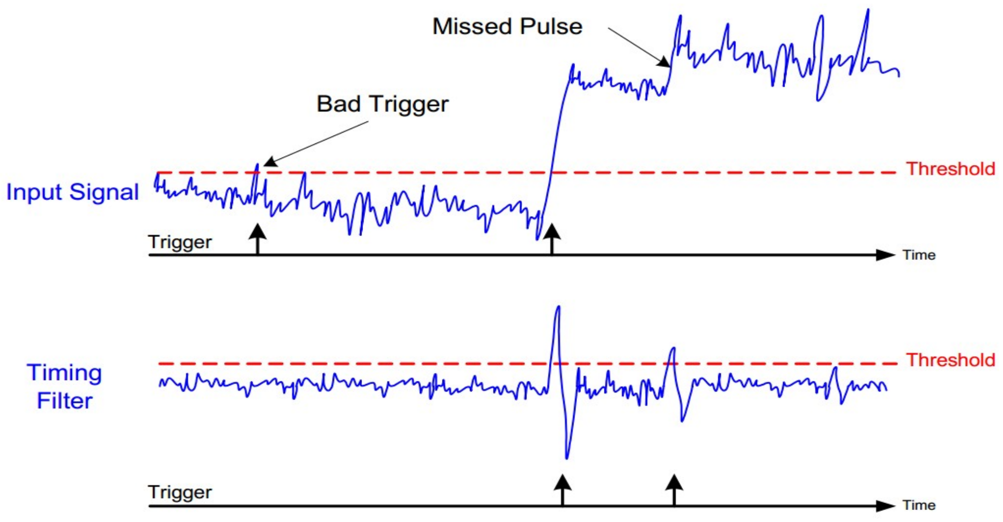


(The figure left is from [CAEN])



(figure is from [XIA].)

For the fast trigger filter, we can set the TRIGGER_RISETIME at about the same as the rise time, i.e. 0.1 µs. The event timestamp is taken at the time where the TRIGGER_THRESHOLD intercepts the fast trigger filter. The function of the trigger threshold is best illustrated in the image below:



(Image from [CAEN]).

For the trace, we can set the TRACE_LENGTH (the time over which the trace is captured) at about 2L + G. i.e. in this example, ~2.7 μs. A too long trace length will average out any subsequent signals that fall within that trace length. It is suggested that TRACE_DELAY be set to about 25% of the TRACE_LENGTH, i.e. in this case ~0.65 μs.

When setting up the PIXIE16 channel parameters, the user should consider the following:

- The energy resolution (i.e. how wide / narrow the peak of interest is) - this involves a proper choice of the slow [energy] filter settings, i.e. ENERGY_RISETIME (length L) and ENERGY_FLATTOP (gap G). For the energy (slow) filter, you need to balance throughput and energy resolution. Longer filters (i.e. long ENERGY_RISETIME) will mean that you're increasing the likelihood of pileup and effective

deadtime of the system. A shorter rise time may lead to worse energy resolution. The likelihood of pileup is (if memory serves) proportional to the filter length and the rate. The suggestion for the energy filter is to keep it as short as possible while maintaining the desired energy resolution. A shorter energy filter means that you're measuring less of the signal and may calculate a "smaller" energy.

- If you make TAU bigger it means that the adjustment to the energy due to overlapping pulses is larger, and thus you will also have a smaller energy. According to [XIA] the signal decay time (TAU) is the most critical parameter for the energy computation. It is used to compensate for the falling edge of a previous pulse in the computation of the energy.

- The dynamic range (i.e. how much of your spectrum is filled up) - note, if the baselines of the signals are adjusted too high (e.g. 1600 instead of 600), you are losing a lot of the ADC range. Shifting the baselines lower will optimise the ADC range usage. The same applies for gains. Gain levels set too low means you are not using the full dynamic range of the ADCs.

- The throughput (i.e. how fast can I shove signals through the filter) - the longer the filter times, the slower the signal analysis, i.e. aim for the shortest realistic filter ranges.

- Pileup (i.e. how likely I am to have another signal arrive while my filter is still getting calculated) - same as above.

Setting up external triggers and external time stamps

External timestamp

One option is to use the K600 spectrometer as external timestamp or as external fast trigger into the digital I/O ports of the four 250 MS/s Pixie16 Rev. F modules. If we use the front panel, then these are 5V TTL signals. When using the external timestamp, each channel signal's trigger will have two timestamps, one from local channel and one from external. You access these timestamps by calling the "*GetExternalTimestamp()*" method from "XiaData" in the processor.

Module Validation Trigger (MVT)

To set a single channel, e.g. CH0 in the master module (mod 0) as the source of the module validation trigger (MVT), do the following:

- Set **TrigConfig0**[**11:8**] = 0000 in mod 0 (this selects CH0 as the source of the internal validation trigger, Int_ValidTrig_Sgl)
- Set **TrigConfig0**[**27:26**] = 00 in mod 0 (this selects the external validation trigger, **Ext_ValidTrig_In**, above as the module validation trigger (MVT))
- Set **TrigConfig0**[**31:28**] = 0001 in mod 0 (this selects the source **Int_ValidTrig_Sgl** as the external validation trigger, **Ext_ValidTrig_In**)
- Set the Module Control Register B bit 6 (**MODULE_CSRB**[**6**]) to 1 for mod 0 to enable the sharing of the **Ext_ValidTrig_In** from this module amongst the other modules in the crate.

The above choices gives the value of **TrigConfig0** = 2147516416.

The module validation trigger is then stretched long enough, e.g. 6 us to allow other channel signals from the clover detectors to fall within this period to be validated. Channel signals outside this window will be rejected.

To use the MVT as a validation trigger for all the channels, set the following bits in the **CHANNE_CSRA** parameter to 1 for each channel:

CHANNEL_CSRA[2] = 1   (Good channel)

CHANNEL_CSRA[11] = 1   (enable MVT for this channel)

CHANNEL_CSRA[21] = 1   (record external clock timestamp counter in event header

-

GetExternalTimestamp() in processor)

One will have to stretch the MVT to an appropriate length in order to allow local channel fast triggers to come in (as illustrated in the figure above), e.g. 6 us (for 250 MHz modules, 0.008 - 32.76 us). This is set in the **ExtTrigStretch** parameter for each channel, pwrite ExtTrigStretch 6

Acquisition with POLL2

Start poll2 from the “acq” directory ~/Software/paass_usr/acq/... with:

```
$: poll2
```

or

```
$: poll2 -f (for fast boot)
```

Wait for successful boot-up:

```
Module  0: Serial Number 1082, Rev F (15), 14-bit 100 MS/s
Module  1: Serial Number 1085, Rev F (15), 14-bit 250 MS/s
Using FIFO threshold of 50% (65536/131072 words).
Using Pixie16 revision F


Starting run control thread.......................[OK]
Starting command thread...........................[OK]
```

```
$: run                                        (To starts the acqusition),
$: stop                                       (To stop the acqusition),
```

It run the acqusition in the MCA mode, in order to run it in List mode use the following commands:

```
$: startacq                                        (To starts the acqusition),
$: stop                                            (To stop the acqusition),
```

Once the data has been stored, files containing event information is created in *.ldf* format. This is further needed to be decoded using utkscan or other event builder.

Utkscan gives us files in *.root* and *.tree* format, which can be analysed further using ROOT. *(Details are given below)*

Analysis

Analysis Libraries

XiaListModeData{Encoder, Decoder, Mask}

These classes provide the functionality needed to encode or decode any XIA header. We worked with XIA to create a history of the changes to the list mode data headers. The history provides a time line of changes to the header. It ensures the correct bit masks when decoding list mode data. XiaData stores all the raw information collected by the Pixie-16 modules.

Unpacking

The unpacking classes read list mode data files from *poll2*. It uses the *XiaListModeDataDecoder* to decode module data. This class also performs basic statistics as its unpacking data.

Event Building

After unpacking the data, it needs to be packaged into time structured events. An event is defined as the group of channels that fired within a user specified time window. This allows users to analyze time structures and correlations in their data.

Utilities

- Event Reader – The event reader reads a data file and outputs the events to the terminal.
- Head Reader – Reads metadata from the list mode data files
- Hex Reader – Reads data from list mode data files and outputs to the terminal in hex format.

- Root Scanner – This program provides on-the-fly histogram generation in the ROOT framework. It produces a ROOT file containing the data.
- Scope – Used to view traces in list mode data. It analyzes traces using CFDs, Fitting Algorithms, or Trapezoidal Filters. Use with SHM mode to set appropriate trace lengths and delays. We adapted Trapezoidal Filter algorithms from an Igor script provided by H. Tan at XIA.
- Skeleton – A bare bones (get it?) program provided for users to build their own analysis.

Utkscan

This is the main workhorse analysis program. It provides a bunch of classes that are capable of analyzing many detector types. It has been written in a modular format to maximize usability. Utkscan includes modules for experiment specific analysis. This allows users to expand the functionality without modifying core features.

An XML configuration allows users to reconfigure the software without recompiling.

Summary

PAASS-LC provides users with a solid base for their DAQ and Analysis needs. Utkscan, poll2 provides direct access to the Pixie-16 modules for configuration, and have a shared memory system in place.

- ***Analysis flow (flowchart to understand the an event in PIXIE-16):***
  - ***#Event Creation:***
    - For each buffer that is collected (*either list mode or mca run*) the **SCAN routine** invokes the **hissub_() function**. The *hissub_(), hissub_sec(),*

and ReadBuffData() functions serve to ultimately reconstruct a spill (*allow-to-flow-data*) of Pixie16 data.

- i.e.-> Reads all of the Pixie16 modules and sort the channels that triggered according to their times.

▪ **ScanList()** is called on the time sorted event list to group channels with similar times into events, store these channels in the variable rawevent and pass them on for further processing.

- ***#Experiment Independent Processing:***
    - In the function DetectorDriver::ProcessEvent(), the RawEvent is processed.
    - Each channel is checked against its threshold value read in during initialization from the XML Based Configuration File and calibrated using ThreshAndCal().
    - The function ThreshAndCal() also keeps track of the multiplicities of different detector types and the maximum energy deposited into each detector type.
    - The raw and calibrated energies are then plotted using the functions *PlotRaw()* and *PlotCal()* using the DAMM spectra numbers read in from the XML Based Configuration File.
    - This is the end of general event processing and should not be altered between experiments.

- ***#Experiment Dependent Processing:***
    - The next step in the event processing is the experiment specific tasks and these will vary for different setups but include such tasks as determining correlations between decays and implants.

- ***Add A New Detector Processor for analysis:***

- In principle, one can add a Processor to handle a specific detector types, or to handle the existing processors.
- To add a new detector first a class must be constructed which defines its behavior by **creating a .hpp and .cpp file**.
  - i.e., "**Template**"-> Understanding the header and source discretely.
    - The first four line comments:-> gives the name of the file, a brief description, the author and the date that it was originally written: (i.e.)
      - /** \file TemplateProcessor.hpp
        * \brief A Template class to be used to build others.
        * \author Kushal Kapoor
        * \date May 6, 2019
        */

- ***#Include Guards*** The include guards should have the form displayed here. It's the name of the file/class with two underscores framing it.

```
#ifndef ___TEMPLATEPROCESSOR_HPP___
#define ___TEMPLATEPROCESSOR_HPP___
```

- ***#Local Includes*** The detector processor is derived from-> **EventProcessor** one needs to include the parent class header. We get away from having to include the vector header since the **EventProcessor** already includes it.

```
#include "EventProcessor.hpp"
class TemplateProcessor : public EventProcessor {
```

public-> what people will need to know about when using the class.

- ***#Default Constructor*** The default constructor is called when instancing (citing) the class.

```
/** Default Constructor */
TemplateProcessor();
```

- ***#PreProcess*** This method handles any processing of the signals from the detector that does not depend on any other detector class.
  - i.e., DSSD method would calculate the pixel that was activated.

    ```
    virtual bool PreProcess(RawEvent &event);
    ```

- ***#Process*** This method handles any processing of the signals that may depend on other detectors but has **no thresholds, cuts or other restrictions put on the histograms**.
  - i.e., Histogramming of the ToF with VANDLE bars.

    ```
    virtual bool Process(RawEvent &event);
    ```

- ***#Get Methods*** -> get methods are used by the experiment processors to obtain information calculated in the preprocess stage.
  - /** \return The processed Template events*/ const std::vector<ChanEvent*>*GetTemplateEvents(void)const { return(&evts_); }

- ***#Private methods and variables*** The private methods and variables always come last, they are things that people generally will not be needing to use.

    ```
    private: std::vector<ChanEvent*> evts_;
    ```

Running Utkscan

To run Utkscan, use the command:

Go to the analysis folder in paass-lc,

Use the command

./utkscan -i *run001.ldf* -c *config.xml* -o *out*

-i -> is the input file

-c -> is the config file which contains Module information & other important information for configuration.

-o is the outfile name, which creates *out.root* & *out.tree*