An English guide to experiments

Heavy Ion Nuclear Physics Laboratory

GIBELIN Julien

March 29, 2010

Forewords

This simple help has been created as an introduction for analysis at RIKEN CYCLOTRON and especially RIPS (E6 room), the RIKEN Projectile fragment Separator. The author hopes than incomers in RIKEN CYCLOTRON will find help basis during the course of experiments.

Writing conventions are :

`< >` compulsory parameter

`{[ ]}` optional parameter

`/` choice

`->` ‘‘ ’’ keyboard input

Examples are signaled by ♢♢.

Please send all comments, bugs... to the author :
gibelin@rarfaxp.riken.go.jp

A *pdf* version of this booklet is available for downloading at:
http://rarfaxp.riken.go.jp/~gibelin/riken_guide.pdf
A *html* version of this booklet is also available at:
http://rarfaxp.riken.go.jp/~gibelin/riken_guide/

Last version: March 29, 2010

Acknowledgments

Especially to the author of ANAPAW & the data acquisition system used by the Heavy Ion Laboratory (Heavion): S. TAKEUCHI & H. BABA.

My others colleagues played a great role in this guide by correcting my numerous English mistakes !

Merci !

J. GIBELIN

PS: Almost 5 years later I am back to RIKEN for an experiment, so I took this opportunity to update this manual, with the help of my colleagues from LPC Caen.

Contents

Chapter 1

Data Acquisition System

Data Acquisition System system for Heavy Ion Nuclear Physics Laboratory has been mainly supervised by H. Baba. For more details refer to his home-page (in Japanese): http://daq.rikkyo.ac.jp/babarl/

1.1 Basis of DAQ system in Riken

For DAQ, both Wme and Camac are used. Below is a simplified scheme[1].

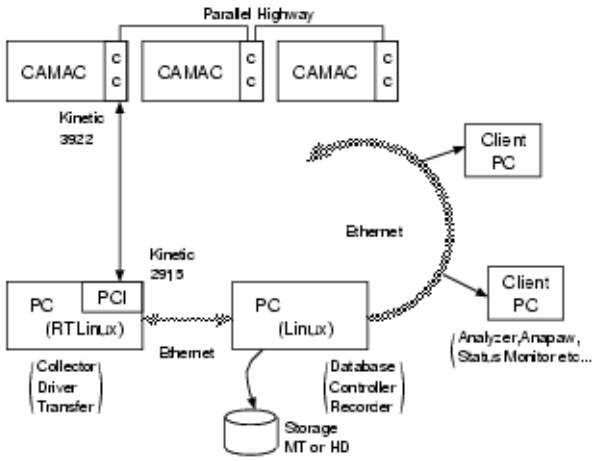


Figure 1.1: Typical configuration of BabarlDAQ. Some client PCs can lie on the Ethernet

Here, our concern is not on how data are read from electronic module but mainly on how we can *monitor* them. So as let's describe three main programs : Bbcaenhv (see Section 1.2) which allows you to read and set high voltage

[1]Extract from *Development of New Data Acquisition System BabarlDAQ*, RIKEN Accel. Prog. Rep. 34 (2001)

of some detectors, BABARL which allows you to launch DAQ and SSDHV for monitoring silicons H.V.

First, let's give name to current main computers[2] :

rips1 controller

ripscc CAMAC control

rips00 analysis

bbhv H.V. control

Finally, you can find in the miscellaneous part (see Section 5.3), the complete description of data structure.

1.2 Bbcaenhv

Allow you to read and edit high voltage power supply values. You must first go to the H.V. computer control (mainly `bbhv` , ask for password).

1.2.1 Header overview

`CRATE ## PAGE #/#` **C**`rate` **P**`age` **E**`dit` **L**`oad` **S**`ave` **O**`n/`**O**`ff` **U**`pdate` **M**`onitor` **Q**`uit`

1.2.2 Command description

Crate

Change/read **c**rate number. Keyboard command : `C`

Edit

Edit selected field. Keyboard command : `E`

Load

Load H.V. configuration's file. Keyboard command : `L`

Save

Save H.V. configuration's file. Keyboard command : `S`

On/Off

Turn **O**n/**O**ff the HV for selected device. Keyboard command : `O`

Update

Update values. Keyboard command : `U`

[2] Note : these appellations might alter from time to time...

Monitor

Update values every 10 seconds. Keyboard command : M

Quit

Exit BBCAENHV. Keyboard command : Q

1.3 BabarlDAQ program's commands

1.3.1 MOUNT

Mount physically the tape on hardware.

```
Command> MOUNT
```

1.3.2 MTON

Mount the **t**ape **on** software.

```
Command> MTON
```

1.3.3 HDON

Use **h**ard **d**rive to write **on**.

```
Command> HDON
```

1.3.4 WTH

Write **t**he **h**eader of the next run.

```
Command> WTH
```

1.3.5 START

Start a saved run.

```
Command> START
```

1.3.6 NSSTA

Non **s**aved **sta**rt.

```
Command> NSSTA
```

1.3.7 STOP

Stop the current run and ask for lender.

```
Command> STOP
```

1.3.8 PRISCA

Print the CAMAC **sca**ler values.

```
Command> PRISCA
```

1.3.9 MTOFF

Un**m**oun**t** the tape **off** from software, swapping automatically to hard drive (see Section 1.3.3).

```
Command> MTOFF
```

1.3.10 DISMOU

Unmount/eject the tape from hardware.

```
Command> DISMOU
```

1.3.11 PRIHEADER

Print the **hea**der and the ender of a raw data file.

```
Command> PRIHEADER <FNAME>
```

1.3.12 SETNUMBER

Change run number (next RUN is *set number + 1*).

```
Command> SETNUMBER <NUM>
```

1.3.13 Main procedure

◇◇If you want to make a non recorded acquisition :

```
Command> nssta
```

◇◇If you want to make a recorded acquisition :

```
Command> mount
```

Mount physically the tape on hardware.

```
Command> mton
```

Mount the tape, case of software

```
Command> wth
Header :
         -> `` Bonjour ceci est une acquisition ''
Command> start
```

. . . And wait for you data. . .

```
Command> stop
Ender :
         -> `` Ceci etait une acquisition ''
```

Stop the acquisition and write the ender

```
Command> prisca
```

...and don't forget to print the scaler values *!* After that you can restart another acquisition. If you want to change the tape.

```
Command> mtoff
Command> dismou
```

1.4 BabarlDAQ Emergency Stop

- End the program using `Ctrl+C`
- In *rips1* shell, list process using `ps x`.
- ...
- Change the console by using `Alt+F2`
- In
- ripscc shell, run `ps ax`. Beware that this command will list all processes in the terminal and is thus non appropriated on *rips1* You should then see `bbcamac`, `collector`, and `transfer`. Kill only `transfer` while logged as root.
- Run `/sbin/lsmod`. You should see a module named `babarldaq`. Kill it by running `/sbin/rmmod babarldad`, as well as all other `nddq*` or `bb-*` modules.
- ...
- Go back to the first console: `Alt+F1`
- Run `netstat -a | less`. Look for any '167.***' tcp number. ?? If processes remain `TIME_WAIT` for more than 1 minute, this means that they (and especially `transfert`) remain in `ripscc` or rips1, so please go back and kill them.
- ??

To restart

Log to

SSDHV allows you to monitor and control HV crate for Silicon detectors. Before lunching SSDHV program you should ssh to `gomigo@gomigo` (ask for the password) and then lunch SSDHV.

1.5 SSDHV

Monitor for silicon H.V. Lunch with `ssdhv`, login `ssdhv`.

1.5.1 GETFILE

Get a status file

```
SSDHV :  GETFILE <FNAME>
```

1.5.2 PUTFILE

Put status into file

```
SSDHV :  PUTFILE <FNAME>
```

1.5.3 SET

Set voltage (in V), current (in nA) for a specified channel, increasing the voltage with a specified speed (V/s).

```
SSDHV :  SET <CHANNEL> <VOLTAGE> <CURRENT> <SPEED>
```

1.5.4 GET

Get/update information on the SSD status.

```
SSDHV :  GET [CHANNEL]
```

1.5.5 CTRL

```
SSDHV :  CTRL <ON/OFF>
```

1.5.6 ZERO

Set the high voltage to zero for *all* channel. Slowly of course *!*.

```
SSDHV :  ZERO
```

1.5.7 NAME

Give a channel a name. Only visible after updating (see Section 1.5.4)

```
SSDHV :  NAME <CHANNEL> [CHNAME]
```

Warning: no possible space, example :

```
SSDHV :  name 1 ssd_1-1
```

1.5.8 MONITOR

Update values every 10 seconds.

```
SSDHV :  MONITOR <ON/OFF>
```

1.5.9 PRINT

Print SSD status.

```
SSDHV :  PRINT
```

1.5.10 EXIT

Exit.

```
SSDHV :  EXIT
```

1.5.11 STAT

Print out statistics on a file.

```
SSDHV :  STAT <FNAME>
```

1.5.12 VIEW

View the contents of a file.

```
SSDHV :  VIEW <FNAME>
```

1.5.13 HELP

View help.

```
SSDHV :  HELP
```

1.5.14 Unix valuable commands

You can use `LS`, `CD` & `PWD`.

Chapter 2

Anapaw

ANAPAW , made by S. TAKEUCHI, combines analysis (in real time) with PAW. For further informations, please consult his site (in japanese) :
http://www.ne.rikkyo.ac.jp/~takeuchi/

2.1 Install Anapaw

2.1.1 Anapaw versions

Because ANAPAW uses CERN Paw and CERNLIB, you should first correctly install this last softwares, see
http://it-div.web.cern.ch/it-div/physicscomputing/

Two versions exist : the latest and fastest is
http://rarfaxp.riken.go.jp/~takesato/anapaw/Download/download.html whereas the oldest one is
http://rarfaxp.riken.go.jp/~takesato/anapaw/Download/download.html, the main difference lies in the way the software deals with memory.

The analysis part is possible thanks to software named ANALYS. Basic source codes should be download for it. The latest version uses :
http://rarfaxp.riken.go.jp/~takesato/anapaw/Download/download.html, the other :
http://rarfaxp.riken.go.jp/~takesato/anapaw/Download/download.html

2.1.2 Before installing

Warning: ANAPAW commands have been written using C SHELL, thus you should swap to this shell.

Edit or create the `.cshrc` file at the root of your home directory.

If we suppose that the ANAPAW source files are in `~/usr/anapaw` and you want your work directory to be `~/exp/r300n`, add the following:

```
#ANAPAW
alias analogin 'source $HOME/usr/anapaw/Setup/setupanapaw/'
setenv ANAPAW_WORK  $HOME/exp/r300n
```

Then for this change to be effective :

```
source .cshrc
```

ANAPAW uses a set of environment variables, contained in the `setupanapaw` file. At least two of them must be changed to fit to your computer's configuration. First, as far as ANAPAW deals CERN's PAW - and other libraries - specify the location of CERNLIB. In second the ANAPAW source files location itself should be specified.

As an example, the first lines of `setupanapaw` can look like (defaults):

```
setenv CERNLIB      /cern/pro/lib
setenv ANAPAW_HOME  $HOME/usr/local/anapaw
```

Off line analysis

ANAPAW can be use for on-line analysis as well as off line analysis.

The suitable configuration for local and off-line is :

```
# For using on local.
source $ANAPAW_SETUP/setup.local
```

You must create work directory, for examples, if you want to to work in `~/exp/r300n`, you must create : `~/exp`. Or if you want `~/exp/r300n/offline` then create `~/exp/r300n`

Finally copy source files (basic ones are in `usersrc-*.tar.gz`) to e.g `~/exp/r300n/src` (or `~/exp/r300n/offline/src`). This can be off course change in the `setupanapaw` file (see Section 2.1.2).

On line analysis

For the on-line case, more than one user can make analysis, but everyone must use, at the beginning, the same source code. Then if the main directory will be `~/exp/r300n` :

```
> makedir ~/exp/
> makedir ~/exp/r300n
> makedir ~/exp/r300n/src
> makedir ~/exp/r300n/users
```

The `analogin` (see Section 2.1.3) commands will automatically create users files.

2.1.3 How to log in user directory

Off line analysis

After definition of main parameters in `setupanapaw` (see Section 2.1.2) you simply have to typewrite :

```
> analogin
```

On line analysis

Due to several user, you will have to specified your login. ♢♢For example for the user named *hogehoge*.

```
> analogin

    == ANAPAW_SETUP ==

    Please Input Your Name.

    -> 'hogehoge'
```

Then if non directory exists for you, `analogin` will create it.

2.1.4 Compiling Anapaw

You first have to build the library, independently of the characteristics of the experiment.

Supposing that the ANAPAW_SOURCE variable has been assigned by the 'analogin' program (see Section 2.1.2). Just type :

```
> make -C $ANAPAW_SOURCE clean
> make -C $ANAPAW_SOURCE
> make -C $ANAPAW_SOURCE install
```

This series of commands can be simplified by using

```
> makelib
```

Then you must build the specific encoding files for the analysis. After changing to the directory where the user source files are (example : `~/exp/r300n/src`), type :

```
> make clean
> make
```

Everything can be simplified by using (from anywhere)

```
> makeana
```

2.2 Software

2.2.1 Login

Simply typewrite `analogin` and follow the instruction. Most of time, except when you want to create a personal ANAPAW setup, your login is `rips`.

Note that you can run a kumac in batch using `-b` argument, followed by the macro's name

```
rips@host > analogin
```

2.2.2 Anapaw structure

The purpose here is not to explain in detail the ANAPAW structure, but to give a rough idea.

2.3 Commands

2.3.1 /ANALYS/

/ANALYS/LOOP

Event loop shell : this command allows you to read raw data, so as to analyze them. You can do it OFF-LINE or ON-LINE.

```
ANAPAW> ANA/LOOP [ FNAME CHOPT ]

        FNAME : ' ' On line reading
                Analys filename
        CHOPT : ' ' Command mode
                B  Batch mode
```

◇◇Let's start with an example of simple OFF-LINE analysis, using the raw data contained in **run1018.rdf** and the analysis command of **sample.ana**

```
ANAPAW> book sample.ana
```

Read the analysis command from **sample.ana**, see Section 2.3.2

```
ANAPAW> hst1
 HST1> (=0)

 1D-Histograms:

  1 ( HID :  101 ) ( TITLE : ch1                 ) ( GATE :   0 )
    ( bin :  200 ) ( xmin :     0.00 ) ( xmax :  2000.00 )

  2 ( HID :  102 ) ( TITLE : ch2                 ) ( GATE :   0 )
    ( bin :  200 ) ( xmin :     0.00 ) ( xmax :  2000.00 )

  3 ( HID :  103 ) ( TITLE : ch3                 ) ( GATE :   0 )
    ( bin :  200 ) ( xmin :     0.00 ) ( xmax :  2000.00 )

  4 ( HID :  104 ) ( TITLE : ch4                 ) ( GATE :   0 )
    ( bin :  200 ) ( xmin :     0.00 ) ( xmax :  2000.00 )
```

Default *1D*-histogram proposed, see Section 2.3.2

```
ANAPAW> ana/loop run1018.rdf
```

Open the data file **run1018.rdf**

```
ANAPAW/EVTLOOP>
ANAPAW/EVTLOOP> start
```

Start reading the values.

```
  ANAPAW-M : Interrupt Event Loop!
ANAPAW/EVTLOOP>
```

This message appears when hitting ↩, so as to stop the reading.

```
ANAPAW/EVTLOOP> scat
 SCAT> on
ANAPAW/EVTLOOP>
```

This allows you to open a scatter histogram window. Especially useful to view on time 2D histograms.

```
ANAPAW/EVTLOOP> quit
```

Quit *!*

◇◇How to make ONLINE analysis *?* Simply write:

```
ANAPAW> ana/loop
```

/ANALYS/STATUS

```
ANAPAW> ana/status

  ANAPAW-M : Current Status.

  Online Mode

  Blocks       :  0
  Valid Events :  0

  RDF :
  ANA : ana/gcheck.ana

  Analyzer :    1
  Analyzer :    3


  Now on Main Prompt.
```

/ANALYS/RDMP

Store the raw data to a file with a given gate. For gate help see Section 3.4. Note that the file has to be opened before any kind of storage !

```
ANAPAW> rdmp [ OC filename GID ]

        OC        : O Open file
                    C Close file
filename  : Raw data file name
GID       : Gate IDentifier
```

♢♢Example :

```
ANAPAW> rdmp O test.rdf
ANAPAW> ana/loop
```

And after taking data ...

```
ANAPAW> rdmp C
```

/ANALYS/ANTUPLE

Store the raw data to N-tuple files with a given gate.

```
ANAPAW> antuple [ OC GID RunNum ]

        OC        :  O Open file
                     C Close file
        GID       :  Gate IDentifier
        RunNum    :  Run number
```

Warning: Please note that this function needs special haking in `add_ntuple.f` file, see Section 4.3 for more details.

2.3.2 /ANALYS/DEFINITION

/ANALYS/DEFINITION/BOOK

Open analysis input file and book histograms. This procedure must be call before taking or reading data.

```
ANAPAW> book [ FNAME ]

        FNAME : File name
```

/ANALYS/DEFINITION/CLEAR

Clear all histogram definitions, histograms and event summary.

```
ANAPAW> clear
```

/ANALYS/DEFINITION/ANAADD

Add analysis parameter.

```
ANAPAW> anaadd [ filename ]
```

♢♢Here under are simple commands for anaadd

```
ANAPAW> anaadd test.ana
FileName :
test.ana

 Type 'AQ' to quit ANAADD mode.

Analys Com >
Analys Com >del
```

Delete last line.

```
Analys Com >aq
```

Exit from editor, save file and update.

/ANALYS/DEFINITION/AWRITE

Write analysis parameters to a given filename.

```
ANAPAW> awrite [ filename ]
```

◇◇Example:

```
ANAPAW> awrite anacode.ana
```

/ANALYS/DEFINITION/CHGATE

Change definition of given gate, as describe in ANA-CODE. You can change definition of [GATE], [AND] and [OR].

```
ANAPAW> chgate [ GID ]

        GID       :  Gate IDentifier in ANA-CODE
```

◇◇Example of a simple GATE and a gate using *2* gates logically linked by OR.

```
ANAPAW> chgate 1

 Kind : GATE

     ( GID :   1  ) ( KIND : Gate )
     ( VALUE :   1   1   1   4  ) ( Limit :     0.50    1.50 )
     ( TOTAL ENTRIES :          0 ) ( ACCEPTED ENTRIES :          0 )


 Analyzer > (=1)
 Word1    > (=1)

 ANAPAW> chgate 11
 Kind : AND

     ( GID :   11  ) ( KIND : And  )
     ( ACCEPTED EVENTS :        0 )

 ELEMENTS :
 9 10

 Elements > (<CR>=No Change)
```

/ANALYS/DEFINITION/CHHST

Change definition of a given histogram, as described in ANA-CODE, see Section 3. For *1D* histograms see Section 3.5.1, and for *2D* histograms see Section 3.5.3.

```
ANAPAW> chhst [ HID CHOPT ]

        HID   : Histogram ID to change a definition.
        CHOPT : Options
                ' ' = Change a definition by Command Line.
                 X  = Change a X-Range on X-window by mouse.
                 Y  = Change a X-Range on Y-window by mouse.
                 T  = Change a XY-Range on X-window by mouse.
```

♢♢Example:

```
ANAPAW> chhst 287

 1D-Histograms:

    ( HID :  287 ) ( TITLE : Multiplicity         ) ( GATE :     0 )
    ( bin :   11 ) ( xmin :    -0.50 ) ( xmax :    10.50 )
    ( VALUE :   4 200 200  2 )

 GateID   > (<CR>=0)
 Analyzer > (<CR>=4)
 ID 1     > (<CR>=200)
 ID 2     > (<CR>=200)
 Word     > (<CR>=2)
 Bin      > (<CR>=11)
 Min      > (<CR>=-0.5)
 Max      > (<CR>=10.5)
 Title    > (<CR>=Multiplicity)

ANAPAW> chhst 1 x
```

/ANALYS/DEFINITION/HST1

Define ZONE ,IDs of *1D*-histograms to see. By defaults ($\boxed{\hookleftarrow}$ press or value $\boxed{0}$ given) the output is created with the first *4* histograms disposed 2×2

```
ANAPAW> hst1 [ NX NY ]

        NX : Number of divisions along X
        NY : Number of divisions along Y
```

♢♢Example of default result :

```
ANAPAW> hst1
 HST1> (=0)

 1D-Histograms:
```

```
( HID :  101 ) ( TITLE : ch1                 ) ( GATE :     2 )
( bin :    2 ) ( xmin :    -0.50 ) ( xmax :     1.50 )
( VALUE :   1   1   1  5 )

( HID :  102 ) ( TITLE : ch2                 ) ( GATE :    -2 )
( bin :    2 ) ( xmin :    -0.50 ) ( xmax :     1.50 )
( VALUE :   1   1   1  5 )

( HID :  103 ) ( TITLE : ch3                 ) ( GATE :     0 )
( bin :   11 ) ( xmin :    -0.50 ) ( xmax :    10.50 )
( VALUE :   3 100 100  2 )

( HID :  104 ) ( TITLE : ch4                 ) ( GATE :     3 )
( bin :   11 ) ( xmin :    -0.50 ) ( xmax :    10.50 )
( VALUE :   3 100 100  2 )
```

♢♢Example 2, where the defaults system allows you to simply replace the first histogram.

```
ANAPAW> hst1
 HST1> (=0) 105
 HST1> (=105) 0

 1D-Histograms:

   ( HID :  105 ) ( TITLE : Coin 2 Gated        ) ( GATE :     2 )
   ( bin :    2 ) ( xmin :    -0.50 ) ( xmax :     1.50 )
   ( VALUE :   1   1   1  5 )

   ( HID :  106 ) ( TITLE : Coin 2 Gated inv    ) ( GATE :    -2 )
   ( bin :    2 ) ( xmin :    -0.50 ) ( xmax :     1.50 )
   ( VALUE :   1   1   1  5 )

   ( HID :  107 ) ( TITLE : M_dE                ) ( GATE :     0 )
   ( bin :   11 ) ( xmin :    -0.50 ) ( xmax :    10.50 )
   ( VALUE :   3 100 100  2 )

   ( HID :  108 ) ( TITLE : M_dE Gated          ) ( GATE :     3 )
   ( bin :   11 ) ( xmin :    -0.50 ) ( xmax :    10.50 )
   ( VALUE :   3 100 100  2 )
```

/ANALYS/DEFINITION/HST2

Define ZONE ,IDs of *2D*-histograms to see. By defaults (⟵ press or value 0 given) the output is created with *single* histogram.See Section 2.3.2 for examples.

/ANALYS/DEFINITION/LIST1D

List *1D*-histograms.

```
ANAPAW> list1d [ HID ]

        HID  :  0, All lists are displayed.
               -1, Definitions of histograms defined
                   by HST1 are displayed.
```

/ANALYS/DEFINITION/LIST2D

List *2D*-histograms.

```
ANAPAW> list2d [ HID ]

        HID  :  0, All lists are displayed.
               -1, Definitions of histograms defined
                   by HST2 are displayed.
```

/ANALYS/DEFINITION/LISTPROF

Print definition of *profile* histograms.

```
ANAPAW> listprof [ HID ]

        HID  :  0, All lists are displayed.
```

/ANALYS/DEFINITION/LISTGATE

Print definitions of the gates defined by ANA-CODE.

```
ANAPAW> listgate [ GKIND ]

        GKIND  :  ' '          , All Gates are displayed
                  G  [GATE]    , Gates defined by [GATE] are displayed
                  A  [AND]     , Gates defined by [AND] are displayed
                  O  [OR]      , Gates defined by [OR] are displayed
                  X  [XYGATE], Gates defined by [XYGATE] are displayed
```

/ANALYS/DEFINITION/AVIEW

Maybe the easiest way to view ANA-CODE definitions: gates ([GATE],[AND], [OR],[XYGATE], and histograms (1D and 2D) see Section 3.4), is by opening a special X-Windows !

```
ANAPAW> aview [ OC ]

        OC  :  O  Open window (default)
               C  Close window
```

/ANALYS/DEFINITION/HCUT

Create 2D-gate data file. One needs to input gate ID before making gates. If HID = 0, current histogram is displayed. If filename=‘ ’ the name **Anapaw.cut** is given. If you uses special options for the current histogram, see Section 2.3.3, you must add them here with histogram ID. Cut is made directly on window, using mouse ; left-click to add a new point and right-click to end the cut. Note that the last and the first point are automaticly connected. Gates are mainly described in ANA-CODE, see Section 3.4.

```
ANAPAW> hcut [ HID filename COPT ]

        HID      : Histogram ID
        filename : File to save 2D-Gate Data
COPT     : M Main window
                  S Scatter window
```

◇◇Example:

```
ANAPAW> hcut 117 bokuno.cut

  ANAPAW-M : FileName = bokuno.cut

 Input ID of 2D-Gate? (=1001)
 Any Comment? > Wakatta !
```

Create a cut in the histogram number 117 and export it with the gate number 1001 in the **bunkuno.cut** file.

2.3.3 /ANALYS/HISTOGRAM/

/ANALYS/HISTO/HSTORE

Store the Histograms in //PAWC to HBOOK type file. Default name file is **Anapaw.hbk** .

```
ANAPAW> hstore [ HBKFILENAME ]

        HBKFILENAME : Filename
```

◇◇Example:

```
ANAPAW> hstore text.hbk
```

/ANALYS/HISTO/FETCH

Fetch histogram ID from HBOOK file to //PAWC.

```
ANAPAW> fetch [ HBKFILENAME OFFSET ]

        HBKFILENAMEC : Filename to load the Histograms
        OFFSET       : Offset for Histogram ID
```

ANALYS/HISTOGRAM/HLIST or I

Print list of all histograms.

```
ANAPAW> HLIST [ CHOPT ]

        CHOPT : P Pager more

ANAPAW> I [ CHOPT ]

        CHOPT : P Pager more
```

/ANALYS/HISTO/HT

Plot a histogram. This command saves current histogram ID. By default plot the current histogram. Options definition needs to put histogram ID.

```
ANAPAW> ht [ HID CHOPT ]

HID   : Histogram ID
        CHOPT : Plot options, same as normal PAW
```

/ANALYS/HISTO/HTP

Plot the *current* histogram. Allow options definition without putting the histogram ID.

```
ANAPAW> htp [ CHOPT ]

        CHOPT : Plot options, same as normal PAW
```

♢♢To plot the current histogram with colorscale enter:

```
ANAPAW> htp colz
```

/ANALYS/HISTO/HNT or /ANALYS/HISTO/HN

Plot the next histogram. This command saves current histogram ID.

```
ANAPAW> hn [ CHOPT ]

        CHOPT : Plot options, same as normal PAW
```

/ANALYS/HISTO/HBT or /ANALYS/HISTO/HB

Plot the previous histogram. This command saves current histogram ID.

```
ANAPAW> hb [ CHOPT ]

        CHOPT : Plot options, same as normal PAW
```

/ANALYS/HISTO/BLOW

Create a histogram between BLXMIN and BLXMAX.

```
ANAPAW> blow BLXMIN BLXMAX

        BLXMIN : Minimun
BLXMAX : Maximun
```

/ANALYS/HISTO/XBLOW

Expand a histogram using mouse in x-axis.

```
ANAPAW> xblow
```

/ANALYS/HISTO/XYBLOW

Expand a histogram using mouse in x-axis and y-axis.

```
ANAPAW> xyblow
```

/ANALYS/HISTO/PRX

Create a histogram for projection on x-axis. Note that this command does not only create a projection but does create the corresponding histogram.

```
ANAPAW> prx
```

/ANALYS/HISTO/PRY

Same as prx but for y-axis.

```
ANAPAW> pry
```

/ANALYS/HISTO/BNX

Create a histogram for projection on x-axis restricted to the y interval (YMIN,YMAX). Note that this command does not only create a projection but does create the corresponding histogram.

```
ANAPAW> bnx YMIN YMAX

        YMIN : Minimun
        YMAX : Maximun
```

/ANALYS/HISTO/BNY

Same as BNX but for the y-axis using restriction on x-axis.

```
ANAPAW> bnx XMIN XMAX

        XMIN : Minimun
        XMAX : Maximun
```

/ANALYS/HISTO/SLX

Create projections onto the y axis, in x-slices. Note that this command does create the SNY corresponding histograms.

```
ANAPAW> slx SNY

        SNY : Number of slices
```

/ANALYS/HISTO/SLY

Same a SLX but for the y-axis.

```
ANAPAW> sly SNX

        SNX : Number of slices
```

/ANALYS/HISTO/MAMI

Change the y-axis limit of current histogram

```
ANAPAW> mami MRYMIN MRYMAX

        MIN : Mininum
MAX : Maximun
```

/ANALYS/HISTO/XSTATUS

Print status of a given histogram

```
ANAPAW> xstatus
```

/ANALYS/HISTO/HSTATUS

Print status of the current histogram

```
ANAPAW> hstatus
```

/ANALYS/HISTO/ERASE

Erase histogram contents and **ev**ents **s**ummary (EVS).

/ANALYS/HISTO/HDELETE

Delete the current (if HID1 = HID2 = 0) or given histograms.

```
ANAPAW> hdelete HID1 HID2

        HID1 : First histogram
MID2 : Second histogram
```

/ANALYS/HISTO/XVAL; /ANALYS/HISTO/XYVAL

Print channel contents of a given bin in x-axis (XVAL) or in x and y axis (XYVAL).

```
ANAPAW> xyval [ CHOPT ]

        CHOPT : ' ' Histogram mode
                S  Scatter mode
```

/ANALYS/HISTO/HSTAT

Print statistics on *current* histogram.

```
ANAPAW> HSTAT
```

/ANALYS/HISTOGRAM/XSTAT

Print statistics on *selected* area for the *current* histogram.

```
ANAPAW> XSTAT
```

/ANALYS/HISTOGRAM/FIGA

Fit the *current* histogram using a Gaussian function. It then creates a new histogram.

```
ANAPAW> FIGA
```

/ANALYS/HISTOGRAM/CPRO

Created a new histogram using (contour) cut in another histogram, see Section 2.3.3.

```
ANAPAW> CPRO [ HID COPT ]

        HID   : Histogram ID
        CHOPT : M Main Window
                S Scatter Window
                OPT HPLOT Option
```

/ANALYS/HISTOGRAM/AVY

Project contents to a profile histogram.

```
ANAPAW> AVY [ HID COPT ]

        HID   : Histogram ID
        CHOPT : ' '    Default
                S      Spread Option
                I      Error on Mean
```

/ANALYS/HISTOGRAM/XFITG

Fit the current histogram using a Gaussian function, but here you can select the abscissa area. It then creates a new histogram.

```
ANAPAW> XFITG
```

2.3.4 /GRAPHICS/

/GRAPHICS/PSON

Open a Postscript file for output.

```
ANAPAW> PSON
```

/GRAPHICS/PSOFF

Open the current output on Postscript file.

```
ANAPAW> PSOFF
```

◇◇Example with one histogram, sent to printer at the end.

```
ANAPAW> pson
File name (<CR>=)
                  -> `` sugoi.ps ''
ANAPAW> ht 101
ANAPAW> psoff
ANAPAW> sh chsize sugoi.ps | lpr
```

Chapter 3

Analys Code

ANA-CODE aims to define gates and histograms for the carried experiment analysis. Everything is written in a file load in ANAPAW by the command `book`, see Section 2.3.2.

3.1 Hierarchy

Before starting with the code itself, one should know how the observables are organized in ANALYS. The first level is the analyzer type, which roughly correspond to the detector. Then the analyzer in divide in analyzer ID, like a detector can be divided in small parts. Finally these smaller parts are also divided in different words, which correspond to different observable (*raw or calculated*) like amplitude, timing, raw or calibrated data...

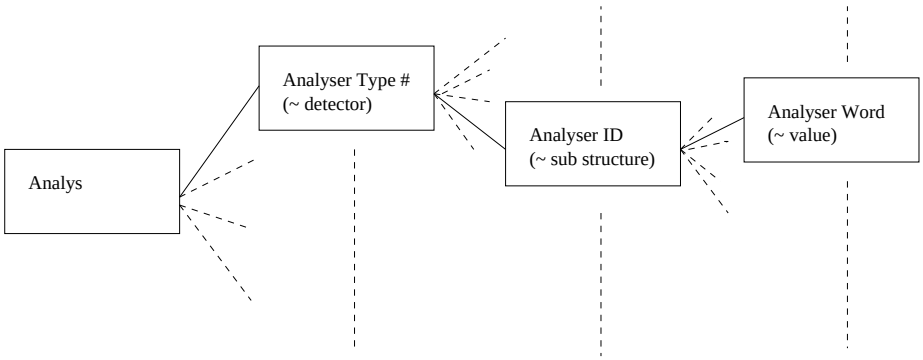


Figure 3.1: Analyzer's hierarchy in Analys

3.2 Comments

As Fortran file, the analysis description can be commented. The command looks like Fortran ones, that is to say using '`c`'.

3.3 Analys

Analys File must start by the declaration of the analyzer used.

```
ANALYS
ID of the first analyzer
ID of the 2nd, etc
```

◇◇Example

```
ANALYS
2
73.4 Gate(s) 3.4.1 1D Histograms
```

3.4 Gates

3.4.1 GATE

For simple data gate, based on 1D-histogram:

```
GATE
gate ID, analyser type,
         ID of first analyzer into analyzer type
         ID of last analyzer into analyzer type
         word defining the analysis
         min X,max X
```

◇◇Example:

```
GATE
c TOF(RF2-F3) 54Ni
1, 2, 5, 5, 6, 263., 268.
```

3.4.2 Logical OR, AND

For logical gate OR (or AND):OR (or AND) gate ID, list of gates linked by OR (or AND), ...

◇◇Example:

```
AND
c 52Fe 850keV
200, 50, 903.5.
```

3.4.3 XYGATE

Create a gate using a cut file (ANAPAW cut file), see Section 2.3.2

```
XYGATE
@,name of cut file
```

◇◇Example using a file:

```
XYGATE
c 1001
@,cut/54Fe.cut
```

3.4.4 STOP

Stop filling histogram placed after this command, for the current event.

```
STOP
[-]gate number
```

♢♢Example with an already defined gate 3. In this case, all the commands placed after stop *won't* be read, if the current event does *not* satisfied (a minus is placed before) gate 3

```
GATE
3,1,4,4,5,25.,44.
STOP
-3
HST1
0,1,5,5,5,100,0,100.,'Beautiful 1D Histo gated'
```

3.5 Histograms

3.5.1 HST1: 1D Histograms

Histograms definition are equivalent wth the HBOOK case. If you are not familiar with this CERN library please have a look at its manual:
http://wwwasdoc.web.cern.ch/wwwasdoc/hbook_html3/hboomain.html
The command to be put a the beginning of the 1D histogram list is `HST1`. Then each histogram is defined by:

```
HST1
gate ID, analyzer type,
         ID of first analyzer into analyzer in X
         ID of last analyzer into analyzer type
         word defining the channel read in X
         num of bin,min,max,'title'
```

3.5.2 PROFILE

This type of 1D histogram can be created using a 2D vector. The Y-coordinate is the *mean* of the vector's 2nd coordinate, with a error. The way to calculate this last depends of the last option, which can be ' ', 'S' or 'I'. Please see CERN HBOOK manual for further precisions.

Warning: the mean and the more probable 1st coordinate are most of time different. Please beware of your the 2nd coordinate distribution !

```
PROF
gate ID, analyzer type,
         ID of first analyzer into analyzer type in X
         ID of last analyzer into analyzer type in X
         word defining the channel read in X
         ID of first analyzer into analyzer type in Y
         ID of last analyzer into analyzer type in Y
```

```
         word defining the channel read in Y
         num of X bin,min X,max X,
         min Y,max Y,
         'title','error'
```

3.5.3 HST2: 2D Histograms

The command to be put a the beginning of the 2D histogram list is `HST2`. Then each histogram is defined by:

```
HST2
gate ID, analyzer type,
         ID of first analyzer into analyzer type in X
         ID of last analyzer into analyzer type in X
         word defining the channel read in X
         ID of first analyzer into analyzer type in Y
         ID of last analyzer into analyzer type in Y
         word defining the channel read in Y
         num of X bin,min X,max X,
         num of Y bin,min Y,max Y,
         'title'
```

3.6 ANA-CODE file example

```
c === ANA-FILE ===
ANALYS
2
5
GATE
c TOF(RF2-F3) 54Ni oya
1, 2, 5, 5, 6, 263., 268.
c TOF(RF2-F3) 54Ni ko
2, 2, 5, 5, 6, 298.5, 303.5
c TOF(RF1-F3) 54Ni oya
3, 2, 5, 5, 4, 191.5, 196.5
c TOF(RF1-F3) 54Ni ko
4, 2, 5, 5, 4, 228., 232.
c TOF(RF2-F3) 52Fe oya
5, 2, 5, 5, 6, 270., 275.
AND
c 52Fe
11, 2, 4
OR
c SSD gate
21, 3, 4
c TOF
HST1
0, 2, 5, 5, 6, 300, 100., 400.,'TOF(RF2-F3)'
0, 2, 5, 5, 4, 300, 100., 400.,'TOF(RF1-F3)'3.8. ANA-CODE FILE EXAMPLE29c SSD
```

```
HST2
c rot dE vs rotE
90,5,301,301,2,2,301,301,3,200, 1000.,4000.,200,-1000.,3000.,'dE vs E 2'
90,5,301,301,3,3,301,301,3,200, 1000.,4000.,200,-1000.,3000.,'dE vs E 3'
90,5,301,301,4,4,301,301,3,200, 1000.,4000.,200,-1000.,3000.,'dE vs E 4'
c
EXIT
```

Chapter 4

Anapaw Source Code

You can directly modify ANAPAW/Analys files, by modifying appropriate encoding files (`enc_*.f`). Each one describes the analyzer used (see Section 2.3.3) and can be modified, for example to calibrate the signal. The main used language is Fortran 77.

4.1 Analyzer Definition

To create a new analyzer, you should modify the source file named `usersrc.f` you can find in `usersrc-*.tar.gz`, see Section 2.1.2.

The minimum required is :

```
If(AnalyzerFlag(#))Then
   Call Enc_ANA( val(1,1,#),nx,ny,naok(#) )
EndIf
```

where `#` stands for the analyzer's number and `Enc_ANA` is a the encoding subroutine defined generally in a `enc_*.f` file.

`val(1,1,#)` represents the output array of observables values defined in the encoding subroutine, `nx` and `ny` give the two first dimension of `val(1,1,#)`.

Finally `naok` in an internal index that organize the observable in its array. Note that data are compressed: you do not need 100 lines in you data matrix `val` to access the word 100! The first column of you matrix contains this word ID....

An analyzer can be related to Raw Data. In this case, one should put in argument the corresponding array.

♢♢Example, where `EvtData(1)` is the raw data array, retrieved by the subroutine `FND_EID` and convert into observables in `Enc_SSD`:

```
c
c Analyzer 4: SSD
c
      If(AnalyzerFlag(4))Then
         call FND_EID(EvtData(1),5,iadr_sub_T,nw_sub_T) ! Timing
         call FND_EID(EvtData(1),6,iadr_sub_A,nw_sub_A) ! Analog
         If(nw_sub_T.le.0 .or. nw_sub_A.le.0)Then
```

```
          EVTERR = .TRUE.
          Return
       EndIf
       Call Enc_SSD( EvtData(iadr_sub_T),nw_sub_T,
&         EvtData(iadr_sub_A),nw_sub_A,
&         val(1,1,4),nx,ny,naok(4) )
 EndIf
```

The purpose here is *not* to describe all the possible subroutines, but to give the main rules to modify them. Source files and their subroutines strongly depend on the experiment, the analysis, the author... Normally you will not have to hack this file, and if so, this manual may not enough to help you.

4.2 Encoding File

4.2.1 From raw data to observable

Here you can find the physics! Because you will certainly want to understand quickly how to deal with the code, we will here explain everything by examples.

Let's take *2* raw values given respectively by the index 7 & 11 of the array `raw_data`, i.e `raw_data(7)` & `raw_data(11)`. In 99% of the cases raw values are asked to be observable.

You will certainly see this type of code in the already written encoding files:

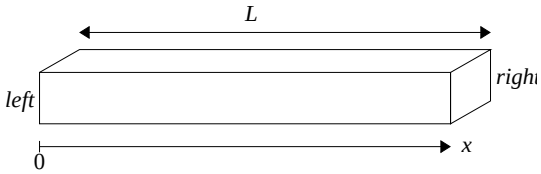
```
ID=2
naok=naok+1
val(1,naok) = ID
val(2,naok) = raw_data(7)
val(3,naok) = raw_data(11)
```

In this case `ID` is the IDenfication for the sub-analyzer, as it is explained in Section 3.

Warning: You must write `val(1,naok) = ID`, because ANAPAW uses this first valu as identifier !

4.2.2 A first example

Let's take the case of long scintillator of length L with one photo tube at each side (let's say $left$ and $right$), see Fig. 4.2.2.



Because the amplitude A received by the light is mainly exponentially decreasing with the distance x where a particle has hit the detector, the position

of the hit can be guess by taking the logarithm of the ratio of the amplitude at each side. In equations:

$$A_{right} \propto e^{-\frac{L-x}{X_0}} \quad \& \quad A_{left} \propto e^{-\frac{x}{X_0}} \quad \Rightarrow \quad x \propto Ln\frac{A_{right}}{A_{left}}$$

In order to use the previous example, we can stand that the raw data for right and left amplitude are given by `raw_data(7)` & `raw_data(11)` respectively. We want this observable to be sort with the same sub analyser, in a encoding file named for example `enc_scinti`. Then the previous code becomes for example:

```
ID=2
naok=naok+1
val(1,naok) = ID
val(2,naok) = raw_data(7)
val(3,naok) = raw_data(11)
if (val(3,naok).ne.0) then
pos_A = ln(val(2,naok)/val(3,naok))
val(3,naok) = pos_A
endif
```

4.2.3 Calibration

Position of hits can also be known by time difference: $x \propto t_{left} - t_{right}$. Imagining that `raw_data(12)` & `raw_data(16)` are timing for right & left PMT, the previous code can be extended as:

```
      ID=2
      naok=naok+1
      val(1,naok) = ID
      val(2,naok) = raw_data(7)
      val(3,naok) = raw_data(11)

c position using amplitude
      if (val(3,naok).ne.0) then
           pos_A = ln(val(2,naok)/val(3,naok))
      val(3,naok) = pos_A
      endif

      val(4,naok) = raw_data(12)
      val(5,naok) = raw_data(16)

c position using time
      pos_t = val(5,naok)-val(4,naok)
      val(6,naok) = pos_t
```

If we want to calibrate this values to get the position in mm, we need to hack 2 other files: one where the declaration of variables are (`*.inc`, here for example `scinti.inc`) and the source files for initializing the coeficients for calibration (`*_prm.f` here `scinti_prm.f`). Finally the coeficients themselves will be put in a `*.prm` file. The name of this file can be changed in the `$USER_SOURCE/setana` file, see Section 2.1.2.

Starting from the end, we want to create a new sub analyzer in order to distinguish between raw and calibrated data.

```
ID=3
naok=naok+1
val(1,naok) = ID
val(2,naok) = to_x(1) + pos_A * to_x(2)
val(3,naok) = to_x(3) + pos_t * to_x(4)
```

where `to_x` is the array for calibration. We have to declare it in `scinti.inc` by adding a line to the common variables.

```
Real to_x
Common/scintillator/
     .
     .
     .
& to_x(4)
```

In order to read the parameter file, for example, `scinti.prm`, you should first now under which number this last is open in `scinti_prm.f`. As an example let's take `80`. The procedure to fill a array of real reading a file is:

```
READ_FLT(Unit_number,
         Array_name,
         Size,
         Error_code)
```

For an array of integer, the syntax is the same, but the function is :

```
READ_INT(Unit_number,
         Array_name,
         Size,
         Error_code)
```

The error code is different from 0 in case of problems [1].
In this example, the code added is:

```
Call read_flt(80,to_x,4,ier)
```

Warning: The parameter are read continously in the `*.prm`, if you exchange 2 `read_flt` procedures without exchanging the parameters in the read file, then you will have no error message (in Fortran !) and the result will be... how shall I put it... stlightly weird.

As for the `scinti.prm` file:

```
c Calibration for scintillator
10, 0, .43, -7.4.3.
```

Of course a better way to hack this is to create two arrays: one for time, and one for analog. You can also create a condition for turning off the calibration. The final code will be :

◇◇`scinti.inc`

[1] You can then add a condition to write a message, stop reading...

```
       Integer iflag_AtoX_scinti,iflag_dttoX_scinti
       Real A_to_x, dt_to_x
       Common/scintillator/
             .
             .
             .
      & iflag_AtoX_scinti, A_to_x(2)
      & iflag_dttoX_scinti, A_to_x(2)
```

◇◇scinti_prm.f

```
c Calibration for scintillator
       Call read_flt(80,A_to_x,4,ier)
       Call read_flt(80,dt_to_x,4,ier)
```

◇◇scinti.prm

```
c Calibration for scintillator : Analog
c iflag_AtoX_scinti
1
c values
10, 0
c Calibration for scintillator : Time
c iflag_dttoX_scinti
0
c values
.43, -7.
```

◇◇enc_scinti.f

```
       ID=3
       naok=naok+1
       if (iflag_AtoX_scinti.ne.0) then
       val(2,naok) = A_to_x(1) + pos_A * A_to_x(2)
       endif
       if (iflag_dttoX_scinti.ne.0) then
       val(3,naok) = dt_to_x(1) + pos_t * dt_to_x(2)
       endif
```

4.2.4 Multi array detector

Finally, in some cases, this plastic can belong to a bigger detector array. Imagining that our array is composed of 10 plastics, identicals, for which we want the same observable, then a quite look at how everything is organised in ANALYS tells us that we don't have to repeat each observable declaration, but loop on ID. and detector IDs will be taken from 1 to 10. Then the code becomes :

◇◇scinti.inc

```
       Integer number_det
       Parameter (number_det=10)
       Integer iflag_AtoX_scinti,iflag_dttoX_scinti
       Real A_to_x(2,number_det), dt_to_x(2,number_det)
```

```
      Common/scintillator/
            .
            .
            .
     & iflag_AtoX_scinti, A_to_x
     & iflag_dttoX_scinti, A_to_x
```

◇◇scinti_prm.f

```
c Calibration for scintillator
      Call read_flt(80,A_to_x,2*number_det,ier)
      Call read_flt(80,dt_to_x,2*number_det,ier)
```

◇◇scinti.prm

```
c Calibration for scintillator : Analog
c iflag_AtoX_scinti
1
c values
10.,  0., 26., 38., 18., 39., 38., 76., 43., 12.,
23., 62., 59., 53., 20., 23., 78., 23., 45., 89.
c Calibration for scintillator : Time
c iflag_dttoX_scinti
0
c values
.43 -7. .2 4.8 -45.6 -6.9 5.1 47.  90.  5.7
5.0 1.6 1.0 2.5 -2.5 -2.0 1.5 -2.0  3.4 2.8
c
```

◇◇enc_scinti.f

```
      Do ID = 1,10
        naok=naok+1
        if (iflag_AtoX_scinti.ne.0) then
           val(2,naok) = A_to_x(1,ID) + pos_A(ID) * A_to_x(2,ID)
        endif
        if (iflag_dttoX_scinti.ne.0) then
            val(3,naok) = dt_to_x(1,ID) + pos_t(ID) * dt_to_x(2,ID)
        endif
      Enddo
```

4.3 Ntuples

Anapaw allows the storage of ntuples (Column-Wise types), using the command ANTUPLE, see Section 2.3.1. But before using it, you should defined ntuples in two subroutines in `add_ntuple.f` file.

In subroutine NTBOOK you call the HBOOK routine HBNAME to describe the variables that are to be stored in the Ntuple. Please refer to HBOOK manual for more details, here under is the (simplified) synopsis:

```
      CALL HBNAME (ID, CHBLOK, VARIABLE, CHFORM)
```

```
      ID
          Identifier of the Ntuple as in the call to HBNT.
      CHBLOK
          Character variable of maximum length 8 characters
          specifying the name by which the block of variables
          described by CHFORM is identified.
      VARIABLE
          The first variable that is described in CHFORM.
      CHFORM
          Character string describing the variables to be
          stored in block CHBLOK
```

Warning: For now the ntuple number is automatically 10 and named EXP.

In the second subroutine, the different variables will be filled, every other events.

♢♢Here under a complete example:

```
c ======================================================================
      Subroutine NTBOOK

c -- from ANALYS Source !! Do Not Edit !! ------------------------------ c
      Logical  AnalyzerFlag(50),INITENCFLAG(50)
      Logical  USERFLAG(10),EVTERR
      Common/ANALYSLOGIC/  AnalyzerFlag,INITENCFLAG,USERFLAG,EVTERR
c ---------------------------------------------------------------------- c

c -- for ntuple defined by user ---------------------------------------- c

c -- Analyzer 1
      Integer   RunNum
      Integer   Coin
      Common/CoinReg/ RunNum,Coin

c == Booking Part ==

      If(AnalyzerFlag(1)) Then
         Call HBNAME(10, 'CoinReg', RunNum,'RunNum:I,Coin:I')
      EndIf

      Return
      End


c ======================================================================
      Subroutine Add_Ntuple(IOFLAG)

c -- for Local --------------------------------------------------------- c
      Logical  IOFLAG   ! ( True : Next File )
```

```
c -- from ANALYS Source !! Do Not Edit !! ------------------------------- c
      Real     val(500,500,50)
      Integer  RunNumber
      Logical  AnalyzerFlag(50),INITENCFLAG(50)
      Logical  USERFLAG(10),EVTERR
      Common/ANALYSVALUE/  val
      Common/AnalysNtuple/ NTUPLE_FIRST,RunNumber
      Common/ANALYSLOGIC/  AnalyzerFlag,INITENCFLAG,USERFLAG,EVTERR
c ---------------------------------------------------------------------- c

c -- for ntuple defined by user ------------------------------------------ c

c -- Analyzer 1
      Integer   RunNum
      Integer   Coin
      Common/CoinReg/ RunNum,Coin

c ---------------------------------------------------------------------- c
      If(IOFLAG)Then                  !! Do Not Edit !!
         Call Ntuple_io(0)            !! Do Not Edit !!
         Call Ntuple_io(1)            !! Do Not Edit !!
      EndIf                           !! Do Not Edit !!

c ---------------------------------------------------------------------- c
c               Definition of val(Word,ID,Analyzer)                      c
c ---------------------------------------------------------------------- c

c -- Analyzer 1
      If(AnalyzerFlag(1)) Then
         Coin = val(3,1,1)
      EndIf

      Return
      End
```

Chapter 5

Miscellaneous

5.1 Phone numbers

Room	Extension
Rips	4186
Control room	4126 or 4127
E6	4176
J5	4188

5.2 Data copy

5.2.1 Tape duplication

In order to create a backup, from one tape (in `/dev/nst0`) to another (in `/dev/nst1`)

```
mtcopy /dev/nst0 /dev/nst1
```

5.2.2 From tape to hard drive

In order to save data from tape (in `/dev/nst0`) on hard disk.

```
mtcopy /dev/nst0 /home/...
```

5.2.3 From hard drive to hard drive

Transferring data from the current hardware to, for example, `rips00` :

```
scp run[number].rdf rips00:/[user]/[exp]
```

Reminder : most of time the current hard drive has a very limited size, so delete the data from it after transfer.

5.3 Data structure

```
Block Header
0001 0000 0000 0000 Header Block
0000 0000 0000 0000 Data Block

#### Header Block  16KBytes ####
0001 0000 0000 0000 0000 0000 0000 0000 --> Constant
0000 0000 ....

Word
0      : 0001 - Flag of Header
1-9    : 0000
10-13  : Run Number (ASCII   8Char.) "RUN-1066"
14     : Space * 2  (ASCII   2Char.) "  "
15-23  : Start Time (ASCII  18Char.) "START => 12:58:56 "
24-32  : Stop Time  (ASCII  18Char.) " STOP => 13:53:36 "
33     : Space * 2  (ASCII   2Char.) "  "
34-42  : Print Time (ASCII  18Char.) "Print -> 13:57:07 "
43-47  : Print Date (ASCII  10Char.) " 22-SEP-98"
48-49  : 0000
50-89  : Header     (ASCII 80char.)
90-129 : Space      (ASCII 80Char.)
130-   : 0000

---- Example ----
\0  \0  \0  \0   R   U   N   -   1   0   6   6           S   T
 A   R   T       =   >       1   2   :   5   8   :   5   6
     S   T   0   P       =   >       1   3   :   5   3   :   3
 6               P   r   i   n   t       =   >       1   3   :
 5   7   :   0   7           2   2   -   S   E   P   -   9   8
\0  \0  \0  \0   R   u   n   1   0   6   6       s   a   m   e
     a   s       b   e   f   o   r   e
-----------------

#### Ender Block 16KBytes ####
ffff 0000 0000 0000 0000 0000 0000 0000 -- > Constant
0000 0000 ....

Word
0      : ffff - Flag of Ender
1-9    : 0000
10-13  : Run Number (ASCII 10Char.) "RUN-1066"
14     : Space * 2  (ASCII  2Char.) "  "
15-23  : Start Time (ASCII 18Char.) "START => 12:58:56 "
24-32  : Stop Time  (ASCII 18Char.) " STOP => 13:53:36 "
33     : Space * 2  (ASCII  2Char.) "  "
34-42  : Print Time (ASCII 18Char.) "Print -> 13:57:07 "
43-47  : Print Date (ASCII 10Char.) " 22-SEP-98"
48-49  : 0000
```

```
50-89  : Header     (ASCII 80char.)
90-129 : Ender      (ASCII 80Char.)
130-   : 0000

#### Event Block 16KBytes ####
0000 0000 0000 0000 .... .... .... ....

Word
0-3    : 0000 - Flag of Data
4      : Event size (Words - Include this)
5      : FID = 1 -> Constant
6      : Event ID
7      : Segment size (Words - Include this)
8      : Segment ID
9-     : Data

$ Event size $
Ex.) 0x8046 --> 1000|0000|0100|0110
                ^    ^^^^^^^^^^^^^
          Always '1'   Event size

$ Fixed data (Non FERA) $
0      : Segment size (Words - Include this)
1      : Segment ID
3-     : Data

$ Variable data (FERA) $
0      : Segment size (Words - Include this)
1      : Segment ID
3      : FERA Data size (Words - Except this)
4      : FERA Header
Ex.) 0x800e --> 1000|1000|0000|1110
                ^
          Always '1'

 16           11
 --------------------------------------
| 1|WC=0to15  |       |                |
 --------------------------------------
```

5.4 Scaler data

Scaler data is written by *babarlDAQ* at the end of each block (16kB). The order is given by order of calling functions

```
read_scaler()
```

in *scaler.c* of the DAQ code. Each scaler value occupies 2 words (4 bytes) in block. For simplicity let's assume the following exmple: we have 12 scaler values to read in the DAQ code. Then the word order in block is the following:

```
Word#:   Data:
8168     Scaler1
8170     Scaler2
8172     Scaler3
....     ......
....     .......
8188     Scaler11
8190     Scaler12
```

There is 8192 words in a block so the word number of the first scaler value is always $= 8192 - N * 2$. Where N is number of scaler values defined in the DAQ code.

Notes


10番