

*Startup Guide
for
 $n\text{-}^3\text{He}$ Analysis*

Latiful Kabir
Version: 2.0

Contents

| | | |
|----|---|----|
| 1 | Resources | 3 |
| 2 | Quick start on basestar | 4 |
| 3 | Event length, dead time and file size | 5 |
| 4 | The data file structure | 6 |
| 5 | ADC count to Volt conversion and time bin | 10 |
| 6 | The ADC channel to wire map | 11 |
| 7 | Setting up local version of n3He analysis library on basestar | 13 |
| 8 | Setting up local version of data browser on basestar | 15 |
| 9 | Current tree structure in n3He analysis library | 17 |
| 10 | Sample analysis | 19 |
| 11 | Reference for TTreeRaw class | 23 |
| 12 | The summary root file | 24 |
| 13 | Individual run status | 26 |
| 14 | Reference for run numbers | 27 |
| 15 | Reference for beam centroid | 29 |
| 16 | Summary of n3He data | 30 |
| 17 | List of n3He parameters | 31 |

1 Resources

All the startup software and manual related to n-³He experiment can be found from the official git repository with detailed instruction.

The easiest way is to go to n3He wiki (n3he.wikispaces.com) and then click on software from the left panel.

Alternatively, here is a direct link.

Any new change goes to this git repository. You might want to clone the entire repository and pull periodically to be updated. Or you can also download the last release of only what your are interested in.

2 Quick start on basestar

On basestar the data is being transferred and saved to the directories:

[/mnt/idata01/data/](#) (run numbers: 14548-23661),
[/mnt/idata02/data/](#) (run numbers: 23662 -31839),
[/mnt/idata03/data/](#) (run numbers: 31840- 40703),
[/mnt/idata04/data/](#) (run numbers: 40704 -44385),
[/mnt/idata05/data/](#) (run numbers: 44386 - 45999),
[/mnt/idata06/data/](#) (run numbers: 46000 - 54040),
[/mnt/idata07/data/](#) (run numbers: 54041 - 58959),
[/mnt/idata08/data/](#) (run numbers: 58960 - 61000)
and the analysis library is compiled in a shared directory [/home/npdg/n3He/libn3He/lib](#)

Actual analyzable run for UD asymmetry starts from run number:18000.
Run numbers before that are different R&D runs and LR asymmetry runs.

So a quick start using the compiled library can be as follows from any user account:

1. Add the following lines to the .bashrc file & save it

```
if [ -f /home/npdg/n3He/libn3He/bin/thisn3He.sh ]; then  
    . /home/npdg/n3He/libn3He/bin/thisn3He.sh  
fi
```

2. Start a new terminal , go to [/home/npdg/n3He/libn3He/analysis/](#) directory & try running sample analysis scripts from ROOT.
3. The data browser GUI (named as n3HeData) can be opened issuing the command [/home/npdg/n3He/n3HeData/n3HeData](#) from the terminal. Copy the binary to your home directory if you will be using the GUI frequently.

Other user specific customization can be achieved following the instruction in the ReadMe file in the respective directory.
For a local version of the library and data browser please read the corresponding section.

3 Event length, dead time and file size

The event length set in the clean DAQ at 50 KHz sample rate is :830 . Theoretically the maximum possible value is $50\text{KHz} \times 16.66 \text{ ms} = 833 \text{ samples}$ per T0 .

But 833 event length gives occasional overlap. So we set to 830.

For dirty DAQ at 100 KHz the theoretical number of samples $100\text{KHz} \times 16.6666 \text{ ms} = 1667$

But to avoid overlap we set to 1660.

More over the DAQ has fixed dead time(readout time) of 35 samples(with no averaging) at the end of any event. This amount of time will be missed for every event.

Clean DAQ event length 830 with nacc=16,16 with hi resolution mode=1

Dirty DAQ event length 1660 with nacc=1,1 with hi resolution mode=0

where nacc=n,n indicates how many samples being averaged.

Thus number of sample per event:

Clean DAQ: $(830-35)/16=49.68 \sim 50$ (1 header + 49 samples)

Dirty DAQ: $(1660-35)=1625$ (1 header + 1624 samples)

Thus the dead time in the DAQ will be –

Clean DAQ : $(52 - 49) \times 320 \text{ micro sec} = .96 \text{ milli sec}$

Dirty DAQ : $(1667-1624) \times 10 \text{ micro sec} = 0.430 \text{ milli sec}$

Run Length/file size calculation:

With 25000 T0 per run-

Clean DAQ file size: $25000 \text{ T0} \times 50 \text{ samples} \times 4 \text{ Byte per sample} \times 48 \text{ Channels} = 240 \times 10^6 \text{ Bytes}$

Dirty DAQ file size (before processing): $25000 \text{ T0} \times 1625 \text{ samples} \times 4 \text{ Bytes per sample} \times 8 \text{ channels} = 1300 \times 10^6 \text{ Bytes}$

Dirty DAQ file seize (after processing) : $25000 \text{ T0} \times 1625 \text{ samples} \times 4 \text{ Bytes per sample} \times 2 \text{ channels} = 325 \times 10^6 \text{ Bytes}$

4 The data file structure

48 Clean DAQ channels divided into two modules:

Each sample is 4 bytes(in hexdump one contiguous pair consists one sample or 4 bytes). Out of this 32 bit(4 bytes) , our data is 24 bit and remaining least significant 8 bits are channel ID.



```
Terminal - kabir@basestar:~
File Edit View Terminal Go Help
[kabir@basestar ~]$ hexdump /mnt/ldata01/data/run-20673data-21 | head -n 20
00000000 f154 aa55 f154 aa55 f154 aa55 f154 aa55
00000010 0001 0000 0000 0000 0002 0000 0000 0000
00000020 f154 aa55 f154 aa55 f154 aa55 f154 aa55
00000030 0001 0000 0000 0000 0002 0000 0000 0000
00000040 f154 aa55 f154 aa55 f154 aa55 f154 aa55
00000050 0001 0000 0000 0000 0002 0000 0000 0000
00000060 f15f aa55 f15f aa55 f15f aa55 f15f aa55
00000070 0000 0000 0000 0000 0002 0000 0000 0000
00000080 f15f aa55 f15f aa55 f15f aa55 f15f aa55
00000090 0000 0000 0000 0000 0002 0000 0000 0000
000000a0 f15f aa55 f15f aa55 f15f aa55 f15f aa55
000000b0 0000 0000 0000 0000 0002 0000 0000 0000
000000c0 0020 0000 0021 0000 0022 0000 0023 0000
000000d0 0024 0000 0025 0000 0026 0000 0027 0000
000000e0 0028 0000 0029 0000 002a 0000 002b 0000
000000f0 0034 0000 0035 0000 0036 0000 0037 0000
00000100 0038 0000 0039 0000 003a 0000 003b 0000
00000110 003c 0000 003d 0000 003e 0000 003f 0000
00000120 0040 0000 0041 0000 0042 0000 0043 0000
00000130 0044 0000 0045 0000 0046 0000 0047 0000
[kabir@basestar ~]$
```

Figure 1: Typical view of hexdump

The above hexdump to be interpreted as follows:
With: mod= module ch = channel

```
mod1event1sample1ch0 mod1event1sample1ch1 mod1event1sample1ch3 ..... mod1event1sample1ch8
mod1event1sample1ch9 mod1event1sample1ch10 mod1event1sample1ch11 ..... mod1event1sample1ch16
mod1event1sample1ch17 mod1event1sample1ch18 mod1event1sample1ch19 ..... mod1event1sample1ch23

mod2event1sample1ch0 mod2event1sample1ch1 mod2event1sample1ch3 ..... mod2event1sample1ch8
mod2event1sample1ch9 mod2event1sample1ch10 mod2event1sample1ch11 ..... mod2event1sample1ch16
mod2event1sample1ch17 mod2event1sample1ch18 mod2event1sample1ch19 ..... mod2event1sample1ch23
```

```

mod1event1sample2Ch0 mod1event1sample2ch1 mod1event1sample2ch3 ..... mod1event1sample2ch8
mod1event1sample2Ch9 mod1event1sample2ch10 mod1event1sample2ch11 ..... mod1event1sample2ch16
mod1event1sample2Ch17 mod1event1sample2ch18 mod1event1sample2ch19 ..... mod1event1sample2ch23

mod2event1sample2Ch0 mod2event1sample2ch1 mod2event1sample2ch3 ..... mod2event1sample2ch8
mod2event1sample2Ch9 mod2event1sample2ch10 mod2event1sample2ch11 ..... mod2event1sample2ch16
mod2event1sample2Ch17 mod2event1sample2ch18 mod2event1sample2ch19 ..... mod2event1sample2ch23

.....

up to n samples

.....
.....
.....

Up to N number of events.

```

where, n= the event length(number of time bins in an event), N= number of T0 (event) in a run.

Now the first sample of any event is the event header with following structure:

```

mod1event1sample1Ch0=mod1event1sample1Ch1=
mod1event1sample1Ch2=mod1event1sample1Ch3 = Event Signature-1 (0xaa55f154)
,
mod1event1sample1Ch4= Event Number
mod1event1sample1Ch5 = checksum using path-1
mod1event1sample1Ch6 = sample number
mod1event1sample1Ch7 = checksum using path-2
Then this pattern repeats 3 more times (i.e. in quanta of 8 channels) up to
channel-23

```

```

mod2event1sample1Ch0 =mod2event1sample1Ch1=mod1event1sample1Ch2
= mod2event1sample1Ch3= Event Signature-2 (0xaa55f15f)
mod1event1sample1Ch4= 0 (always)
mod2event1sample1Ch5 = checksum using path-1
mod2event1sample1Ch6 = sample number
mod2event1sample1Ch7 = checksum using path-2
Then this pattern repeats 3 more times (i.e. in quanta of 8 channels) up to

```

channel-23

For Dirty DAQ the data is taken in 8 channels (bank mask B) with one module only and then processed to 2 channels. On BNC panel, M1 signal is connected to marked channel-26 and RFSF signal is connected to marked channel 27. This corresponds to ADC channel-5 (with checksum) and channel-6 (with sample number) where for ADC channel number starts with 0.

So for the Dirty DAQ after the processing the data structure is:

```
event1sample1Ch0 event1sample1ch1
event1sample2Ch0 event1sample2ch1
... ..
up to n samples
.....
```

Up to N events.

with the first sample of any event being checksum and sample number i.e.
event1sample1ch0 = checksum.
event1sample1ch1 = sample number.

In the analysis we recommend to skip the last event to be safe in case the last event of the run is a partial event (though every effort has been made to avoid this).

Bad data/run:

Sometimes the header shows up in the incorrect position in the data file making the data/run unusable. We call this the “header or trigger issue” of the DAQ. This can be identified by detecting the event signature (which has a value greater than 6 volt) in non-header region.

Header of Different channels for each event in decimal format

| | | | | | | | |
|-----------------|-----------------|-----------------|-----------------|---------|-----------------|------------|-----------------|
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:1 | ch#5:0 | ch#6:2 | ch#7:0 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:2 | ch#5:2966364697 | ch#6:704 | ch#7:2966364697 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:3 | ch#5:2287808137 | ch#6:1406 | ch#7:2287808137 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:4 | ch#5:2075675947 | ch#6:2108 | ch#7:2075675947 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:5 | ch#5:3109521645 | ch#6:2810 | ch#7:3109521645 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:6 | ch#5:386965141 | ch#6:3512 | ch#7:386965141 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:7 | ch#5:1729520582 | ch#6:4214 | ch#7:1729520582 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:8 | ch#5:693835723 | ch#6:4916 | ch#7:693835723 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:9 | ch#5:1096971516 | ch#6:5618 | ch#7:1096971516 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:10 | ch#5:1003074332 | ch#6:6320 | ch#7:1003074332 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:11 | ch#5:2279207300 | ch#6:7022 | ch#7:2279207300 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:12 | ch#5:1877619817 | ch#6:7724 | ch#7:1877619817 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:13 | ch#5:1093183995 | ch#6:8426 | ch#7:1093183995 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:14 | ch#5:2947198100 | ch#6:9128 | ch#7:2947198100 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:15 | ch#5:1810161880 | ch#6:9830 | ch#7:1810161880 |
| ch#0:2857759060 | ch#1:2857759060 | ch#2:2857759060 | ch#3:2857759060 | ch#4:16 | ch#5:1067282547 | ch#6:10532 | ch#7:1067282547 |

Figure 2: Header of different channels for each event in decimal

Header of Different channels for each event in hexadecimal format (The sample number is also in Hex)

| | | | | | | | |
|---------------|---------------|---------------|---------------|---------|---------------|-----------|---------------|
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:1 | ch#5:0 | ch#6:2 | ch#7:0 |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:2 | ch#5:b0cf2219 | ch#6:2c0 | ch#7:b0cf2219 |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:3 | ch#5:885d2e89 | ch#6:57e | ch#7:885d2e89 |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:4 | ch#5:7bb84d2b | ch#6:83c | ch#7:7bb84d2b |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:5 | ch#5:b95788ed | ch#6:afa | ch#7:b95788ed |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:6 | ch#5:17109e95 | ch#6:db8 | ch#7:17109e95 |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:7 | ch#5:671663c6 | ch#6:1076 | ch#7:671663c6 |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:8 | ch#5:295b17cb | ch#6:1334 | ch#7:295b17cb |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:9 | ch#5:416274fc | ch#6:15f2 | ch#7:416274fc |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:a | ch#5:3bc9b31c | ch#6:18b0 | ch#7:3bc9b31c |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:b | ch#5:87d9f184 | ch#6:1b6e | ch#7:87d9f184 |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:c | ch#5:6fea3469 | ch#6:1e2c | ch#7:6fea3469 |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:d | ch#5:4128a9fb | ch#6:20ea | ch#7:4128a9fb |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:e | ch#5:afaaac94 | ch#6:23a8 | ch#7:afaaac94 |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:f | ch#5:6be4e0d8 | ch#6:2666 | ch#7:6be4e0d8 |
| ch#0:aa55f154 | ch#1:aa55f154 | ch#2:aa55f154 | ch#3:aa55f154 | ch#4:10 | ch#5:3f9d7073 | ch#6:2924 | ch#7:3f9d7073 |

Figure 3: Header of different channels for each event in hex

5 ADC count to Volt conversion and time bin

The resolution and device range of a DAQ device determine the smallest detectable change in the input signal. We can calculate the smallest detectable change, called the precision (code width), using the following formula.

$$Precision = \frac{ADCRange}{2^{resolution}}$$

Our DAQ is a 24 bit ADC. But each sample is stored as 32 bit wording. Out of this 32 bit least significant 8 bit is the channel ID and the remaining 24 bit contains the signal. And the ADC has a range of +10 V to -10 V. Then,

$$Precision = \frac{20V_{olt}}{2^{32}}$$

Thus $4.656612873 \times 10^{-9}$ will be an approximation for the conversion factor from ADC value to volt (if the least 8 bits are NOT thrown away). But to be precise, it is recommended to throw away the least significant 8 bit out of 32 bit, (use *ADCCount* >> 8 in your analysis code), then

$$Precision = \frac{20V_{olt}}{2^{24}}$$

i.e. $1.192092896 \times 10^{-6}$ will be the correct ADC count to volt conversion factor (if 24 bit is used throwing away least significant 8 bit).

The time bin depends on the sample rate at which the DAQ is taking data. For the n3He experiment we run the clean DAQs at 50 KHz sample rate and dirty DAQs at 100KHz. Again for clean DAQs 16 samples are averaged to give one sample and for dirty DAQ no averaging is used. Thus for clean (detector) data each time bin corresponds to:

$$16 \times \frac{1}{50KHz} = 320\mu sec$$

And for dirty data it is

$$\frac{1}{100KHz} = 10\mu sec$$

6 The ADC channel to wire map

Out of 48 clean ADC channels, only 36 ADC channels are connected to the pre-amps (ADC channels 0 to 17 and channels 24 to 41). There are two bad channels: DAQ21- channels 5 & 6(they are combined on channel 6). Moreover (in UD mode) DAQ22 channels 35 and DAQ 24 channel-35, 39 have opposite polarity (they can be corrected easily by flipping the sign in the analysis).

Following is the ADC channel to wire map for reference:

```
Number of layers = 16;
Number of wires per layer= 9;

Layer_to_DAQ_map[Nlayers]={21, 23, 21, 23, 21, 23, 21, 23,
                             22, 24, 22, 24, 22, 24, 22, 24};

Layer_to_ADC_channel_map[16][9] =
{
    {0,1,2,3,4,5,6,7,8},
    {0,1,2,3,4,5,6,7,8},
    {9,10,11,12,13,14,15,16,17},
    {9,10,11,12,13,14,15,16,17},
    {24,25,26,27,28,29,30,31,32},
    {24,25,26,27,28,29,30,31,32},
    {33,34,35,36,37,38,39,40,41},
    {33,34,35,36,37,38,39,40,41},
    {0,1,2,3,4,5,6,7,8},
    {0,1,2,3,4,5,6,7,8},
    {9,10,11,12,13,14,15,16,17},
    {9,10,11,12,13,14,15,16,17},
    {24,25,26,27,28,29,30,31,32},
    {24,25,26,27,28,29,30,31,32},
    {33,34,35,36,37,38,39,40,41},
    {33,34,35,36,37,38,39,40,41},
};
```

Thus , if we label layers and wires starting from 1, then the above mapping to be interpreted as -

The ADC21 , channel-0 corresponds to layer 1 wire 1.
The ADC24, channel-40 corresponds to layer 16 wire 8.
The ADC22, channel-5 corresponds to layer 9 wire 6. And so on.
Where (for Up-down asymmetry) wire 1 label starts from (beam) bottom of any layer. And wire 9 is the (beam) top wire of any layer.
And the layer counting starts from the face where beam enters the chamber.

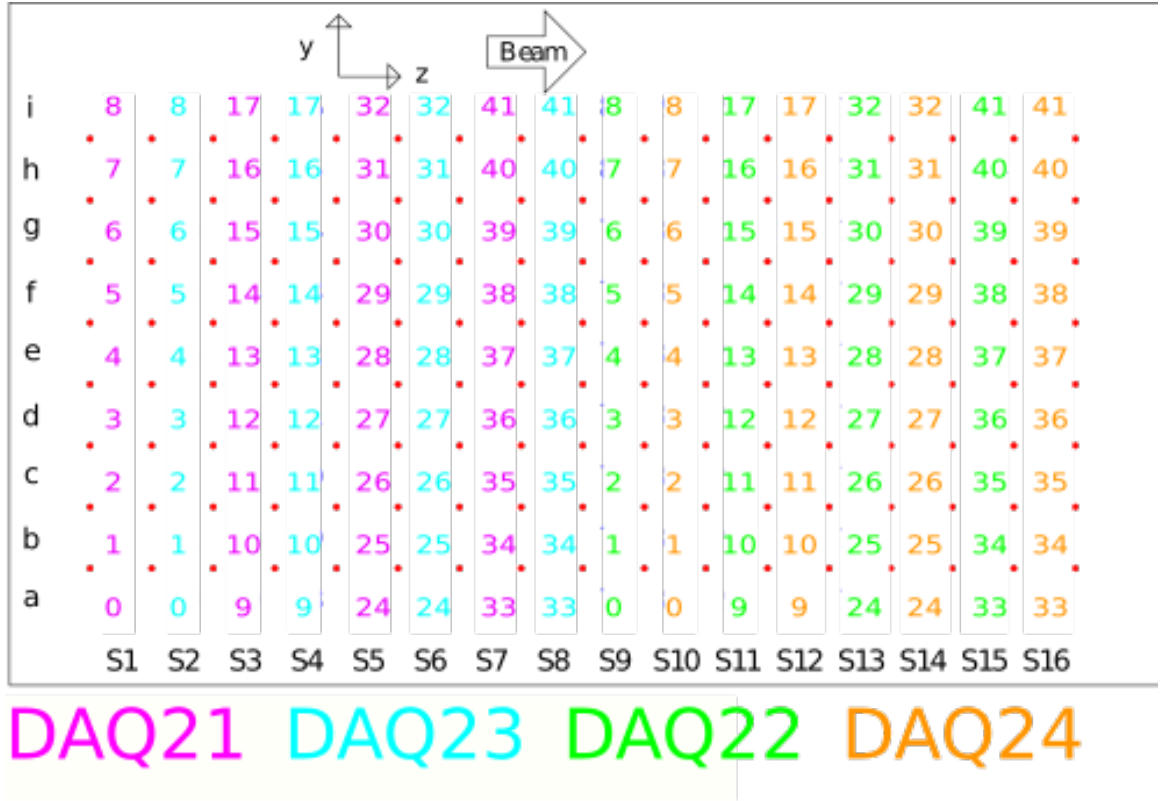


Figure 4: ADC to Wire Map

7 Setting up local version of n3He analysis library on basestar

Eventually you will want to set up your own version of the analysis library and ROOT environment. This way you can modify any part of the library and add more functionality.

1. Download the source code from here.
lib3He is the analysis library for n3He experiment.
2. Make any necessary changes in Constants.h file that is required.
3. Do **make** to compile the library.
4. This will produce **libn3He.so** (shared library will be inside lib directory).
5. Place the .so file in a directory under **LD_LIBRARY_PATH** .
6. Now start root and load the Library as: **gSystem->Load("libTree")**
& **gSystem->Load("libn3He.so")** . (For Online analysis)
7. For analysis from a script if you include TTree.h file then you need not to do **gSystem->Load("libTree")**; Just load **gSystem->Load("libn3He.so")**.
You need to give full path unless the directory is included in **LD_LIBRARY_PATH**.
8. If you put the **rootlogon.C** file in **macros** directory under Root installation directory, then the library will be loaded automatically and step-6 is NOT necessary.
If you do NOT have local version of ROOT, rather it's installed under root account(a shared version, which is the case for basestar by default), then just copy system.rootrc file from **/usr/share/root** to your home directory and rename it to **.rootrc**, Now add the directory having **rootlogon.C** file to the list of **Unix.*.Root.MacroPath** variable.
9. Now from your root script create a Tree by calling: **TTreeRaw *my_tree = new TreeRaw(runNumber#)** or Just **TTreeRaw t(runNumber#)**
10. Do **my_tree->Print()** to print the tree and branch structure.
11. Now do what ever analysis you want using **my_tree** .

12. Try running example analysis scripts in “analysis” directory.
13. To make life easier it’s convenient to put the following command into your `~/.bash_profile` or `~/.bashrc` file:

```
if [ -f /path/to/libn3He/bin/thisn3He.sh ]; then  
    . /path/to/libn3He/bin/thisn3He.sh  
fi
```

Note: This version of the library works both for ROOT 5 and ROOT 6.

8 Setting up local version of data browser on basestar

1. Download the source code from here.
2. In bin directory: contains just binary files(obtained after doing make) named `n3HeData`.
3. In libn3He directory: Contains all the library required for running the Data check GUI.
4. Modify and compile the library (Unless you have already set up the library): You need to change the Data file directory from `Constants.h` in `libn3He` to appropriate directory. Before you do `make` do: `make clean` in the same directory. and then make a fresh shared binary files after you make any changes.
5. Place `.so` file under `LD_LIBRARY_PATH`: Now make sure the shared library (`libn3He.so`) file is in a directory under your `LD_LIBRARY_PATH`.

Alternatively, more convenient and professional way is as follows:

Include command in your `.bashrc` file to run `thisn3He.sh` file each time you open the terminal. i.e. include the following lines:

```
if [ -f path_to/libn3He/bin/thisn3He.sh ]; then
    . path_to/libn3He/bin/thisn3He.sh
fi
```

6. Produce binary for GUI: To produce a new binary file named `n3HeData`, go to `n3HeData` directory, open makefile and change `LIB_INCLUDE` and `GLIBS` to appropriate location for you and then do `make`. It will produce `n3HeData` binary file in the same directory.
7. Run the GUI: Now run the binary file `n3HeData` by doing `./n3HeData` from your recently compiled version in `n3HeData/`.

- Modify the `.desktop` file (included one only for Ubuntu distribution) and `.sh` file in bin directory accordingly and place it in your desktop if you want to run the GUI just by double clicking from your desktop.

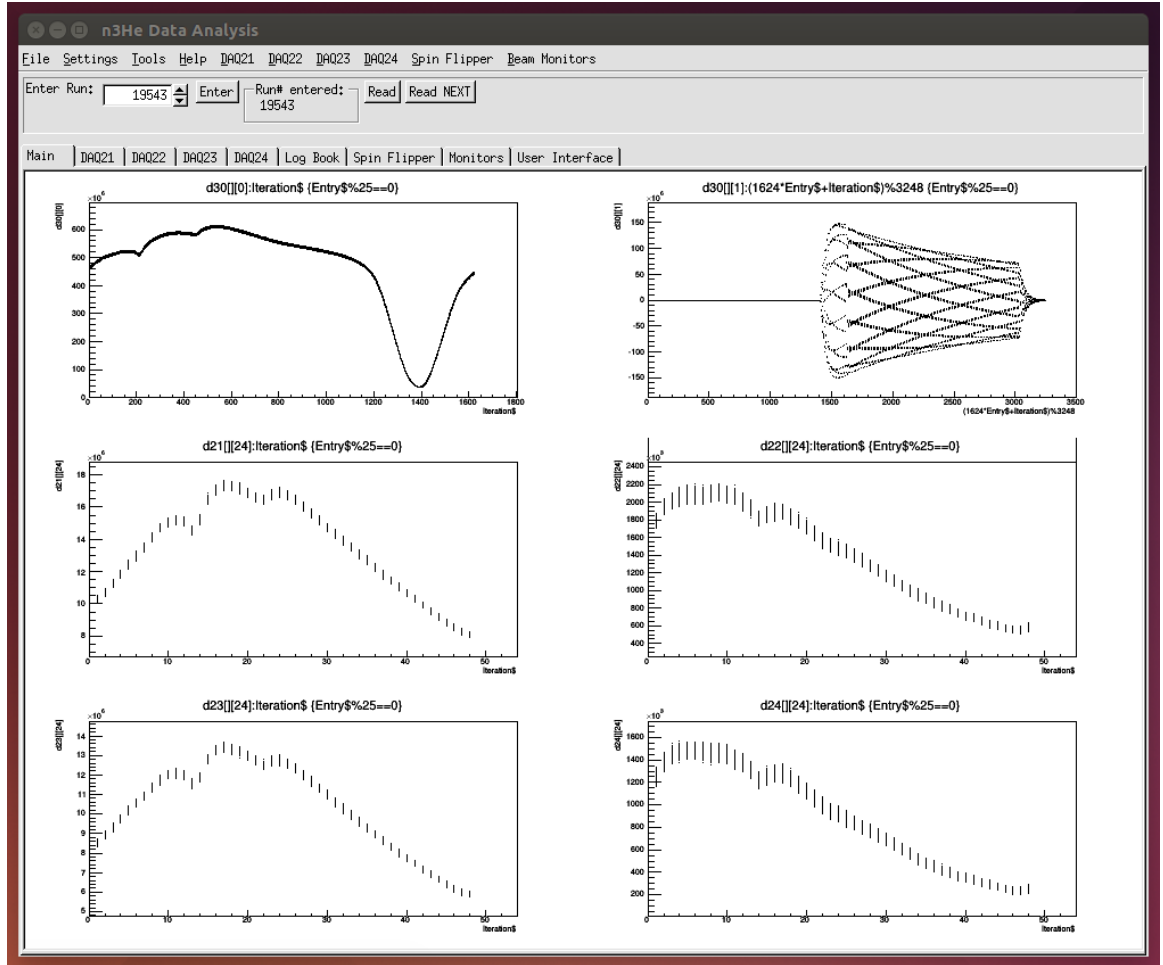


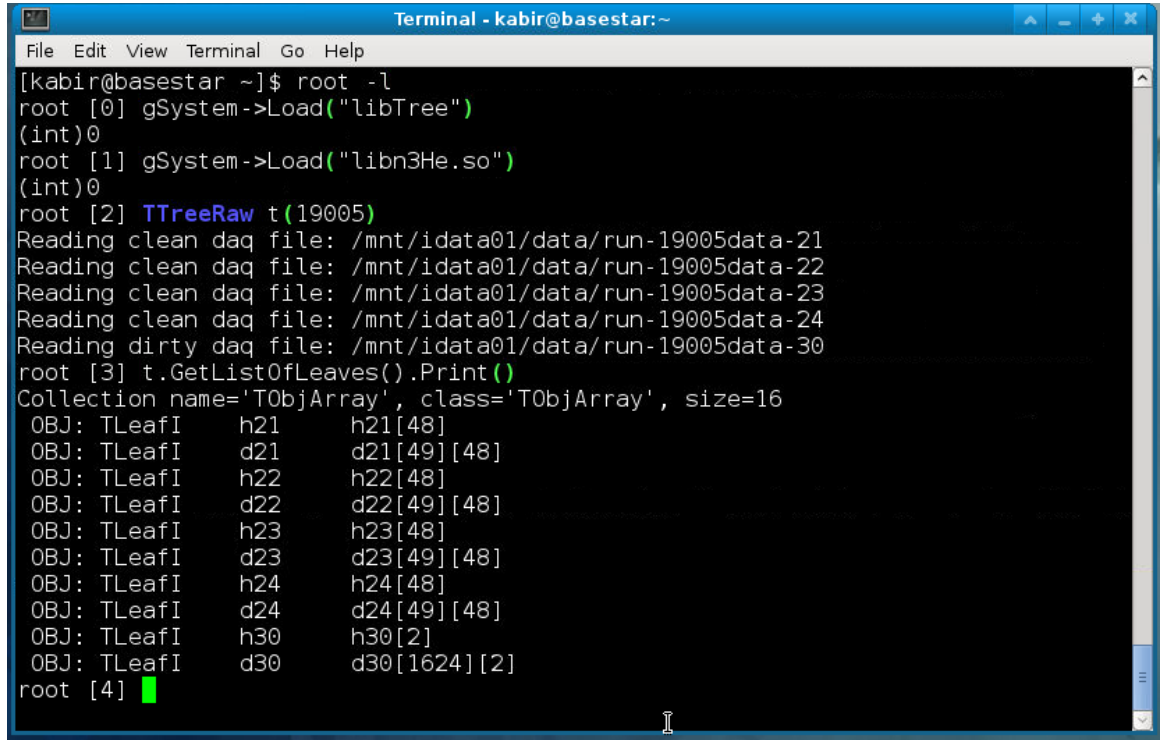
Figure 5: The n3He data browser

9 Current tree structure in n3He analysis library

Currently n3He Tree has five branches corresponding to four clean DAQ and one dirty DAQ, following is the leaf list in the n3He Tree.

```
DAQ21_LEAF "h21[48]/I:d21[49][48]/I"  
DAQ22_LEAF "h22[48]/I:d22[49][48]/I"  
DAQ23_LEAF "h23[48]/I:d23[49][48]/I"  
DAQ24_LEAF "h24[48]/I:d24[49][48]/I"  
DAQ30_LEAF "h30[2]/I:d30[1624][2]/I"
```

where h used to indicate header and d used to indicate detector signal. 21 to 24 are clean DAQs and 30 is dirty DAQ. Clean DAQs contain detector signals. Dirty DAQ (after data processing) ADC channel-0 is monitor-1 signal and ACD channel-1 is RFSF signal.

A terminal window titled "Terminal - kabir@basestar:~" showing the execution of ROOT commands to load the n3He library and print the tree's leaf list. The output shows the loading of 'libTree' and 'libn3He.so', followed by the creation of a TTreeRaw object 't' with 19005 entries. The command 't.GetListOfLeaves().Print()' is executed, resulting in a list of 16 leaves. The leaves are organized into pairs for each DAQ: (h21, d21), (h22, d22), (h23, d23), (h24, d24) for clean DAQs, and (h30, d30) for the dirty DAQ. The dirty DAQ has a much larger second branch (1624 elements) compared to the others (48 elements).

```
File Edit View Terminal Go Help  
[kabir@basestar ~]$ root -l  
root [0] gSystem->Load("libTree")  
(int)0  
root [1] gSystem->Load("libn3He.so")  
(int)0  
root [2] TTreeRaw t(19005)  
Reading clean daq file: /mnt/idata01/data/run-19005data-21  
Reading clean daq file: /mnt/idata01/data/run-19005data-22  
Reading clean daq file: /mnt/idata01/data/run-19005data-23  
Reading clean daq file: /mnt/idata01/data/run-19005data-24  
Reading dirty daq file: /mnt/idata01/data/run-19005data-30  
root [3] t.GetListOfLeaves().Print()  
Collection name='TObjArray', class='TObjArray', size=16  
OBJ: TLeafI      h21      h21[48]  
OBJ: TLeafI      d21      d21[49][48]  
OBJ: TLeafI      h22      h22[48]  
OBJ: TLeafI      d22      d22[49][48]  
OBJ: TLeafI      h23      h23[48]  
OBJ: TLeafI      d23      d23[49][48]  
OBJ: TLeafI      h24      h24[48]  
OBJ: TLeafI      d24      d24[49][48]  
OBJ: TLeafI      h30      h30[2]  
OBJ: TLeafI      d30      d30[1624][2]  
root [4] █
```

Figure 6: The n3He leaf list

Now the library always skips the first four or five events since those might

NOT be reliable. As a result the number of events in any branch for a typical n3He run is 24996 or 24995(This offset is set dynamically).Also in the analysis we recommend to skip the last event to be safe in case the last event of the run is a partial event (though every effort has been made to avoid this).

```

Terminal - kabir@basestar:~
File Edit View Terminal Go Help
OBJ: TLeafI h30 h30[2]
OBJ: TLeafI d30 d30[1624][2]
root [4] t.Print()
*****
*Tree :n3He : n3He raw data *
*Entries : 24996 : Total = 3339 bytes File Size = 0 *
* : : Tree compression factor = 1.00 *
*****
*Br 0 :b21 : h21[48]/I:d21[49][48]/I *
*Entries : 24996 : Total Size= 600 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
* ..... *
*Br 1 :b22 : h22[48]/I:d22[49][48]/I *
*Entries : 24996 : Total Size= 600 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
* ..... *
*Br 2 :b23 : h23[48]/I:d23[49][48]/I *
*Entries : 24996 : Total Size= 600 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
* ..... *
*Br 3 :b24 : h24[48]/I:d24[49][48]/I *
*Entries : 24996 : Total Size= 600 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
* ..... *
*Br 4 :b30 : h30[2]/I:d30[1624][2]/I *
*Entries : 24996 : Total Size= 600 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
* ..... *
root [5]

```

Figure 7: The n3He tree structure

10 Sample analysis

Sample online analysis on basestar:

```
//OnlineAnalysis.C
//Demo Online Analysis using n3He Library.(By Online I mean 'from
    CINT, doing analysis on the fly, less thoughtful but preferred
    in some conditions')
//Author: Latiful Kabir
//Date: 12/23/14

void OnlineAnalysis()
{
    gSystem->Load("libTree"); //You need to load libTree first in
        order to Load libn3He. This is not necessary if you include
        TTree.h
    gSystem->Load("libn3He.so");

    TTreeRaw *t=new TTreeRaw(19900);
    t->Draw("d21[] [0]:Iteration$");
}
```

This script when you run using `root -l OnlineAnalysis.C` will produce the following output:

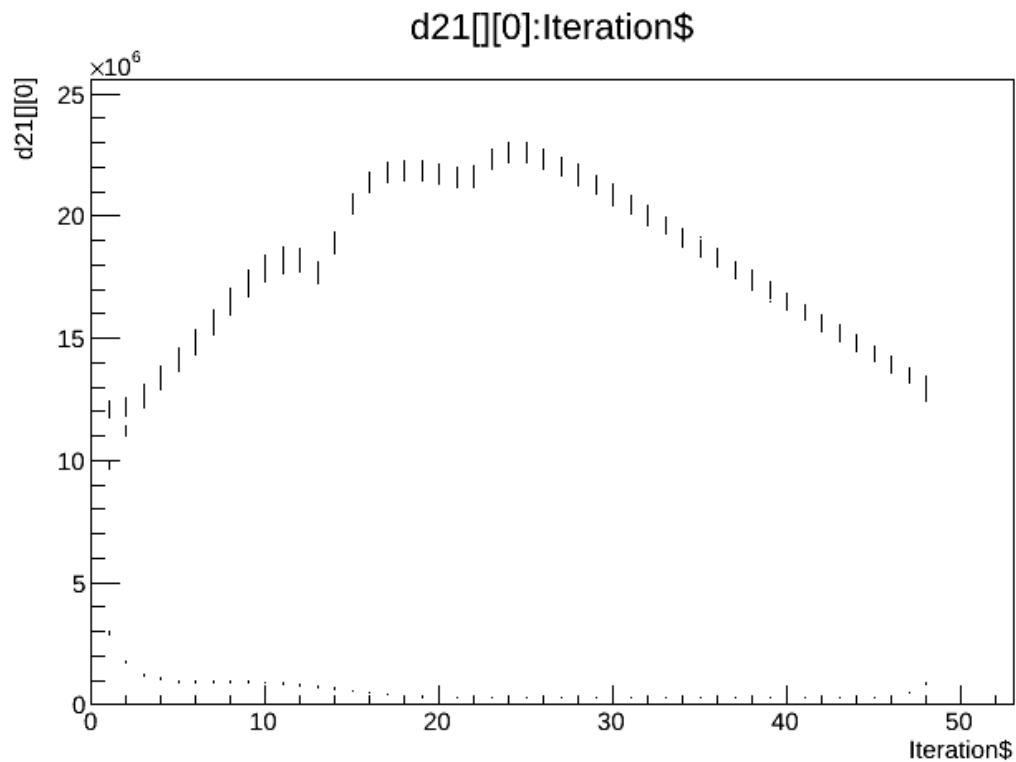


Figure 8: The Output from OnlineAnalysis.C

Sample offline analysis on basestar:

```
//OfflineAnalysis.C
//Demo Offline Analysis using n3He Library.(By Offline I mean 'in
    a script more thoughtful and serious analysis unlike from CINT)
//This script shows how to accress Tree using SetAddress
// and plots only the all event/pulses of channel-0
//Author: Latiful Kabir
//Date: 01/14/15
//This is the fastest and most preferred method for reading Tree

#include<TTree.h>
#include<TBranch.h>
#include<TGraph.h>
```

```

void OfflineAnalysis(){

    //Load the library unless loaded automatically by ROOT
    gSystem->Load("libTree");
    gSystem->Load("libn3He.so");

    //Create a TTreeRaw object with desired run number
    TTreeRaw *t=new TTreeRaw(17900);
    t->Print(); // Print to see what's inside the Tree
    int ch=0; //Channel to analyze

    //Create a struc buffer to keep your events
    struct myData
    {
        int header[48];
        int det[49][48];
    };

    myData md;

    //Get the branch you want to analyze
    TBranch *b=t->GetBranch("b21");
    b->SetAddress(&md.header[0]);

    //-----

    TGraph *g=new TGraph();

    //Loop through all the events in the run.
    for(int i = 0; i < b->GetEntries(); i++)
    {
        //Load the samples for a event/pulse in buffer
        b->GetEntry(i);

        //Loops through the sample for the loaded event
        for(int k=0; k<49; k++)
            g->SetPoint(i*49+k, i*49+k, md.det[k][ch]);
    }

    g->Draw("AP");
}

```

```
delete t;  
}
```

This script when you run using `root -l OfflineAnalysis.C` will produce the following output when zoomed in:

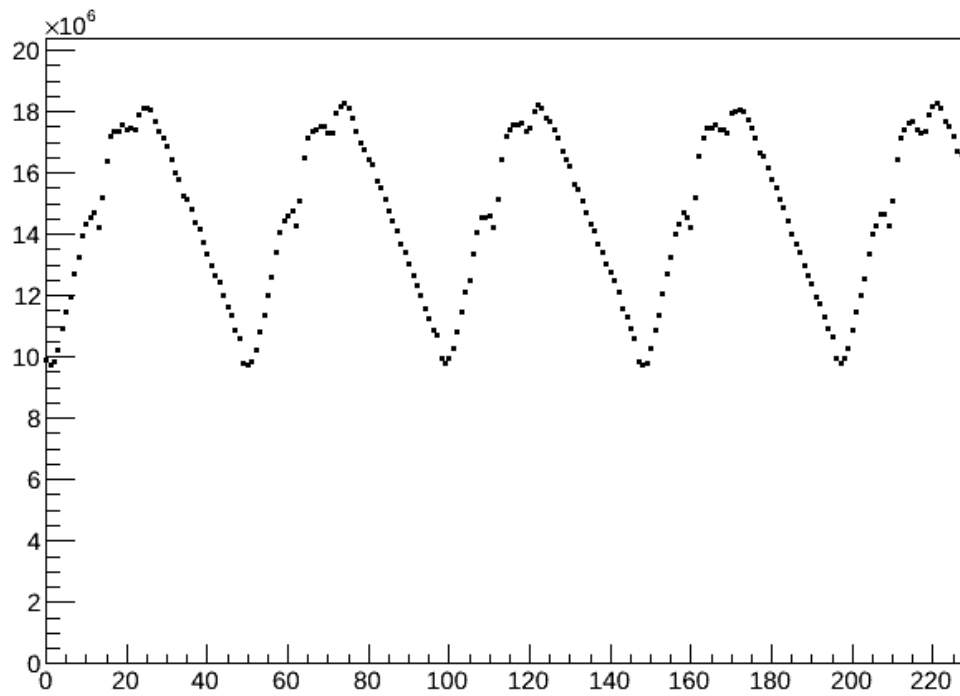


Figure 9: The output from OfflineAnalysis.C

11 Reference for TTreeRaw class

The base Class is TTree.

As a result all data and member functions from TTree are automatically inherited. For a list of TTree data and member functions go here

public:

```
TString dataPath;
TString *DaqLeaf;
TString *dataFile;
static int module[5];

TBranch *b21,*b22,*b23,*b24,*b30;

void Init(int runNumber);
TTreeRaw(int runNumber);
~TTreeRaw();
```

This list will be updated gradually as new functionality is added to the TTreeRaw class.

12 The summary root file

For each GOOD run (which is analyzable for either LR or UD asymmetry), a summary file is processed which has time bin integration (sum of signals) of each event both for detector channels(clean DAQ channels) and dirty DAQ channels (M1 signal and RFSF current). Also a calculated asymmetry for pair of adjacent events of each detector channel is computed. All these values are saved in tree structure in a root file in the directory /mnt/idata05/summary/ on basestar. The tree has three branches : sumc (sum of signals from clean DAQ), sumd (sum of signals from dirty DAQ) and asym (calculated asymmetry).

sumc[4][36]:

The first index is for DAQ (0:DAQ21, 1:DAQ22,2:DAQ23,3:DAQ24) and second index is for each channel (Note the channel index is continuous here).

sumd[2]:

sumd[0] is time bin integration of M1 signal and sumd[1] is the same thing for RFSF current.

```

root [0] TFile *f=new TFile("/mnt/idata05/summary/run-20100.root")
root [1] TTree *t=(TTree*)f->Get("T")
root [2] t->Print()
*****
*Tree      :T              : My n3He Tree                                     *
*Entries   :    24991      : Total =          58071562 bytes  File Size =   54348009 *
*          :              : Tree compression factor =    1.07                    *
*****
*Br       0 :sumc          : sumc[4][36]/D                                   *
*Entries   :    24991      : Total Size=    28835088 bytes  File Size =   26240846 *
*Baskets   :      513      : Basket Size=    4766720 bytes  Compression=    1.10    *
*.....*
*Br       1 :sumd          : sumd[2]/D                                       *
*Entries   :    24991      : Total Size=     401040 bytes  File Size =     320375 *
*Baskets   :       10      : Basket Size=     66048 bytes  Compression=    1.25    *
*.....*
*Br       2 :asym          : asym[4][36]/D                                   *
*Entries   :    24991      : Total Size=    28835088 bytes  File Size =   27778466 *
*Baskets   :      513      : Basket Size=    4766720 bytes  Compression=    1.04    *
*.....*
root [3] █

```

Figure 10: The summary root file tree structure

Each root file(for each run) has 24991 entries as some events at the beginning and end of the run are skipped.

asym[4][36] :

The first index is for DAQ and second index is for channels. Now the first entry of the branch “asym” is a flag which is the run number. second entry is asymmetry between first event and second event of the run, third entry is asymmetry between second and third event of the run and so on.

0:run number 1:(0,1) 2:(1,2) 3:(2,3) 4:(3,4)

So in the analysis the first entry always to be skipped (since its a flag for run number) and only either odd entries or even entries to be considered (to avoid double counting or correlations) of the “asym” branch (in addition to the imposed cut).

In calculating the above asymmetry, the signal of each channels for each event is normalized by the sum over all the detector signals to get rid of beam fluctuation.

The asymmetry is constructed using :

$$(SF_{on} - SF_{off}) / (SF_{on} + SF_{off})$$

which corresponds to neutron spin pattern:

$$(Spin_{down} - Spin_{up}) / (Spin_{down} + Spin_{up}).$$

For details check the source code.

13 Individual run status

I used my analysis library to create a list for individual run status.

The complete run list can be found on basestar in the file:
/mnt/idata05/summary/runList.txt

For UD target position runs, individual run status can be found on basestar in the file:
/mnt/idata05/summary/runListUD.txt

For LR target position runs, the individual run status can be found on basestar in the file:
/mnt/idata05/summary/runListLR.txt

In the run list each run is checked for the following run quality :

- DAQ header/trigger issue (error code -1)
- No beam or partial beam (error code -2)
- Branches having different number of entries (error code -3)
- Short runs (error code -4)
- All DAQs are NOT synchronized. (error code -5)
- No or partial data files on basestar. (error code -6)
- Wrong chopper phases. (error code -7)
- Wrong RFSF phases (error code -8)
- High voltage turned off (error code -9)

If any run apparently passed all the above checks, then the run is deemed as GOOD (error code 1). GOOD runs with error code 0 means , the last event of the run is incomplete and the last event needs to be thrown away for that run in the analysis.

Note the list assumes using libn3He where I fix the header/trigger issues of many runs just right in the library.

14 Reference for run numbers

.....LR Orientation, NPDG Chopper Settings.....

- Actual n3He data with full run length(25000 T0 per run) starts from 14586.
- Left Right(PC) Asymmetry Data : Run numbers with beam before 16390 (exclude chopper tuning, polarimetry, beam scan, collimation data runs)
- Run range used so far for PC asymmetry analysis :14785 - 15785
- Chopper Tuning Runs : 15867 - 15924

.....UD Orientation, n3He Chopper Settings.....

- Instrumental Asymmetry Data : 17357 - 17889 , and many more run sets
- U/D Asymmetry Data : Starts from run number 17400 and continues as we take data. For the Up-Down analysis it is recommended to use run numbers after 18000 since we were still testing and changing things before that.The last UD run recorded to be run number 57177.
- The stable optimization of n3He library was done for run numbers starting from 17400.
- Polarimetry Data : Consecutive short runs are generally polarimetry data. It is generally taken on Wednesday of first week of the month.
- Hi and low beam power runs: Runs before April 25 are 850KW and after that are 1.2MW for a while then its again around 850KW. Thus run numbers around 27000 - 47000 has beam power 1.0- 1.3 MW. Run numbers around 28500 has maximum. All other runs have around 850MW. Again at the end of beam cycle-2, starting from Nov 18th (run numbers 54846 - on words) the beam power starts to go up (up to 1.4MW).(The run numbers are approximate only)
- Second Collimation scan runs: 37987-38022

- The last UD run number 57177 (on Nov 30 before rotating target and other scans).
- HV scan runs 57177- 57201
- Thin slit scan 57204 - 57218

.....**LR Orientation, Various Collimations**.....

- Run numbers starting 57403 till the end of run, all are LR runs.
LR asymmetry with 8 x 10 cm collimation: 57403 -57796, 6 x 10 cm collimation: 57797-58197, 4 x 10 cm Collimation: 58199 - 58535.

.....**LR, Various IC Pressure and Collimations, NPDG Choppers**.....

All the following runs here after are in tighter choppers setting (The NPDG chopper settings runs)

- Tighten Chopper setting(Ch-1:15.3109ms CW, Ch-2: 0.133ms CW)
LR runs with 8 x 10cm collimation: 58613 - 59808.
- Second (and complete) HV scan : 59810 - 59853
- Hi resolution scan (All DAQs running at 100KHz, event length is 1624 tbin plus 1 header) : 59860 - 59875.
- Ion Chamber pressure modified to 1 atm: M1 signal flips(this is true for rest of the runs) and one preamps signals hits the saturation point after changing the IC pressure. Also last half of the chamber mostly shows very weak noise like signal at 1 atm IC pressure. 8x10 cm collimation Run numbers: 59980 - 60314, 6x10 cm collimation runs: 60314 - 60503.
- Ion Chamber pressure 0.25 atm and 8x10 cm collimation : 60504 - 60700, 6x10cm collimation runs: 60709 -60869, 4x10 cm collimation runs: 60871 - 61009.

.....**Run Numbers for Various Scan**.....

- Check the text file in this link

Note: This list is NOT complete. For a complete list or reference work out through the paper log book.

15 Reference for beam centroid

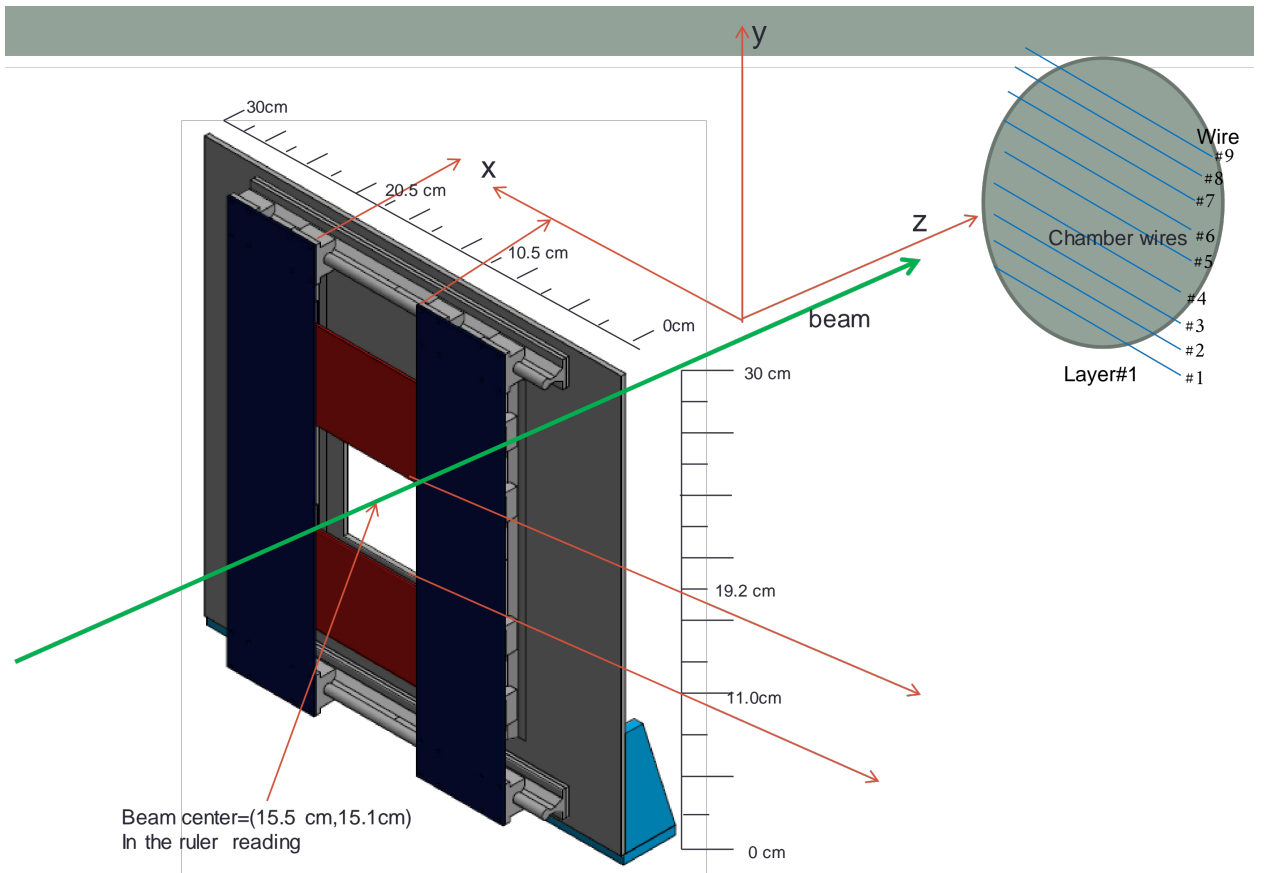


Figure 11: n3He Run Statistics

16 Summary of n3He data

Run Range: 14785 to 61000

Total Runs on tape(on basestar) : 46217

Number of GOOD Runs with beam: 34754

Number of runs with partial last event :902 (Still GOOD if last event is skipped)

Number of runs with header/trigger issue : 838

Number of runs with partial beam or NO beam :6897

Number of short runs : 2602

Number of runs with synchronization issue : 56

Number of runs with NO data files on basestar: 107

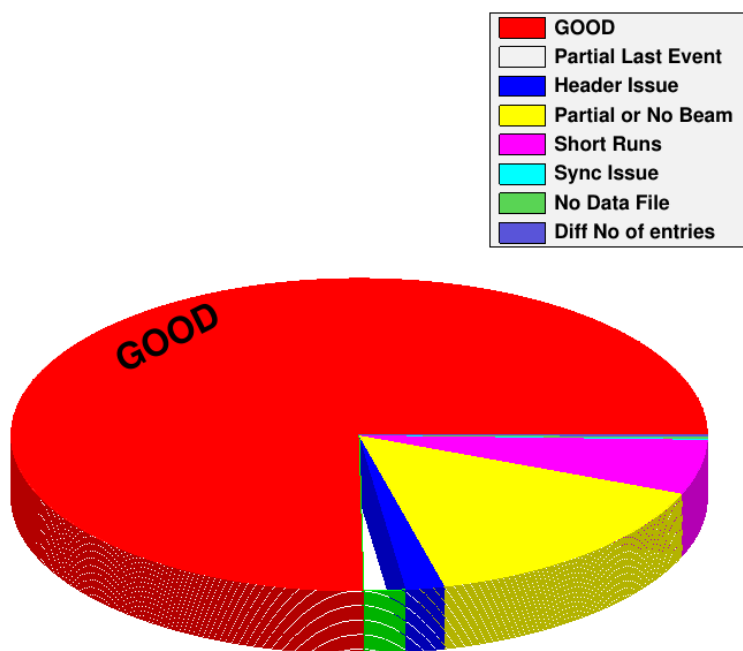


Figure 12: n3He Run Staistics

17 List of n3He parameters

Chopper 1 Phase : 14.045 CW

Chopper 2 Phase : 0.245 CCW

T0 Delay :13.5 ms

DAQ dead time : 0.96 ms

Distance between moderator and detector front face : 18 m (approx)

Vertical B-field(Guide field) value : -9.14 ± 0.02 Gauss.

Number of time bin per event for detector data : 49

Number of time bin per event for M1 data : 1624

Width of each detector data time bin : 320μ sec.

Width of each M1 data time bin : 10μ sec

Each run length :

25000 T0 (events) (in raw file)

24996 ± 1 T0 (events) (when loaded in TTreeRaw)

Pressure in the chamber : 0.5 atm (approx)

Opening for Collimator :

X-Axis : 10 cm

Y-Axis : 8.2 cm

Number of detector wires per layer : 9

Number of detector layers : 16

Others to be included gradually

