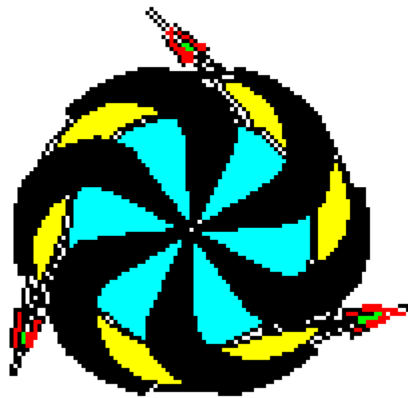
TWIST

MOFIA 2.2 MANUAL



May 6, 2002

Maher Quraan

I. INTRODUCTION 3

II. SOURCE CODE ORGANIZATION 5

III. CVS 6

IV. INSTALLING AND COMPILING MOFIA 8

IV.1 COMPILING MOFIA 8
IV.2 COMPILING A DEBUG VERSION 9
IV.3 MODULE DEPENDENCIES 9

V. RUNNING MOFIA 10

V.1 GENERAL COMMANDS 10
V.2 FLAGS 11
V.3 NAMELISTS 12
V.4 FUNCTIONS 13
V.5 ENVIRONMENT VARIABLES 14

VI. CALIBRATION FILE MANAGER (CFM) 15

VII. INITIALIZATION BRANCH 18

VII.1 GEOMETRY 18
VII.2 HISTOGRAMMING 23

VIII. ANALYSIS BRANCH 27

VIII.1 TDC UNPACKING 28
VIII.2 FILTERING 32
VIII.3 CROSS TALK 32
VIII.4 CALIBRATIONS 33
VIII.4.1 Efficiency 33
VIII.4.2 Time Zero 34
VIII.4.3 Alignments 34
VIII.4.3.1 Translational Alignments 35
VIII.4.3.2 Rotational Alignments 35
VIII.4.4 Resolution 35
VIII.5 PATTERN RECOGNITION 37
VIII.6 TRACKING 37
VIII.6.1 χ^2 Fit 37
VIII.6.2 Kalman Filter 37

X. APPENDECIES 38

X.1 NAMELIST VARIABLES 38
X.2 FAILURE CODES 45
X.2.1 Event Filtering Failure Codes 45
X.2.2 Pattern recognition Failure Codes 46
X.2.3 $\chi 2$ Fit Failure codes 47
X.2.4 Kalman Filtering Failure Codes 47

X.3 DATA STRUCTURES 48
X.3.1 Geometry Structures *48*
X.3.2 TDC Structures *50*
X.3.3 Calibrations Structures *52*
X.3.4 Windowing Structures *53*
X.3.5 Clustering Structures *54*
X.3.6 First Guess Structures *55*
X.3.7 χ^2 *Helix Fit Structures* *56*
X.3.8 Kalman Filter Structures *57*
X.3.9 MC Banks Structures *58*
X.3.9 MC Banks Structures *59*
X.4 FLOWCHARTS 60
X.4.1 Initialization Branch *60*
X.4.2 Analysis Branch *61*

I. Introduction

The framework of MOFIA has been adapted from KOFIA, the software package developed for BNL experiment E787. The first version, MOFIA 1.0 contains the code used to analyze the test data acquired with 5 x-planes and 2 y-planes in the test run of August 1997. MOFIA 1.0 contains a full software analysis package of this test data, including: calibrations, efficiency and tracking. The second version, MOFIA 1.5, includes a “skeleton” that would allow the user to access the Monte Carlo data for purposes of developing the remaining software packages for TWIST. Both MOFIA 1.0 and MOFIA 1.5 are written in FORTRAN 77. Since then the collaboration have decided to adopt a more modern language, FORTRAN 90, due to many attractive features in this language. This lead to the development of MOFIA 2.0. Since FORTRAN 90 compilers are able to compile FORTRAN 77 code, we decided to keep as much as we can of the MOFIA main framework code (the code adapted from E787) in FORTRAN 77. This was mainly done for two reasons: first, the code adapted from E787 is well written and tested so that changing it will unnecessarily consume too much manpower; and second, compatibility with E787 is desired since it will allow us to benefit from E787 modifications to the code. For the same reasons, we decided not to modify any of the other packages that we adapted, namely: CFM, YBOS, GPLOT, BRU and CERN libraries. On the ALPHA, these packages are compiled using the f77 compiler. On LINUX, however, it was necessary to recompile them with f90, due to incompatibilities between our compiler of choice for LINUX f90, ABSOFT, and the g77 compiler. Only minor changes, however, were made to these packages.

All the TWIST code written for MOFIA 1.5 was completely rewritten and reorganized in MOFIA 2.0 in order to utilize the nice features of FORTRAN 90, including the data structures. MOFIA 2.0 also includes the geometry structures and the TDC unpacking for the proportional chambers. In addition, MOFIA 2.0 is capable of analyzing the test data of August 1999, for the UV prototype chamber pair. This last feature allowed us to test MOFIA 2.0 with real data and compare directly the results from MOFIA 2.0 and MOFIA 1.5. While the collaboration has initially decided to support four computer platforms, only the LINUX and ALPHA platforms have been used since the release of MOFIA 2.0. While the TWIST cluster will consist entirely of LINUX boxes, the continuing support of the ALPHA platform is considered beneficial since it might reveal code (or even compiler) bugs that may otherwise go undetected by the LINUX ABSOFT compiler. MOFIA 2.0 has been tested on ALPHA and LINUX, and the analysis results from both Monte Carlo and real data were identical.

Since then another version, MOFIA 2.1, has been released but the MOFIA 2.0 manual has not been updated to reflect the modifications in that version. This manual will be entirely focused on the current version, MOFIA 2.2.

In the discussion below, some knowledge of FORTRAN 90 is assumed. We found that the book “Upgrading to FORTRAN 90” by Cooper Redwine provides a good introduction to

FORTRAN 90, at least for those familiar with FORTRAN 77. Specific documents for the ALPHA and the ABSOFT compilers and debuggers are also useful references. For ABSOFT, this documentation may be found on any machine where ABSOFT is installed in the directory ***/usr/absoft/doc.*** In particular, the file ***FxUserGuide.pdf*** provides information on the ABSOFT ***Fx*** multi-language debugger, and the file ***F90_Reference.pdf*** provides information on the FORTRAN 90 compiler. Information on our debugger of choice for the ALPHA, ***ladebug,*** is on the web at

http://www.compaq.com/products/software/ladebug

and information on DEC FORTRAN 90 is at

http://www.triumf.ca/internal/internal-links/df90/dfau.htm

II. Source Code Organization

The source code for MOFIA 2.2 is in the directory ***~username/mofia/2.2/source***. Several subdirectories reside in it: ***main***, ***mainf90***, ***dummy***, ***include***, ***modules***, ***photo***, and ***user***. The subdirectory ***main*** contains the MOFIA "mainframe" code (adapted from E787); this is the part that was not changed and is still in FORTRAN 77 format. Code in this directory has the extension *.F* and is compiled with the *f90* compiler as FORTRAN 90 *fixed format* code. The subdirectory ***mainf90*** contains the TWIST main analysis code. This subdirectory will also contain other software packages once they are developed and tested. In contrast, code that is still under development and testing, as well as utility code that is regularly modified by many users will reside in the ***user*** subdirectory. The code in the ***mainf90*** and ***user*** subdirectories carries the extension *.f90* and is therefore compiled with the *f90* compiler as FORTRAN 90 *free format* code. The ***photo*** subdirectory includes the code needed for the **ROOT** package (event display) which is written in C++. The ***photo*** directory also contains some FORTRAN subroutines to provide the link between the MOFIA FORTRAN code and the event display C++ code. The ***include*** subdirectory includes all the include files, common blocks, parameter files, and interface blocks. For convenience, the environment variables MOFIA_SOURCE, MOFIA_DUMMY, MOFIA_MAIN, MOFIA_MAINf90, MOFIA_PHOTO and MOFIA_USER are set to point to the source directory and its subdirectories. For example, the command

cd $MOFIA_USER

puts the user in the directory ***…/source/user***

When a module is compiled, the compilation process goes through two steps: first a file with extension *.mod* is created and then the *.o* file is created. The *.mod* file resides in the ***modules*** subdirectory. These *.mod* files are needed when other modules that *USE* them are compiled to enforce type checking on all procedure calls. Finally, the subdirectory ***dummy*** contains "dummy" modules and subroutines. These modules serve two purposes. First, for purposes of testing (or convenience) a certain part of the code might not be required. This is achieved by calling the equivalent dummy subroutine to replace the actual package. Second, the use of dummy modules provides a way to simplify the compiling and linking process when interdependencies between modules in different subdirectories are present. The dummy subdirectory also contains dummy subroutines for user specific code such as ***my_begin_run***, ***my_end_run***, ***my_init***, etc. When the user needs to modify these subroutines, they should be copied to the user subdirectory, modified, and added explicitly to the *Makefile* (in the ***user*** subdirectory). The compiler will first look for these files in the user subdirectory, if the specified files are not present, the compiler will find the *.o* files in ***…/lib/libdummy.a*** and link with them instead.

III. CVS

CVS is the version control system used for the TWIST software. CVS allows us to save the different versions of a source file, tag the source code to create a version of MOFIA when desired, and keep track of differences between the different versions of a file, as well as differences between files in the user's development directories and those in the CVS repository. The contents of the TWIST CVS code can be viewed on the web at

http://e614db.triumf.ca/cgi-bin/cvsweb.cgi

A detailed discussion of CVS as well as a detailed description of its commands may be found in the CVS manual. The following is a short list of the most commonly used CVS commands for easy reference.

The command

cvs checkout *argument*

may be used to checkout a file or a directory. It may also be used with qualifiers to specify the version number of the code to be checked out and the directory it should be placed in. For example, the command

cvs checkout -d 2.2 –r MOFIA-2-2 mofia

will create a subdirectory called "2.2" and place revision "MOFIA-2-2" of the "mofia" code inside it. This code will include all the directories of the MOFIA source code and their contents.

The command

cvs status filename

displays the status of a specific file. If the filename is omitted the status of all the files in the directory in which the command is entered are displayed. The user might often wish to check the status of only the files that differ from CVS in that particular directory. In this case the command

cvs status | grep –i need

is handy. This will pipe the cvs status command through the grep command to search for the files that need to be updated or committed (see below).

cvs update *filename*

This command allows the user to update their source code with new code from CVS. When no filename is specified, code in the entire directory in which the command is issued will be updated. If the user has modified files in his/her own directory, the CVS code will be merged with the user's code. In some cases conflicts may occur in which case CVS will prompt the

user. In such cases the user has to edit the file and resolve the conflicts by hand. Conflict sections are labeled by CVS with a “>>>>” and “<<<<”. Care must therefore be taken when using this command. In some cases users may want to save their modified files under a different name before updating from CVS in case they need to check differences, etc. The cvs update command lists the filenames in which differences are found as well as the code that differs for the user to make comparisons. It is often convenient to only list the filenames containing differences only (without displaying any code). This is achieved by using a qualifier

cvs update --brief

Notice that there are two “-“ signs preceding “brief”!

The Command

cvs commit filename

Is used to commit files to the CVS repository. The command will prompt the user to type in a description of the modifications made to the file. The editor defined by the environment variable CVSEDITOR is invoked for this purpose. The user may therefore find it handy to set this environment variable to his/her editor of choice in their own .login file. For example to have CVS invoke the emacs editor when using the cvs commit command, the following line would be needed to be entered prior to using the commit command (or better yet added to the user’s own .login file)

setenv CVSEDITOR emacs

The cvs commit command will only allow the user to commit files that already exist in the CVS repository. If the a new file is to be added to the repository the command

cvs add filename

Should precede the cvs commit command.

Files may also be removed from the CVS repository using the command

cvs remove filename

Followed by

cvs commit filename

All the above commands are best understood when they are tried.

The CVS manual may be found on the web at

http://

IV. Installing and Compiling MOFIA

All the MOFIA source code as well the setup files are committed to cvs. In order to have access to CVS, the user needs to be a member of the *e614cvs* group. This allows the user to checkout the MOFIA code. Detailed step–by-step instructions on installing MOFIA are posted on the web at

http://e614db.triumf.ca/~e614/triumf/doc/install.html

IV.1 Compiling MOFIA

Each of the subdirectories: ***main***, ***mainf90***, ***dummy***, and ***photo*** has its own *Makefile*. When the make file is executed, a library corresponding to this subdirectory is created (or updated) and placed in the directory ***mofia/2.2/lib***. The corresponding libraries are ***libmain.a***, ***libmainf90.a***, ***libdummy.a***, and ***libphoto.a***. Notice that the include files used by code in any of these subdirectories resides in the ***include*** subdirectory, and similarly the *.mod* files created when compiling code in any of these subdirectories resides in the ***modules*** subdirectory. In contrast, when the *Makefile* in the ***user*** subdirectory is executed, the *.mod* and *.o* files remain in the ***user*** subdirectory (no libraries are created).

Scripts are available to build MOFIA. These may be found in the directory

…/e614soft/triumf/mofia/2.2

for the MOFIA 2.2 version. In particular, the script make_all allows the user to build MOFIA from scratch. To do so, simply execute the command

make_all

Once MOFIA is built from scratch once, the user will only need to execute the *make* command in the directory where modifications have been made and the subsequent directories (which contain code that depends on these modifications). Often, the user will be modifying code only in the **user** subdirectory in which case only code in the ***user*** subdirectory needs to be compiled.

When compiling MOFIA (in the user subdirectory) the user has the option of making any of three different executables: **plain**, **mofia** or **photo**. The first executable, **plain**, does not contain any code from the ***mainf90***, ***user***, or ***photo*** subdirectories. The second executable, **mofia**, contains all of the MOFIA code except for the event display (***photo*** subdirectory). The last executable, **photo**, contains the entire code. To make any of these three executables the user needs to issue the command "*gmake executable*" where "*executable*" stands for **plain**, **mofia** or **photo**. For example,

make photo

makes the photo executable. If only the *gmake* command is issued (with no executable name) all three: **plain**, **mofia** and **photo** will be made and placed in the ***user*** subdirectory.

IV.2 Compiling a debug Version

The libraries corresponding to ***main***, ***mainf90***, ***dummy*** and ***photo*** in ***mofia/2.2/lib*** are all compiled with the *debug* flag off to allow faster execution. Since debug information may be needed at times, a debug version of all these libraries is provided in the subdirectory ***mofia/2.2/debug***. The default libraries are the non-debug ones, so that the environment variable ***MOFIA_LIBDIR*** is assigned to ***mofia/2.2/lib*** by default, and the *debug* flag is turned off when the code in the ***user*** subdirectory is compiled. To compile a debug version of MOFIA the user needs to issue the command

mlib f90 debug

Before compiling. This will assign the environment variable ***MOFIA_LIBDIR*** to ***mofia/2.2/debug*** and turn the debug flag on when compiling the ***mymofia*** code. If the user wishes to go back and recompile a non-debug version, the command

mlib f90

must be issued before executing the *Makefile*.

IV.3 Module Dependencies

It would be inefficient to recompile all the code whenever a file is modified in one of the ***source*** subdirectories. On the other hand, compiling only the file modified will not be sufficient in some cases since other files might depend on it. For example, if module A is changed, one needs to find all the modules and procedures that *USE* module A and recompile them. This is also true when any of the include files is changed. In each of the subdirectories ***main***, ***mainf90***, ***dummy*** and ***photo*****,** a file called ***Dependencies*** is supplied to serve this purpose. This file contains a list of modules and includes files that a *.o* file "depends on" so that if any of these files is changed the *.f90* (or *.F*) file corresponding to this *.o* file is recompiled. This information is made available by inserting the statement "include Dependencies" in every *Makefile* in the ***source*** subdirectories.

The ***Dependencies*** file itself is created by executing the script ***depend_f90.csh*** (which resides in the directory defined by the environment variable ***TRIUMF_ROOT***) from the ***source*** subdirectory in which the dependencies are to be found. This script goes through every file in this subdirectory and extracts all filenames that this file depends on (by examining all the *USE* and *INCLUDE* statements in that file).

It is important to remember that the list of object files in the Makefile has to be constructed in such a way so that the dependent files come after those they depend on.

Problems are sometimes encountered when compiling code only in a specific MOFIA subdirectory. While the origin of these compiling problems is not understood, it is found to be related to file dependency issues. If the user runs into compiling problems or runtime errors after only compiling code in a specific MOFIA subdirectory, the user can try to rebuild MOFIA from scratch using the make_all command discussed above.

V. Running MOFIA

As mentioned in the introduction, the framework for MOFIA was not changed from the previous versions. The commands in this section are therefore identical to previous versions of MOFIA.

V.1 General Commands

MTIN "argument": Assigns the input file/device to argument. For example,

MOFIA> ***MTIN "/ralph1/usr/data8/TWIST/data/run00092.dat"***

assigns the input file to *run00092.dat* in directory */ralph1/usr/data8/TWIST/data*. Similarly, the command

MOFIA> ***MTIN "/dev/mx3d"***

assigns the input to the tape drive */dev/mx3d*.

analyze "argument": Starts the analysis process. The number of events to be analyzed may be used as an argument; for example

MOFIA> ***analyze 1000***

analyzes 1000 events. If no argument is specified the entire file will be analyzed.

event "argument": Moves FORWARD to the event specified and analyze it. For example

MOFIA> ***event 237***

moves forward to event 237 and analyzes it.

show "argument": Shows the current contents of the argument. For example

MOFIA> ***show MTIN***

shows the name of the file/device that *MTIN* is assigned to. If no argument is provided, a listing of possible arguments is displayed.

@filename: Executes the command file specified by *filename*. Command files may include any MOFIA command that may be otherwise issued at the MOFIA command line. If the file extension is not specified the extension *.kcm* is assumed. Command files may also be nested (so that a command file may call another) up to 10 layers deep. For more information on command (*.kcm*) files please see the NAMELISTS section below.

show fail: prints out statistics showing the number of events failing a given event filter.

exit: Exit MOFIA.
help: Run the MOFIA help facility.

V.2 Flags

Several flags have been installed in MOFIA. To view these flags type ***show flags*** at the MOFIA command line.

```
MOFIA> show flags

 MOFIA FLAGS:

 BPRINT      = OFF
 PHOTO_FLAG = OFF
 SKIM        = OFF

        DECODE      TRACK      CUTS
 DC         OFF         OFF        OFF
 PC         OFF         OFF        OFF
 SC         OFF         OFF        OFF
 AP         OFF         OFF        OFF
 AS         OFF         OFF        OFF
 PU         OFF         OFF        OFF
```

The set command is used to change the contents of these flags. For example, to turn on the *PHOTO_FLAG* type ***set photo on*** at the MOFIA command line. To turn on the decoding for the DC detector subsystem type ***set dc on***. To turn on the tracking and/or the cuts, a qualifier following the subsystem name is needed. For example to turn on the cuts for the DC subsystem type ***set dc/cuts on***. Currently six subsystems are defined in MOFIA: (DC) drift chamber, (PC) proportional chamber, (SC) scintillators, (AP) proportional chambers ADCs, (AS) scintillator ADCs, and (PU) pulsars. Each subsystem is a CHARACTER(LEN=2). The command **show subsystems** allows the user to see the defined subsystems

```
MOFIA> show subsystems

  There are        6 defined Sub-Systems

  Index    Code    Description

     1      DC      Drift Chambers
     2      PC      Proportional Chambers
     3      SC      Scintillators
     4      AP      ADC for PCs
     5      AS      ADC for SCs
     6      PU      Pulsers
```

V.3 Namelists

Several namelists have been installed in MOFIA. The following commands are used to access these namelists.

***show namelist "argument"**:* Shows the namelist and its description. If no argument is provided all namelists and their descriptions are shown. If an argument is provided the contents of the namelist specified by the argument are shown as well as their respective description and default settings. For example

MOFIA> ***show namelist DCCUTS***
shows the contents of the namelist *DCCUTS*.

namelist "argument"**:* Accesses the namelist specified by argument for purposes of checking and/or modifying the current settings. Invoking this command puts the user inside the namelist editor. Once inside the editor, the user may check the current values of the namelist by entering "?***", exit the editor by entering "***&***", or change the value of a namelist variable by entering "***variable = value***". Below is an example:

```
MOFIA> name DCSET
Enter Drift Chamber SETtingings:
?
&NLDCSET
FIRSTPLANEDC =   1,
LASTPLANEDC =   44,
/
LASTPLANEDC = 22
&
MOFIA>
```

While these commands may be typed in interactively (on the MOFIA command line), it is more convenient to have them in a command file so that the user can execute the appropriate command file (which contains the appropriate settings, flags, cuts, as well as pointers to the appropriate calibration files) for the desired analysis. Five *sample* files are provided (and may be checked out from the ../source/user directory in CVS). The file ***helix.kcm*** contains settings appropriate for analyzing data with the magnetic field on, while ***str8.kcm*** contains field off settings. The files ***helix_mc.kcm*** and ***str8_mc.kcm*** contain the corresponding settings for analyzing Monte Carlo data. The file ***eff.kcm*** contains appropriate settings for computing the intrinsic efficiency of the chamber from straight tracks. To execute any of these files type ***@filename*** at the MOFIA command line. The commands in the file will show up on the screen and will also be saved to the log file ***mofialog.dat***.

Command files used to analyze MC data (such as *helix_mc.kcm*) should have the two namelist variables MonteCarlo and UnpackMC in the namelist GLOBAL set to true. **Calibration files to be used in analyzing MC data should also be specified in these**

command files, since CFM is not currently used to handle calibration files in the Monte Carlo.

V.4 Functions

Several functions have also been installed in MOFIA The command: ***func "argument***" allows the user to execute these functions. For example executing the command ***func 6*** prints out the drift and proportional chambers geometry files. If no argument is provided a listing of all functions and their descriptions is provided. **Table 1** contains a list of all the functions currently available:

Function	Description
1	Turn ON printing for list of wire hits
2	Turn OFF printing for list of wire hits
3	Turn ON printing for tdcunp debugging
4	Turn OFF printing for tdcunp debugging
6	Print DC and PC geometry files
9	Reset all histograms
10	Initialize cross talk counters
11	Print cross talk counters
12	Print efficiency counters
13	Print residuals
14	Initialize efficiency counters
20	Determine and output time zero

Table 1*. A list of functions available in MOFIA.*

V.5 Environment variables

Environment variables may also be assigned before executing MOFIA. Two convenient environment variables that come in handy for MOFIA are worth mentioning here: *MTIN* which specifies the input file/device as explained above, and *MHIST* which specifies the directory to which the hbook histograms should be written. If *MHIST* is not specified, the hbook histograms will be written to the current directory in which MOFIA is running. Some environment variables may be best to assign through the *.login* file. If multiple users are sharing the same account and wish to have their own definitions of environment variables, a file of the form *.user* (where *user* stands for the user name) would be handy. Each user would then have to *source .user* before running MOFIA.

When MOFIA is run, a log file is created with the default filename ***mofialog.dat*** and is written to the same directory in which MOFIA is running. The user may wish to change this default and can do so by assigning the environment variable *MOFIALOG* to the desired filename (which may include a directory name if the user wishes to write the log file to a different directory). This is particularly useful when the user wishes to save the log file, since otherwise this file will be overwritten the next time MOFIA is run.

The environment variable *MOFIA_INIT* provides yet another handy tool. When this variable is assigned to a MOFIA command file (for example, ***helix.kcm***) MOFIA will automatically execute this file upon entering. It is therefore handy for the user to assign this variable to a command file that contains the user customized settings.

VI. Calibration File Manager (CFM)

TWIST also adopted the Calibration File Manager (CFM) developed for BNL experiment E787. CFM allows the user to associate run numbers with calibration files. Calibration types (plane position corrections, wire position corrections, etc) may be defined through CFM. For example, *DC_PPC* defines the calibration type for the **D**rift **C**hamber **P**lane **P**osition **C**orrections. CFM allows up to 5 characters to identify a component (Drift Chamber, DC, in this case) and up to 3 characters to identify its attributes (Plane Position Corrections, PPC). The file name for a calibration type has the form *XXXXX_YYY.NNNNN* where *NNNNN* is a 5-digit (version) number called an *indicator*. For example *dc_ppc.00001* is a PPC file name with indicator number equal to 1. CFM is then asked to associate a run number, or a group of runs with indicator 1 for the calibration type *DC_PPC*. When MOFIA is run it checks the run number and obtains the *indicator* number and the file name for each calibration type from CFM. CFM expects all calibration files to reside in the directory defined by the environment variable ***CAL_DB***. All calibration files are in ascii format which allows the user to easily display and edit their contents. To run CFM type ***CFM*** (outside MOFIA). Commands may be issued at the CFM command line to view the contents of CFM or modify and change the existing information. The ***help*** command provides the user with online help.

Three CFM commands are handy for viewing the contents of CFM and are worth mentioning here. The command ***show types*** displays the calibration types defined in CFM as shown in the example below

```
CFM> show types

 1:DC_PPC    2:DC_WPC    3:DC_RES    4:DC_PRC    5:DT_GEO    6:FBC1_MAP
 7:FBC2_MAP  8:FBC3_MAP  9:DC_PZC   10:DC_WZC   11:DC_WRC
12:DC_STR 13:PC_T0      14:DC_T0    15:SC_T0    16:PC_ADC   17:SC_ADC
```

These calibration files are described in **Table 2.** Another useful command is **show set n** (where n equals the desired set number). This command will show the run numbers associated with set ***n*** as well as the calibration file names associated with set ***n*** as shown in the example below.

```
CFM> show set 15
--------------------------------------------------------------------------
CDF Set  Run Total  Run Range
-------  ---------  ---------
  15         0
CDF Names:
  DC_PPC.00003    DC_STR.00010    DC_WZC.00003  FBC2_MAP.00018
  PC_T0.00004 DC_PRC.00003    DC_T0.00006    DT_GEO.00024  FBC3_MAP
  SC_ADC.00001 C_PZC.00003    DC_WPC.00003  FBC1_MAP.00018
  PC_ADC.00001    SC_T0.00013   DC_RES.00004    DC_WRC.00003
```

Type Number	Type	Description
1	DC_PPC	Drift chamber UV plane position corrections
2	DC_WPC	Drift chamber UV wire position corrections
3	DC_RES	Drift chamber resolution parameters
4	DC_PRC	Drift chamber plane rotation corrections
5	DT_GEO	Detector Geometry
6	FBC1_MAP	Mapping file for first FASTBUS crate
7	FBC2_MAP	Mapping file for second FASTBUS crate
8	FBC3_MAP	Mapping file for third FASTBUS crate
9	DC_PZC	Drift chamber plane Z position corrections
10	DC_WZC	Drift chamber wire Z position corrections
11	DC_WRC	Drift chamber wire rotation corrections
12	DC_STR	Drift chamber space-time relations
13	PC_T0	Proportional chambers time zero
14	DC_T0	Drift chambers time zero
15	PC_T0	Proportional chambers
16	PC_ADC	Proportional chambers ADC calibrations
17	SC_ADC	Scintillators ADC calibrations

Table 2. Calibration types currently installed in the Calibration File Manager (CFM).

When a set is displayed with no indicator number (as in FBC3_MAP in the example above) it means that although the type has been defined, no file has been associated with it for the set displayed. The command ***show set*** (with no set number provided) displays all sets currently defined in CFM. Finally the command ***show runs*** shows which runs are associated with the different sets as in the example below

```
CFM> show runs
     Runs       Total  CDF Set
-------------------------------
    1 :    4      4       1
    5 :    6      2       2
    7 :   20     14       3
   21 :  499    479      --
  500 :  800    301       4
  801 :  922    122       5
  923 :  924      2       6
  925 :  932      8       7
  933 :  955     23       8
   956            1       9
  957 : 1311    355       8
 1312 : 1500    189      12
 1501 : 1693    193      --
 1694 : 2499    806      13
 2500 : 5000   2501      19
```

For a more detailed description of CFM, please refer to the E787 document by Morgan Burke at

http://e614db.triumf.ca/~e614/e614slow/offline/cfm/index.html

VII. Initialization Branch

The MOFIA initialization branch may be divided into two parts: geometry implementation and histogram definitions as shown in **Figure 1**.

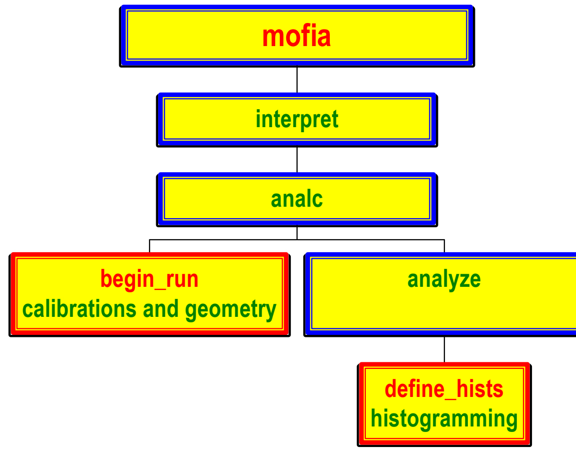


Figure 1 MOFIA initialization branch.

VII.1 Geometry

The MOFIA XYZ coordinate system is defined to be right handed with the +Y direction pointing upwards and the +Z direction defined by the beam direction. The UVZ coordinate system is obtained through a clockwise rotation of the XYZ system by a $+45^0$ rotation around the Z-axis as shown in **Figure 2**.

The geometry description of the TWIST detector is read in from an *ascii* data file common to both the Monte Carlo and MOFIA to ensure consistency. The geometry input file name has the form *dt_geo.NNNNN*, where *NNNNN* is the indicator number (see section III). The geometry input file is managed through the Calibration File Manager ***CFM***, where the association between version numbers and run numbers is made. This allows us to keep track of any geometry changes in the TWIST detector.

The module ***mainf90/det_geom_mod.f90*** contains the PUBLIC subroutines ***OpenGeom***, which opens the appropriate geometry data file for the run number at hand (by consulting with CFM), and calls the function ***read_det_geom*** which reads in the geometry data. The geometry data file contains four sections for drift chamber geometry, proportional chamber geometry, scintillator geometry, and target geometry. Each of these sections is read by a function called by ***read_det_geom: read_dc_geom, read_pc_geom, read_sc_geom, and read_tg_geom.*** These functions are all PRIVATE and internal to the module ***det_geom_mod***. The geometry information is saved in PUBLIC variables declared in this module which carry the same names as the

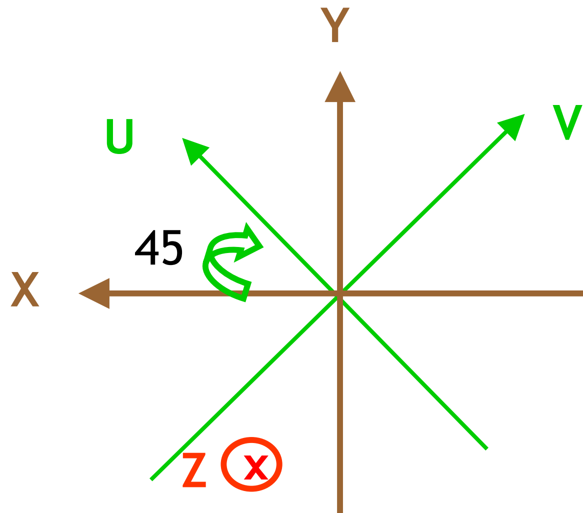


Figure 2 UV coordinate system relative to XY.

corresponding Monte Carlo variables. In the Monte Carlo these variables are stored in the common block ***det_geom.inc*** and the parameter file ***det_geom.par***. **Figure 3** shows a block diagram for the code organization in the geometry initialization branch.

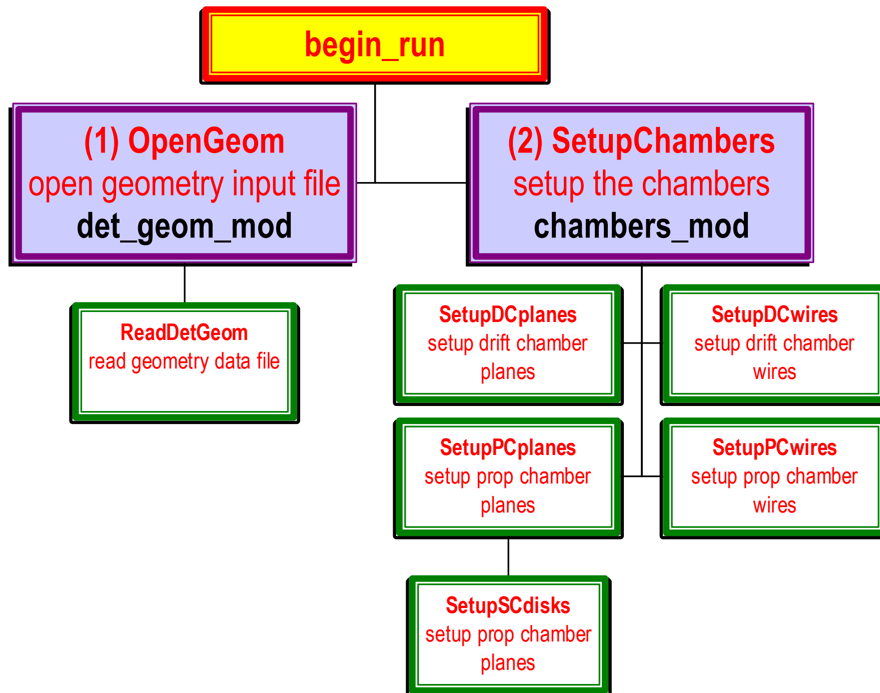


Figure 3 MOFIA geometry branch.

The module ***mainf90/chambers_mod.f90*** is where all the geometry structures for the drift

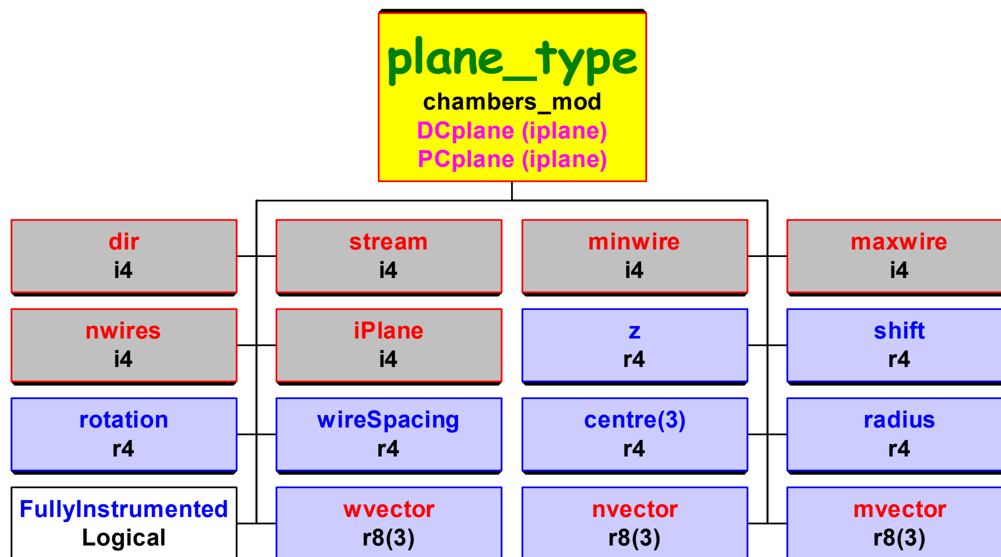


Figure 4 Components of the plane_type geometry structure.

chamber and proportional chamber are defined and filled. These structures are shown schematically in **Figure 4**.

The two types of structures, *plane_type* and *wire_type*, include the plane and wire geometry structures, respectively. Each type has two instantiations, one for the drift chamber and one for the proportional chamber: ***DCplane(iPlane)***, ***PCplane(iPlane)***, ***DCwire(iPlane,iWire)***, and ***PCwire(iPlane,iWire)***. **Table 3** contains a brief description of the contents of the *plane_type* structure.

Plane_type	Description
Dir	Coordinate measured by plane (U or V)
Stream	Plane location (upstream or downstream)
MinWire	Number of the first wire in plane
MaxWire	Number of the last wire in plane
NWires	Total number of wires in plane
Z	Z position of plane
Shift	U or V position of plane
Rotation	Angular orientation of plane
IPlane	Plane number
wireSpacing	Spacing between wires in plane
Center(3)	Coordinates of plane center
radius	Plane radius
fullyInstrumented	Logical indicating whether all wires in plane are instrumented

Table 3 A brief description of the components in the geometry structure *plane_type*.

The following are examples of how the first four components of *plane_type* are used. See ***chambers_mod*** for more details.

```
IF (DCplane(iPlane)%dir == dir%u) THEN

IF (PCplane(iPlane)%stream == stream%up) THEN

DO iWire = DCplane(iPlane)%MinWire, DCplane(iPlane)%MaxWire
```

The structures, ***dir*** with components ***u*** and ***v*** (*dir%u,dir%v*) and ***stream*** with components ***up*** and ***down*** (*stream%up,stream%down*), have also been publicly declared ***chambers_mod*** and are available for usage by any module or procedure that uses ***chambers_mod***. The planes are labeled according to the coordinate they measure so that "U-planes" measure a U coordinate while "V-planes" measure a V coordinate. The component ***rotation*** contains the angular orientation of the plane. Four different angle orientations appear in the geometry file. This implementaion is necessary to achieve consistency with the hardware labeling of wires. In order to define a single coordinate system for all planes (upstream and downstream) as well as maintain the wire numbering scheme assigned in hardware, DC V-planes in the upstream half of the detector must have increasing wire numbers in the -V direction, and DC U-planes in the downstream half of the detector must have increasing wire numbers in the -U direction. The U and V planes are made out of X- planes by rotating them around the Z-axis, as shown in **Figure 5** resulting in the four angular orientations listed in the figure.

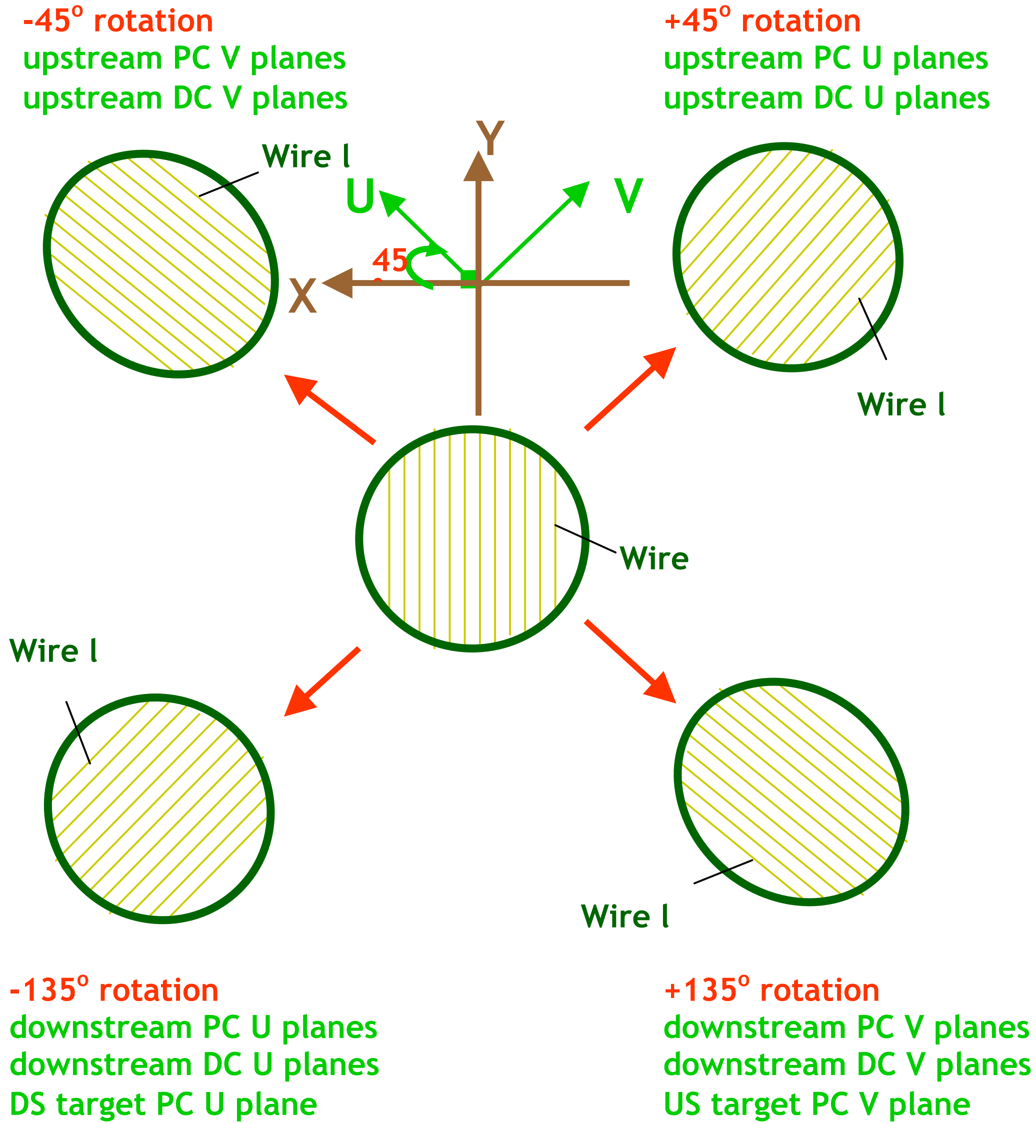


Figure 5 Construction of U and V planes from X planes in GEANT and MOFIA. The angles indicated are the ones that appear in the geometry file

While the component ***iPlane*** is obviously redundant when used in the form *DCplane(iPlane)%iPlane*, the reason for putting it in the structure is to allow accessing the plane number through a pointer in the hit structure, and its usefulness will become evident when the hit structure is discussed below.

Table 4 contains a brief description of the components of *wire_type*. The components ***iWire*** and ***planeP*** were introduced in this structure for the same reason that ***iPlane*** was introduced in the *plane_type* structure, and their usefulness will also become evident when the hit structure is discussed.

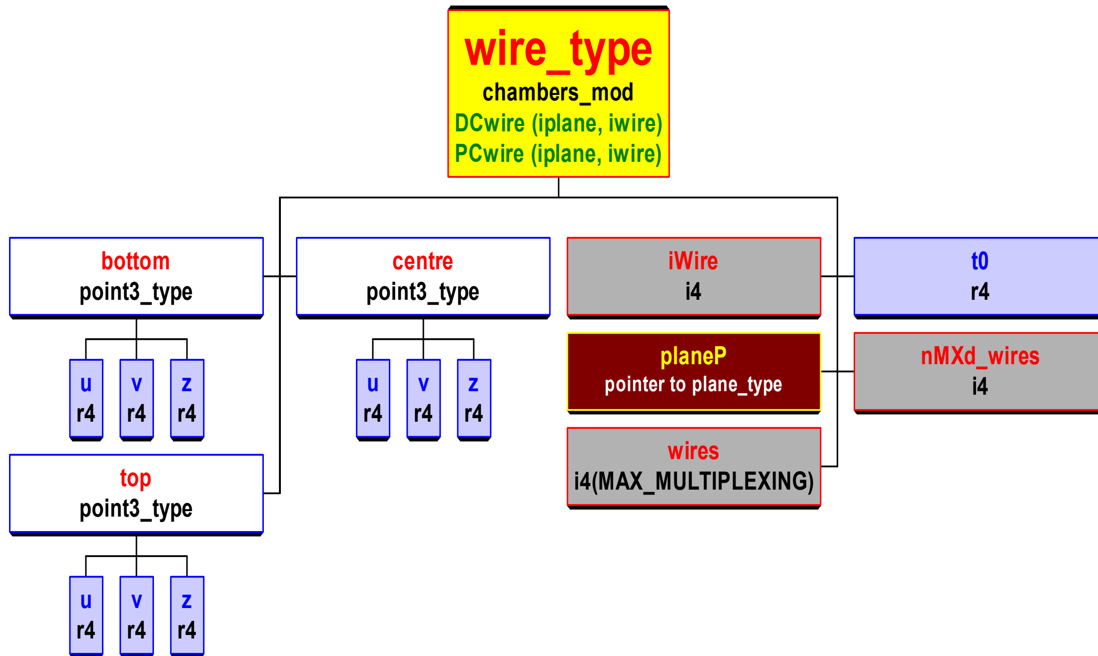


Figure 6 Components of the wire_type geometry structure.

As shown in **Figure 3**, access to ***chambers_mod*** is achieved through a call from ***mainf90/begin_run.f90*** to ***chambers_mod*** PUBLIC subroutine ***SetupChambers***, which in turn calls PRIVATE subroutines within ***chambers_mod*** that do the job of filling up the geometry structures (***SetupDCplanes, SetupPCplanes, SetupDCwires, SetupPCwires*** and ***SetupSCdisks***). This reflects a general philosophy employed in developing the code, namely minimizing the number of entry points to the module by having a few (preferably one) PUBLIC subroutines to be called from outside the module, while the remaining subroutines in the module are made PRIVATE and can, therefore, only be called from within the module. The ***chambers_mod*** has a second entry point for testing purposes. The PUBLIC subroutine ***PrintGeom***, is accessed through a call from the subroutine ***func***, and maybe invoked by typing **func 6** at the MOFIA command line. When invoked, ***PrintGeom*** calls private subroutines within ***chambers_mod*** that create geometry ouput data files for purposes of testing the geometry information (*dc_planes_geom.out, pc_planes_geom.out, dc_wires_geom.out, and pc_wires_geom.out).*

wire_type	Description
bottom	Wire “bottom” end point position coordinates (bottom%u,bottom%v,bottom%z)
center	Wire central position coordinates (center%u,center%v,center%z)
top	Wire “top” end point position coordinates (top%u,top%v,top%z)
iWire	Wire number
planeP	Pointer to *plane_type*

Table 4 A brief description of the components in the geometry structure wire_type.

It is worth noting that the geometry file does not contain any information on wire positions and orientations. These values are calculated in the subroutines ***SetupDCwires*** and ***SetupPCwires*** using plane positions and orientations provided in the geometry data file and the wires nominal separation (of 0.4 cm for the DCs and 0.2 cm for the PCs). Corrections to both, planes nominal positions and wires nominal positions, are read in from calibration files and implemented in ***chambers_mod*** as corrections to the nominal positions. Procedures in the module ***mainf90/calibrations_mod.f90*** are accessed through a call from ***mainf90/begin_run.f90*** to the module's PUBLIC subroutine ***ReadCalibFiles***, which in turn calls PRIVATE procedures within the module to open the appropriate calibration files for the run at hand (through calls to CFM) and read them in. Currently 11 subroutines are called from ***ReadCalibFiles*** corresponding to 11 calibration types that are defined in CFM, these subroutines are: ***CalibPlaneCorrUV, CalibPlaneCorrZ, CalibPlaneCorrRot, CalibWireCorrUV, CalibWireCorrZ, CalibWireCorrRot, CalibSTR, CalibT0, CalibEff, CalibRes, CalibADC.*** Some of these calibrations, however are either not implemented in MOFIA or contain trivial data since their contents have not been determined yet.

Figure 7 shows the calibrations data structures related to the chamber's geometry. As shown in this figure, these corrections include U or V and Z position shifts for each DC plane and wire, as well as rotation corrections for each DC plane and wire. These corrections are included in the geometry structures in ***chambers_mod***, so that the components ***Zshift***, ***UVshift*** and ***rotation*** in the *plane_type* structure already have these corrections built in.

The contents of the calibrations structures are accessed in ***chambers_mod*** through a *USE* statement (*USE calibrations_mod*), and the appropriate corrections are made to the geometry variables.

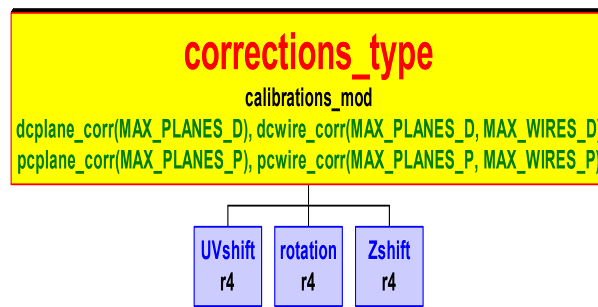


Figure 7 Corrections structures for geometry calibrations.

VII.2 Histogramming

TWIST adopted HBOOK for histogramming purposes since it is a package that many members of the collaboration are familiar with. Since users tend to define a large number of histograms for purposes of testing their code or analyzing some specific data, in many cases keeping these histograms as part of the official TWIST code is not practical. For one thing, CPU time will be wasted filling in histograms that most users don't need; and for another, the number of histograms created becomes large so that sorting through the histogram list to find

the desired histogram becomes tedious. The decision was therefore made to provide two histogramming modules. The first, ***mainf90/hists_mod.f90***, is the module that contains the official histograms that are to be kept and used by all users. The second module, ***user/user_hists_mod.f90,*** is where each user defines their own histograms. The user is responsible for keeping and maintaining their own copy of this module. If you add your own histograms make sure you don't overwrite your own copy of ***user_hists_mod.f90*** when you update your code; save this module under a different name before updating your code. The histogramming branch is shown in **Figure 8**.

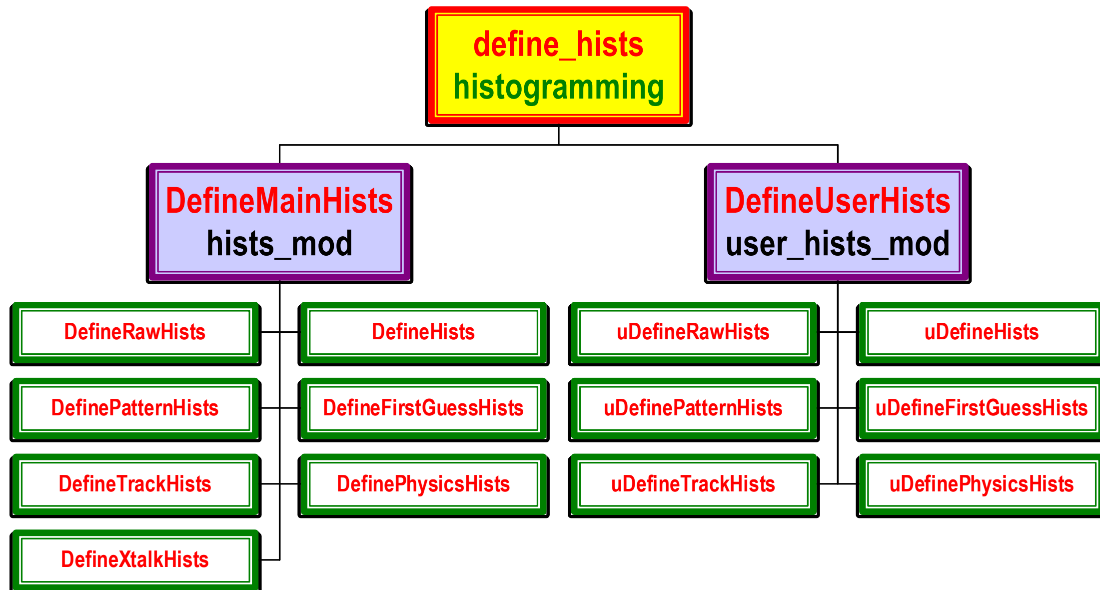


Figure 8. MOFIA histogramming branch.

The subroutine ***main/define_hists.F*** makes two calls, one to the PUBLIC subroutine ***DefineMainHists*** in ***mainf90/hists_mod.f90***, and the other to its counterpart, ***DefineUserHists*** in ***user/user_hists_mod.f90***. These subroutines, in turn, call PRIVATE subroutines within the module, one for each section of the code: ***DefineRawHists*** to define raw histograms, ***DefineHists*** to define histograms after initial filtering, ***DefineXtalkHists*** to define cross talk histograms, ***DefinePatternHists*** to define pattern recognition histograms, ***DefineFirstGuessHists*** to define helix fit "first guess" histograms, ***DefineTrackHists*** to define tracking histograms, and ***DefinePhysicsHists*** to define physics analysis histograms. As seen from figure 5 these subroutines have similar names in the ***hists_mod*** and ***user_hists_mod*** except that the letter "u" (for user) is pre-pended to the name in the ***user_hists_mod***. The call to each one of these sections is controlled by a namelist flag in the namelist **hist** which allows the user to turn off that section so that histograms are neither defined nor filled. For example setting "FillRawHists = .FALSE." will bypass the call to ***DefineXtalkHists.*** The namelist **hist** contains several other useful variables; the command "show name hist" displays the list

MOFIA> ***show name hist***

NameList HIST: HISTogramming parameters

nEVTprocessed = -1 (dflt = -1,OFF): Write hists every nEVTprocessed events

FillRawHist = T (dflt = T) : Fill raw histograms

FillHist = T (dflt = T) : Fill histograms after initial filtering

FillTrackHist = T (dflt = T) : Fill tracking histograms

FillPatternHist = T (dflt = T) : Fill pattern recognition histograms

PulserHistToggle = T (dflt = F) : Fill histograms with random pulser data(T) or normal triggers(F)

FillPhysicsHist = T (dflt = T) : Fill physics histograms

FillFirstGuessHist = F (dflt = T) : Fill helix first guess histograms

FillXtalkHist = T (dflt = T) : Fill cross talk histograms

PlaneHists = T (dflt = T) : Generate individual plane histograms

Globalmem_toggle = T (dflt = T) : Turn Global Memory Section on (T) /off (F)

TDC_MIN = 0. (dflt = 0.) : Lower limit on TDC spectra

TDC_MAX = 6000. (dflt = 6000.) : Upper limit on TDC spectra

Raw_XMI_tdcslot = 0. (dflt = 0.) : Lower limit on RAW TDC plots

Raw_XMA_tdcslot = 30000. (dflt = 30000.) : Upper limit on RAW TDC plots

Raw_NX_tdcslot = 3 (dflt = 3) : Number of RAW x Channels for TDC plots

WEvent_per_plane = F (dflt = F) : Turns on/off #of wires hit/event/plane hists

HMult_times = F (dflt = F) : Turns on/off Time difference between multiple hits on a wire hists

To avoid conflicts between the user defined histogram numbers and the main histogram numbers, values between 1 and 50,000 are reserved for the main histograms; user histograms should always have numbers higher than 50,000. To avoid conflicting histogram numbers between the main histograms themselves, a range of histogram numbers has been reserved for each section of the code, as specified on the chart of **Figure 8**. Histogram numbers are assigned to variables in the declaration section of the histogramming modules. This has two advantages. First, if a histogram number is to be changed it only needs to be done once (rather than once where the histogram is defined and once where the histogram is filled) which reduces the potential for mismatches. Second, if this list is maintained in ascending order it becomes easy to see which histogram numbers have already been used and reduces the risk of conflicts.

The module ***hists_mod.f90*** (***user_hists_mod.f90***) also contains a set of PUBLIC subroutines for filling histograms. These are ***FillRawHists***, ***FillHists***, ***FillXtalkHists***, ***FillPatternHists***, ***DefineFirstGuessHists***, ***FillTrackHists***, and ***FillPhysicsHists***. Each of these subroutines is (or otherwise should be) called from the appropriate module where the corresponding calculations are made.

To define a new histogram the user should start by declaring an ID parameter for the histogram in ***mainf90/hists_mod.f90*** (or ***user/user_hists_mod.f90*** if the histogram is not intended to become part of the official code). For example, to define a raw histogram containing the TDC spectra of all the wires in one histogram we declare the parameter IDH_TDC_ALL to be the histogram's ID

```
INTEGER (i4), PARAMETER :: IDH_TDC_ALL  = 3
```

We then install a call to the HBOOK subroutine "HBOOK1" to define a 1-D histogram in the subroutine ***DefineRawHists***

```
! TDC time (total)
CALL HBOOK1 (IDH_TDC_ALL, 'DC TDC TIME', TDC_MAX-TDC_MIN, TDC_MIN, TDC_MAX, 0.0)
```

The first parameter in the call to hbook1 is the histogram ID. The second is a description of the contents of this histogram to be displayed as a histogram label. The third parameter is the number of bins, which in this example is the expression TDC_MAX-TDC_MIN. The next two parameters are the histogram's lower and upper limits, respectively (TDC_MIN and TDC_MAX). To fill this histogram a call is made to the HBOOK subroutine HFILL from the subroutine ***FillRawHists***

```
! RAW TDC spectra for all wires in one histogram
CALL hfill (IDH_TDC_ALL, timeP, 0.0, 1.0)
```

The first parameter in this call is the histogram ID (as in the call to ***hbook1***). The second parameter is the variable containing the TDC time, in this case a pointer to the TDC time, *timeP*. Note that this variable has to be real, if an integer is to be plotted the variable has to be converted to real first. For example if *timeP* was a pointer to an integer variable the second parameter in the ***hfill*** call should be *REAL(timeP)* rather than timeP. Failure to do so will result in run time errors.

VIII. Analysis Branch

A schematic view of the MOFIA analysis branch is shown in **Figure 8**. The procedure ***dplot*** is the TWIST event analysis subroutine. From ***dplot*** calls are made to analyze the event starting with the TDC unpacking, filling the histograms, filtering the event, etc.

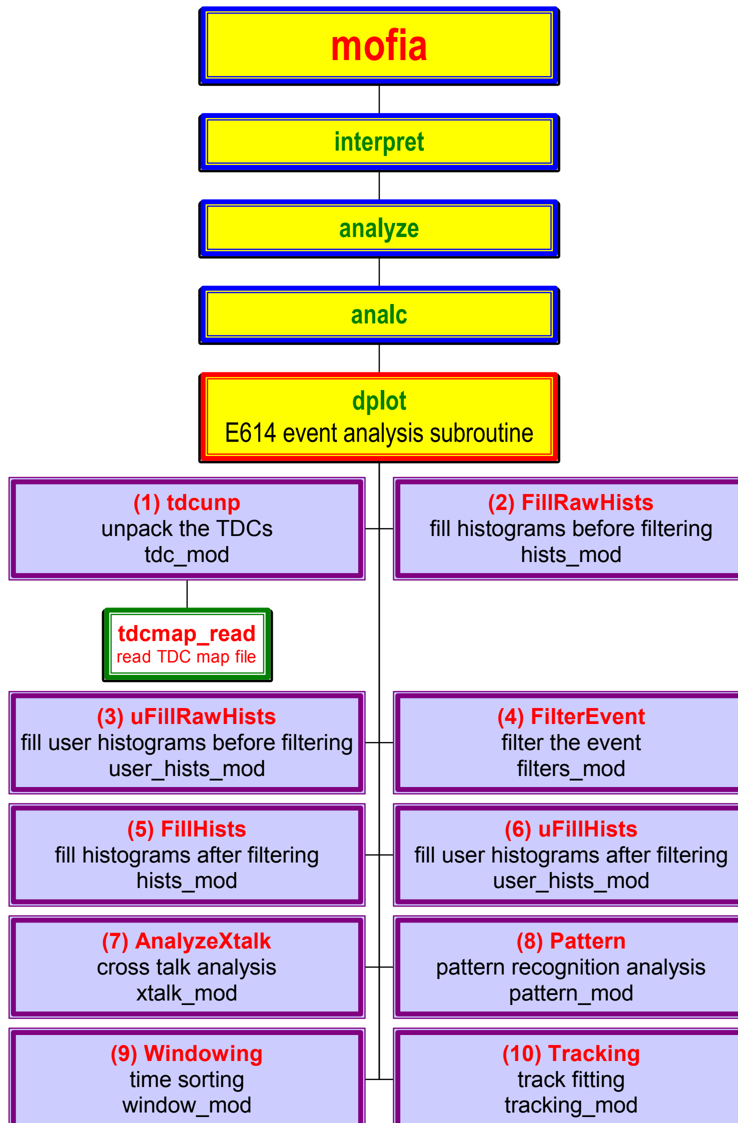


Figure 8 MOFIA analysis branch.

VIII.1 TDC Unpacking

From ***dplot*** a call is made to the PUBLIC function ***tdcunp*** in module ***tdc_mod***, in order to unpack the TDCs. This module also contains the declarations and initializations (and filling) of the data structures associated with the TDC hit structures which are shown in **Figure 9**. **Table 5** shows a brief description of the *tdc_type* structure. This type has two instantiations, ***DCtdc(ihit)*** and ***PCtdc(ihit)***. The first two components of the structure are the time and width of the TDC signal. The time stored in these structures is in nanoseconds, and is measured relative to the delayed (by ~10 μs) trigger signal.The third component, ***flag***, is an integer that is assigned a zero value for normal TDC signals, and a non-zero value if the TDC signal has a peculiar characteristic (such as a leading edge but no trailing edge, etc). **Table 6** contains a listing of these flags. The last two components in this structure are two pointers. The first, ***wireP***, points back to the wire geometry structure and, therefore, provides the link between the hit structure and the geometry structures. To point to the wire and plane numbers for a particular DC TDC hit, for example, we have

```
INTEGER (i4), POINTER :: iwP, ipP

iwP => DCtdc(ihit)%wireP%iwire

ipP => DCtdc(ihit)%wireP%planeP%iplane
```

and so on. In the above example we could have defined integer variables iw and ip (instead of pointers iwP and ipP); in which case we have

```
INTEGER (i4) :: iw , ip

iw = DCtdc(ihit)%wireP%iwire

ip = DCtdc(ihit)%wireP%planeP%iplane
```

However, since copying data from a structure to a new variable is generally less efficient than using a pointer, it is recommended that the above style be used.

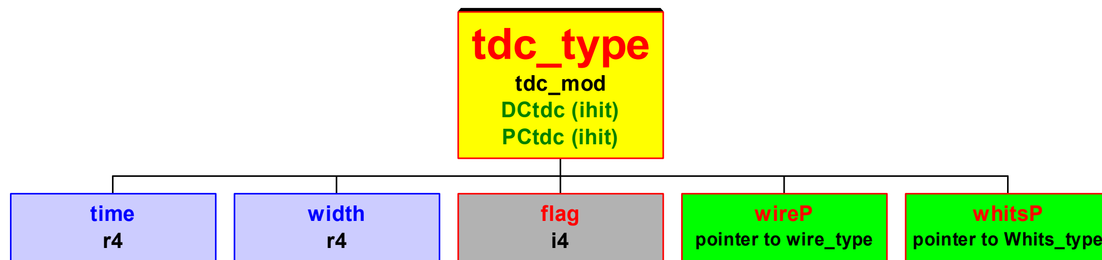


Figure 9 MOFIA TDC hit structure.

tdc_type	Description
Time	TDC signal leading edge
width	TDC signal width
Flag	Characteristic of the TDC signal (described in table 3)
wireP	Pointer to wire geometry structure wire_type
whitsP	Pointer to wire hits structure whits_type

Table 5 A brief description of the components of the hit structure tdc_type.

flag Value	Description	*width* Value	Details
0	Good hit		Trailing edge followed by leading edge on same channel
1	No leading edge	0	Trailing edge following another trailing edge
2	No leading edge	0	Last edge on channel, but it's a trailing edge
3	No leading edge	0	Last edge was a trailing edge from a different channel
4	No trailing edge	99999	First edge on channel, but it's a leading edge
5	No trailing edge	99999	Leading edge following another leading edge
6	No leading edge	0	Last edge on channel is a trailing edge
7	No trailing edge	99999	First edge on channel is a leading edge
7	Width < 0	0	Trailing edge followed by leading edge, but width < 0

Table 6 Description of the ***flag*** values in the *tdc_type* structure.

The second pointer in the *tdc_type* structure, ***whitsP***, provides the link to the *whits_type* structure shown in **Figure 9**. This structure is also defined and filled in the module *tdc_mod*. The first element in this structure, ***nhits***, is the number of TDC hits on that wire and the second element, ***hits(index)***, contains the hit number as it appears in the hit list in the *tdc_type* structure. This is best understood through an example. The number of hits on wire 32 in DC plane 26 is then

```
DCwhits(26,32)%nhits
```

To assign two pointers, ihitP and timeP, to the hit index and TDC time for the *second* hit on this wire we have

```
INTEGER (i4), POINTER:: ihitP

REAL (r4):: timeP

ihitP => DCwhits(26,32)%hits(2)

timeP => DCtdc(ihitP)%time
```

The above example demonstrates another case of assigning pointers to a component of a structure. While the last two lines may have been combined in one, so that

```
timeP => DCtdc(DCwhits(26,32)%hits(2))%time
```

the style of assigning a pointer makes the code more readable. It is also more efficient, since typically such statements would be inside do loops (looping over planes, wires, hits, etc), and hence the hit number is extracted from the structures **once** by assigning it to a pointer(ihitP, in this case), and the pointer is then used within that loop from that point on. Another useful technique is to assign a pointer to a structure (as opposed to a component of a structure in the examples above). This also provides faster execution time; and improves the readability of the code. For example, one can declare the pointer whitP and assign it to a structure

```
DO iPlane = 1, Ndplanes

   DO iWire = 1, Ndwires(iPlane)

      whitP => DCWhits(iPlane,iWire)

      nhitsP => whitP%nhits

   END DO

END DO
```

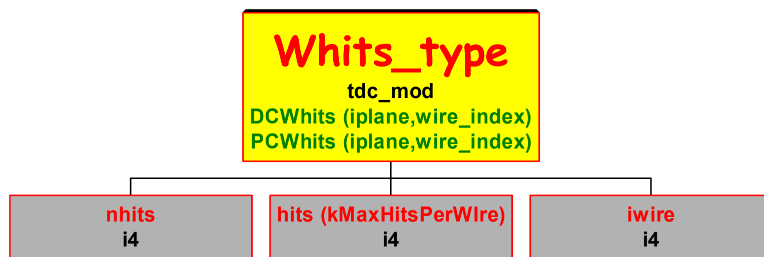


Figure 10 MOFIA wire hits structure.

Scintillator hit times are stored in the *SCtdc* structure which is similar to the TDC structure above, and differs only in that the *scintP* pointer to the scintillator geometry replaces *wireP*.

The two structures *SCadc* and *PCadc* hold ADC data from the PACT modules, unpacked and interpreted to give the energy deposited in the detector. These structures are similar to the TDC structures above with the two components time and width replaced by the component *e_lost* which holds the energy deposited in KeV.

Additionally the raw TDC data is unpacked into *DCtdc_raw*, *PCtdc_raw*, *SCtdc_raw*, *PUtdc_raw*, *PCadc_raw* and *SCadc_raw* whenever the namelist variable rawout > 0. This is intended for use in debugging and possibly determination of time calibration. The structure *PUtdc_raw* holds pulsar information.

Procedures to add and remove entries from these structures, as well as the type definitions, are found in *tdc_mod.f90* in the directory **mainf90**. Tables relating scintillator numbers, plane and wire numbers, as well as pulsar information to the TDC slot and address locations are handled in *tdcmap_mod.f90* in the directory **mainf90**. This code reads in the mapping information through CFM. The CFM types corresponding to these map files are FBC1_MAP, FBC2_MAP, and FBC3_MAP.

VIII.2 Filtering

Following the call to ***tdcunp***, a call is made to ***FillRawHists*** (and its user counterpart, ***uFillRawHists***) where some histograms are filled before any event filtering is done. The event is then filtered by calling the PUBLIC subroutine ***Filters*** in module ***mainf90/filters_mod.f90***, followed by a call to ***FillHists*** and ***uFillHists*** to fill some histograms after event filtering.

The public subroutine ***Filters*** makes calls to three PRIVATE subroutines: ***FiltersInit*** initializes the various counters used in this module, ***FiltersCounters*** calculates these counters, and ***FiltersApply*** is the subroutine where the event filters are applied. These filters include: scintillator filters, drift chamber filters, proportional chamber filters and RF filters. Several tests are performed on each event to determine whether it passes or fails a filter/cut. Values for these cuts are determined by the user through the namelist variables which will be discussed below. Counters are maintained for events failing a certain cut and statistics of these failures may be displayed by typing **show fail** at the MOFIA command line, as in the example below

MOFIA> ***show fail***

Event Statistics:

ICFAIL= 0 (GOOD EVENT) 2072 2072

ICFAIL= 2 (NO HITS IN CHAMBERS) 5 5

ICFAIL=11 (BAD EVENT) 684 684

ICFAIL=12 (BAD EVENT) 13 13

ICFAIL=14 (BAD EVENT) 225 225

ICFAIL=15 (BAD EVENT) 1 1

The first column of numbers shows the number of events failing the filters since the last ***analyze*** command was entered while the second column shows the total since the MOFIA session was started. In addition, a histogram (ID=5000) is incremented every time an event fails a filter. A description of the failure codes is documented in the appendix.

VIII.3 Cross Talk

Following event filtering a call is made to the PUBLIC subroutine ***Xtalk*** in the module ***user/talk_mod.f90***. ***Xtalk*** Calls two PRIVATE subroutines, ***XtalkInit*** which is used to initialize some counters, and ***XtalkAnalyze*** which performs an analysis of each hit to determine if it is a cross talk hit, increments the cross talk counters for each plane and wire in the DC, and removes the hit from the data structures if it is determined to be a cross talk hit. Three criteria are used to determine whether a hit is a likely cross talk hit. First, the hit is required to have a TDC width shorter than a user imposed value determined by the variable DC_XTALK_WCUT in the namelist **DCCUTS**. Second, the hit is required to occur in a cell adjacent to one with a hit that has a TDC width longer than DC_XTALK_WCUT. Third, the suspected cross talk hit is required to coincide in time with the hit in the adjacent cell. Note,

however, that the user must provide a value for DC_XTALK_WCUT, otherwise no cross talk analysis will be performed.

The user can initialize or print out the cross talk percentages at any time during a MOFIA session by typing **func 12** on the MOFIA command line. Output files will be created and a message will be displayed on the screen notifying the user of the file names.

VIII.4 Calibrations

The calibrations branches of MOFIA are controlled by a set of flags in the namelist **DCCFLAGS** (DC Calibration FLAGS). These branches are not normally executed when running MOFIA, and the user has to turn a specific flag on to execute a given calibration branch. The calibrations flags control MOFIA branches that are used to compute efficiency, plane positions, wire positions, plane rotations, DC chamber resolution and time zero.

MOFIA> ***show name dccflags***

NameList Alignment: Alignment parameters

FindPlanePos = F (dflt = F) : Find plane positions

FindWirePos = F (dflt = F) : Find wire positions

FindPlaneRot = F (dflt = F) : Find plane rotations

FindTDC0 = F (dflt = F) : Find TDC time zero

FindResolution = F (dflt = F) : Find drift chamber resolution

VIII.4.1 Efficiency

The efficiency code relies on the tracking to compute plane and wire efficiencies for both the DCs and the PCs. After a track is successfully reconstructed a call is made to the PUBLIC subroutines ***EffDC*** and ***EffPC*** in the module ***user/efficiency_mod.f90***. These subroutines make calls to ***EffDCinit*** and ***EffPCinit*** to initialize the efficiency counters, and to ***EffDCcalc*** and ***EffPCcalc*** to calculate the efficiencies. The call to ***EffDC*** and ***EffPC***, however, is controlled by the namelist flag FindEff in namelist **efficiency**, so that if this namelist variable is set to false the efficiency code will not get executed. In this case if the user attempts to output the efficiency counter (using ***func 12***) a message will appear on the screen informing the user that the FindEff flag has to be turned on before the data is analyzed. The ***show*** command may be used to display the contents of namelist **efficiency**

MOFIA> ***show name efficiency***

Namelist Efficiency: Efficiency calculation parameters

RadiusCutDense = 5.000 (dflt = 15 cm): DC dense stack radius cut

RadiusCutSparse = 15.000 (dflt = 15 cm): DC sparse stack radius cut

CellCut = 2 (dflt = 2): Max number of adjacent cells to investigate on each side of the cell expected to have a hit.

CellCut = 10 (dflt = 10): Minimum number of hit planes

FindEff = F (dflt = .FALSE.): Calculate chamber efficiency

The efficiency code uses the reconstructed track parameters to traverse through the detector and find the cells intersected by the track. It then checks whether these cells have hits and increments the appropriate counters shown in the structures diagram of **Figure 11**. In order to avoid edge effects (i.e. cases where the track has actually exited the active area of the chamber, but the track parameters point to the first/last cell due to tracking errors) two parameters are provided to impose an edge cut. These are RadiusCutDense which allows the user to set a radius cut on planes in the dense stack that have only 48 instrumented wires, and RadiusCutSparse to place a radius cut on the sparse stack in which all 80 wires are instrumented. The variable CellCut allows the user to determine how many adjacent cells should be investigated if no hit is found where the reconstructed track parameters point.

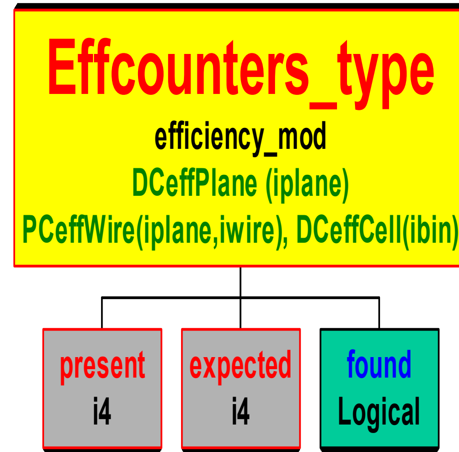


Figure 11MOFIA efficiency structure.

VIII.4.2 Time Zero

VIII.4.3 Alignments

Both rotational and translational alignments are currently implemented in MOFIA. When any of the variables FindPlanePos, FindWirePos or FindPlaneRot in the namelist **DCCFLAGS** is turned on a call is made to the PUBLIC subroutine Align in the module ***user/align_mod.f90***. ***Align*** calls ***AlignInit***, a PRIVATE initialization subroutine in the module, followed by a call to one or more of the PRIVATE subroutines ***AlignPlaneShifts***, ***AlignWireShifts***, and ***AlignPlaneRotations*** depending on whether the corresponding namelist flags are turned on. These are the subroutines that perform calculations of plane and wire translational position corrections (in the U and V directions) and the plane rotational corrections (around the z-axis). Once the plane translational and rotational alignments are determined, the results are stored in calibration files that are read into MOFIA through CFM. The CFM types corresponding to these calibrations are DC_PPC and DC_PRC, for the DC plane position corrections and plane rotation corrections, respectively.

VIII.4.3.1 TRANSLATIONAL ALIGNMENTS

After analyzing a number of events determined by the user through the variable nEventsPlane in the namelist **alignment** the subroutine ***AlignPlaneShifts*** is called to determine an average residual for each plane from tracks reconstructed successfully. The subroutine proceeds by installing the average residuals as a correction to the plane and wire positions. The tracking then proceeds as normal until another nEventsPlane are analyzed and the call is made again to ***AlignPlaneShifts*** to determine and install the new corrections. This iteration continues until the specified runs (or events) are analyzed. The user can also iterate on the same file by simply using the MOFIA commands to rewind and reanalyze the file as many times as the user desires. The variable FixPlanes in namelist **alignment** also allows the user to fix two U and two V planes if the user so desires, and to specify which planes should be fixed through the variables FixedPlane1, FixedPlane2, FixedPlane3 and FixedPlane4 in the same namelist. The corrections for the fixed planes will not be installed, instead their nominal positions will be used. The command ***show name alignment*** shows the contents of the alignment namelist.

MOFIA> ***show name alignment***

NameList Alignment: Alignment parameters

nEventsPlane (def = 10000) = 10000 Total number of events per iteration for calculating plane positions

nEventsWire(def = 100000) = 100000 Total number of events per iteration for calculating wire positions

AlignAngleU (def = 0.0) = 0.000 Angle of the U planes transverse-alignment line with respect to the beam

AlignAngleV (def = 0.0) = 0.000 Angle of the V planes transverse-alignment line with respect to the beam

FixPlanes (def = FALSE) = F Choose a line for transverse alignment defined by any 2 U and 2 V planes

FixedPlane1 (def = 1) = 1 Plane number for one of the four fixed planes used to define the alignment line

FixedPlane1 (def = 1) = 2 Plane number for one of the four fixed planes used to define the alignment line

FixedPlane1 (def = 1) = 7 Plane number for one of the four fixed planes used to define the alignment line

FixedPlane1 (def = 1) = 8 Plane number for one of the four fixed planes used to define the alignment line

VIII.4.3.2 ROTATIONAL ALIGNMENTS

For rotational alignments the subroutine ***AlignPlaneRotations*** is called once nEventsPlane are analyzed. In this case an average residual is determined for each plane, binned in 7 bins along the length of a wire. For each plane the average residuals in each length bin can be used to determine the plane rotational correction.

VIII.4.4 Resolution

Resolution for DME is strongly dependent on drift distance being mainly influenced by two processes: ionization statistics and diffusion. At distances close to the wire ionization statistics dominate, while diffusion becomes dominant near the edge of the

cell where the electric field is weak. The strong variation in resolution across the cell requires the binning of the residuals along the distance from the wire. The resolution code currently uses a bin width of 100 μm.

When the calibration flag FindRes in the namelist **DCCFLAGS** is turned on, a call is made to the PUBLIC subroutine ***Resolution*** in module ***user/resolution_mod.f90***. This subroutine makes a call to ***ResolutionDC*** which proceeds to calculate the resolution by examining the residuals in each of the distance bins. One of three methods can be chosen by the user to analyze the residuals: fitting the binned residuals to a gaussian, performing a squared sum calculation, or determining the FWHM of the binned distributions. Each of these methods has its advantages and disadvantages. For example, the gaussian method would be appropriate if the resolution is constant within a certain bin (or the bin width is infinitesmal). Details on the resolution calculations will be provided in a technical note.

The method used to analyze the residuals is determined by the variable calcResidualMethod in namelist **rezs**. Also in this namelist the user can choose the number of events required to be analyzed before the residuals are calculated. This is determined by the variable nEventsMax.

Once the resolution (defined by σ of a guassian fit, σ from a squared sum, or σ from FWHM) is determined, the tracking proceeds again using the new resolutions in the track fitting until nEventsMax are analyzed at which point the subroutine ***Resolution*** is called again to determine and install the new resolution. The iterations continue until all the specified events/runs are analyzed with each iteration using the resolutions determined from the previous iteration in the track fit.

The command ***show name rezs*** shows the contents of the **rezs** namelist

MOFIA> ***show name rezs***

Namelist REZS: Resolutions

nEventsMax = 5000 (dflt = 5000): # Events per Resolution Iteration

MinHistFitEntries = 1000 (dflt = 1000): # histogram entries necessary to do fit

nResidualBins = 300 (dflt = 300): # Bins in residual histograms.

MinResidualValue = 0.30 (dflt = -0.3): Min histogram channel (cm).

MaxResidualValue = 0.00 (dflt = 0.3): Max histogram channel (cm).

*calcResidualMethod = *** (dflt = 3): Residual calculation method.*

1=> Gaussian Fit to residual histograms.

2=> SquaredSum calculation.

3=> FWHM calculation.

VIII.5 Pattern Recognition

The purpose of the pattern recognition is to assign hits to tracks and provide starting track parameters to the fitting routine. For information on the pattern recognition code please refer to the document by Jim Musser at

http://

VIII.6 Tracking

Both a χ^2 fit and a Kalman filter are used for tracking purposes. Currently the Kalman filter is used for straight track fitting and the χ^2 fit is used for helix track fitting. The intention is to modify the Kalman filter in the future to handle helix tracks as well.

VIII.6.1 χ^2 Fit

Details on the χ^2 fit may be found on the web in the document written by Konstantin Olchanski at

http://

VIII.6.2 Kalman Filter

Details on the Kalman filter may be found on the web in the document written by Maher Quraan at

http://

X. Appendecies

X.1 Namelist Variables

MOFIA> **show name**

NAMELIST	**DESCRIPTION:**
BATCH	BATCH LOG control parameters
HCUTS	DPLOT user cuts
DCSET	Drift Chamber SETtings
PCSET	Proportional Chamber SETtings
HIST	HISTogramming parameters
PHOTO	PHOTO flags
DCCFLAGS	Drift Chamber Calibration FLAGS
SCSET	SCintillator SETtings
SCCUTS	SCintillator CUTS
RFCUTS	RF CUTS
DCCUTS	Drift Chamber CUTS
PCCUTS	Proportional Chamber CUTS
KPUNIT	MOFIA print units
SCFLAGS	SCintillator FLAGS
KFLAGS	MOFIA execution control flags
PRCNTL	Print control flags
GLOBAL	GLOBAL settings
QOD	QOD Monitor params
STRSET	STR SETtings
KalmanCuts	Kalman Tracking Cuts
Alignment	Alignment parameters
FirstGuess	First Guess parameters
HelixFit	HelixFit parameters
TimeZero	Time zero fit settings
Efficiency	Efficiency settings
REZS	REZolutionS controls

MOFIA> **show name DCSET**

NameList **DCSET**: Drift Chamber SETtings

FirstPlaneDC = 1 (dflt = 1): First DC plane

LastPlaneDC = 44 (dflt = 44): Last DC plane

MOFIA> **show name PCSET**

NameList **PCSET**: Proportional Chamber SETtings

FirstPlanePC = 1 (dflt = 1): First PC plane

LastPlanePC = 12 (dflt = 12): Last PC plane

MOFIA> **show name hist**

NameList **HIST**: HISTogramming parameters

nEVTprocessed = -1 (dflt = -1,OFF): Write hists every nEVTprocessed events

FillRawHist = T (dflt = T) : Fill raw histograms

FillHist = T (dflt = T) : Fill histograms after initial filtering

FillTrackHist = T (dflt = T) : Fill tracking histograms

FillPatternHist = T (dflt = T) : Fill pattern recognition histograms

PulserHistToggle = T (dflt = F) : Fill histograms with random pulser data(T) or normal triggers(F)

FillPhysicsHist = T (dflt = T) : Fill physics histograms

FillFirstGuessHist = F (dflt = T) : Fill helix first guess histograms

FillXtalkHist = T (dflt = T) : Fill cross talk histograms

PlaneHists = T (dflt = T) : Generate individual plane histograms

Globalmem_toggle = T (dflt = T) : Turn Global Memory Section on (T) /off (F)

TDC_MIN = 0. (dflt = 0.) : Lower limit on TDC spectra

TDC_MAX = 6000. (dflt = 6000.) : Upper limit on TDC spectra

Raw_XMI_tdcslot = 0. (dflt = 0.) : Lower limit on RAW TDC plots

Raw_XMA_tdcslot = 30000. (dflt = 30000.) : Upper limit on RAW TDC plots

Raw_NX_tdcslot = 3 (dflt = 3) : Number of RAW x Channels for TDC plots

WEvent_per_plane = F (dflt = F) : Turns on/off #of wires hit/event/plane hists

HMult_times = F (dflt = F) : Turns on/off Time difference between multiple hits on a wire hists

MOFIA> **show name dccflags**

NameList **DCCFLAGS**: Drift Chamber Calibration FLAGS

FindPlanePos = F (dflt = F) : Find plane positions

FindWirePos = F (dflt = F) : Find wire positions

FindPlaneRot = F (dflt = F) : Find plane rotations

FindTDC0 = F (dflt = F) : Find TDC time zero

FindResolution = F (dflt = F) : Find drift chamber resolution

MOFIA> **show name dccuts**

NameList **DCCUTS**: Drift Chamber CUTS

DC_MAXTDC_CUT = 20000.00 (dflt = -1, OFF): Max TDC channel cut

DC_MINTDC_CUT = ******* (dflt = -1, OFF): Min TDC channel cut

DC_MAXWTDC_CUT = 150.00 (dflt = -1, OFF): Max TDC width cut

DC_MINWTDC_CUT = -1.00 (dflt = -1, OFF): Min TDC width cut

DC_MAX_HITS_IN_PLANE = -1 (dflt = -1, OFF): Max hit wires in plane cut

DC_MIN_PLANES = -1 (dflt = -1, OFF): Min planes cut

DC_XTALK_WCUT = -1.00 (dflt = -1, OFF): Min cross talk TDC width cut

DC_NOISE_WCUT = -1.00 (dflt = -1, OFF): Min noise TDC width cut

MOFIA> **show name pccuts**

NameList **PCCUTS**: Proportional Chamber CUTS

PC_MAXTDC_CUT = 6000.00 (dflt = -1, OFF): Max TDC channel cut

PC_MINTDC_CUT = 0.00 (dflt = -1, OFF): Min TDC channel cut

PC_MAXWTDC_CUT = 150.00 (dflt = -1, OFF): Max TDC width cut

PC_MINWTDC_CUT = -1.00 (dflt = -1, OFF): Min TDC width cut

PC_MAX_HITS_IN_PLANE = -1 (dflt = -1, OFF): Max hit wires in plane cut

PC_MIN_PLANES = -1 (dflt = -1, OFF): Min planes cut

PC_XTALK_WCUT = -1.00 (dflt = -1, OFF): Min cross talk TDC width cut

PC_NOISE_WCUT = -1.00 (dflt = -1, OFF): Min noise TDC width cut

MOFIA> **show name global**

BField (def = 0.) 0.000000

UnpackMC = F

MOFIA> **show name alignment**

NameList **Alignment**: Alignment parameters

nEventsPlane (def = 10000) = 10000 Total number of events per iteration for calculating plane positions

nEventsWire(def = 100000) = 100000 Total number of events per iteration for calculating wire positions

AlignAngleU (def = 0.0) = 0.000 Angle of the U planes transverse-alignment line with respect to the beam

AlignAngleV (def = 0.0) = 0.000 Angle of the V planes transverse-alignment line with respect to the beam

FixPlanes (def = FALSE) = F Choose a line for transverse alignment defined by any 2 U and 2 V planes

FixedPlane1 (def = 1) = 1 Plane number for one of the four fixed planes used to define the alignment line

FixedPlane1 (def = 1) = 2 Plane number for one of the four fixed planes used to define the alignment line

FixedPlane1 (def = 1) = 7 Plane number for one of the four fixed planes used to define the alignment line

FixedPlane1 (def = 1) = 8 Plane number for one of the four fixed planes used to define the alignment line

MOFIA> **show name efficiency**

Namelist **Efficiency**: Efficiency calculation parameters

RadiusCutDense = 5.000 (dflt = 15 cm): DC dense stack radius cut

RadiusCutSparse = 15.000 (dflt = 15 cm): DC sparse stack radius cut

CellCut = 2 (dflt = 2): Max number of adjacent cells to investigate on each side of the cell expected to have a hit.

CellCut = 10 (dflt = 10): Minimum number of hit planes

FindEff = F (dflt = .FALSE.): Calculate chamber efficiency

MOFIA> **show name timezero**

NameList **TimeZero**: T0 fit settings

T0_TDC_MIN (def = 3000.) = 3000.0 Min range for TDC spectra rising time

T0_TDC_MAX (def = 3080.) = 3080.0 Max range for TDC spectra rising time

TDCnsPerBin (def = 0.5) = 0.5 Number of bins per channel

FitT0 (def = FALSE) = F Accumulate histograms for fitting T0 spectra

TimeBackwards (def = TRUE) = T Time increases backwards

TriggerTimeDC = 0.0

TriggerTimePC = 0.0

TriggerTimeSC = 0.0

TCAP Cut Low Time = -1.0

TCAP Cut High Time = -1.0

MOFIA> **show name rezs**

Namelist **REZS**: Resolutions

nEventsMax = 5000 (dflt = 5000): # Events per Resolution Iteration

MinHistFitEntries = 1000 (dflt = 1000): # histogram entries necessary to do fit

nResidualBins = 300 (dflt = 300): # Bins in residual histograms.

MinResidualValue = 0.30 (dflt = -0.3): Min histogram channel (cm).

MaxResidualValue = 0.00 (dflt = 0.3): Max histogram channel (cm).

PLEASE do NOT mess with the histogram definitions without THINKING about the consequences. See resolution_mod

for details

calcResidualMethod = *** (dflt = 3): Residual calculation method.

1=> Gaussian Fit to residual histograms.

2=> SquaredSum calculation.

3=> FWHM calculation.

peakTOL =

MOFIA> **show name sccuts**

NameList **SCCUTS**: SCintillator CUTS

S1_MAX_NHITS = -1 (dflt = -1, OFF): Scint 1 max number of hits cut

S2_MAX_NHITS = -1 (dflt = -1, OFF): Scint 2 max number of hits cut

S3_MAX_NHITS = -1 (dflt = -1, OFF): Scint 3 max number of hits cut

S1_WIDTH_CUT = -1.00 (dflt = -1, OFF): Scint 1 peak width cut

S2_WIDTH_CUT = -1.00 (dflt = -1, OFF): Scint 2 peak width cut

S3_WIDTH_CUT = -1.00 (dflt = -1, OFF): Scint 2 peak width cut

S1_MAX_TDC = ******* (dflt = -1, OFF): Scint 3 peak width cut

S2_MAX_TDC = ******* (dflt = -1, OFF): Scint 1 max TDC channel cut

S3_MAX_TDC = 2500.00 (dflt = -1, OFF): Scint 2 max TDC channel cut

S1_MIN_TDC = -1.00 (dflt = -1, OFF): Scint 1 min TDC channel cut

S2_MIN_TDC = -1.00 (dflt = -1, OFF): Scint 2 min TDC channel cut

S3_MIN_TDC = -1.00 (dflt = -1, OFF): Scint 2 min TDC channel cut

S1_MAX_ADC = 5000.00 (dflt = -1, OFF): Scint 1 max ADC channel cut

S2_MAX_ADC = 5000.00 (dflt = -1, OFF): Scint 2 max ADC channel cut

S1_MIN_ADC = -1.00 (dflt = -1, OFF): Scint 1 min ADC channel cut

S2_MIN_ADC = -1.00 (dflt = -1, OFF): Scint 2 min ADC channel cut

MOFIA> **show name rfcuts**

NameList **RFCUTS**: RF CUTS

RF_MIN_TDC = -1.00 (dflt = -1, OFF) : RF min TDC cut

RF_MAX_TDC = -1.00 (dflt = -1, OFF) : RF max TDC cut

MOFIA> **show name strset**

use_cos_increments (def= T) T str_angle_inc (def= 0.05) 5.000000E-02

str_upper_angle_limit (def= 85. degrees) 85.0000

MOFIA> **show name FirstGuess**

NameList **FirstGuess**: First Guess parameters

enableFirstGuess = T (logical: T or F)

enableFirstGuessNtuple = F (logical: T or F)

NameList FirstGuess: Time separation between windows

winPCthreshold = 500.000 (ns)

NameList FirstGuess: DC window start

winDCstart = -100.000 (ns)

NameList FirstGuess: DC window end

winDCend = 1000.000 (ns)

MOFIA> **show name HelixFit**

NameList **HelixFit**: HelixFit parameters

enableHelixFit= T

enableHelixNtuple= F

HelixFitVerbose= 0

HelixFitDPDS= 0.000000 energy loss, (MeV/cm)

HelixFitStartFG, StartMC= T F

HelixFitUpstream, HelixFitDownstream= T T

HelixFitCosTmin, HelixFitCosTmax= 0.0100 1.1000

HelixFitWireRes, HelixFitDriftRes, HelixFitTimeRes= 0.1600 0.0300 50.0

HelixFitCutWC, CutDrift, CutTime, CutTref, CutFit= -1.0000 -1.0000 -1.0000 0.0500 0.0200

HelixFitMaxIter= 20

MOFIA> Show name KalmanCuts

NameList **KalmanCuts**: Kalman tracking cuts

ChiDiffCluster (def = 1.E-02) = 0.10E-01 Chi2 convergence level for cluster iteration

ChiDiffTime (def = 1.E-04) = 0.10E-03 Chi2 convergence level for timing iteration

Chi2sCutCluster (def = 1.E05) = 0.10E+06 Chi2 cut for cluster iteration

Chi2sCutTime (def = 1.E05) = 0.10E+06 Chi2 cut for timing iteration

MaxIterateCluster (def = 50) = 50 Maximum number of iterations for cluster fit

MaxIterateTime (def = 50) = 50 Maximum number of iterations for timing fit

EnableKalman (def = TRUE) = T Execute Kalman filterng code

SwitchLR = (def = TRUE) = T Attempt reducing the value of Chi2 by switching left and right

NoiseExcludeMax (def = 2) =

MOFIA> **show name photo**

NameList **PHOTO**: PHOTO FLAGS

IPIC = 1 (dflt = 1) : Chooses priority PHOTO view

HARD = F (dflt = F) : Set TRUE for automatic PHOTO hardcopy

EDGR = F (dflt = F) : Set TRUE to invoke EDGR picture editor

PRINTER = 2 (dflt = 2) : Printer is one of Printronix, HPLaser HPThinkJet, La100, HPPaintJet

DURATION = 0.0 (dflt=0.0) : duration of automatic photos

DETAIL = 0 (dflt = 0) : number of times to automatically add detail to photo

SHOW_COORD = F (dflt = F) : Set TRUE to display counter coordinates

DRAWFOILS = T (dflt = T) : Set TRUE to display UTC superlayer boundaries

TrackedOnly = F (dflt = F) : IF TRUE skip events with no tracks

MOFIA> **show name qod**

n_QOD_buffers (def = 4) 4 qod bufferlength (def =10000) 10000 short term warning message prob

1.000000E-03 long term warning message prob 1.000000E-04 Ratio of hot wire counts to base wire counts 5.00000 baseline

filename (default)

Gives DC wire occupancies, INCLUDING all multiple hits.Set the variable to the plane number you'd like to see. 0 (Default = 0, no hist)

MOFIA> **show name scflags**

NameList **SCFLAGS**: SCintillator FLAGS

S1_signal = T (dflt = TRUE) : Require a signal in S1

S2_signal = T (dflt = TRUE) : Require a signal in S2

S3_signal = T (dflt = TRUE) : Require a signal in S3

X.2 Failure Codes

X.2.1 Event Filtering Failure Codes

Failure Code	Description	Associated Namelist
1	successful unpacking	
2	at least one DC hit present	
3	there is less than S1_MAX_NHITS in scint 1	SCCUTS
4	there is less than S2_MAX_NHITS in scint 2	SCCUTS
5	require a signal in scint 1	SCCUTS
6	require a signal in scint 2	SCCUTS
7		
8		
9	time from scint 1 is close to its peak location	SCCUTS
10	time from scint 2 is close to its peak location	SCCUTS
11	number of hits in DC plane is less than DC_MAX_HITS_IN_PLANE	DCCUTS
12	require the numbr of DC hits in each plane to be less than DC_MIN_PLANES	DCCUTS
13	require the total number of DC hits to be less than DCmaxAllowedHits	DCCUTS
14	number of hits in PC plane is less than PC_MAX_HITS_IN_PLANE	PCCUTS
15	require the numbr of PC hits in each plane to be less than PC_MIN_PLANES	PCCUTS
16	require the RF time to be within the range [RF_MIN_TDC,RF_MAX_TDC]	RFCUTS
17	require the total number of PC hits to be less than PCmaxAllowedHits	PCCUTS
18	there is less than S3_MAX_NHITS in scint 3	SCCUTS
19	require a signal in scint 3	SCCUTS
20	time from scint 3 is close to its peak location	SCCUTS

Table 7 Event filtering failure codes.

X.2.2 Pattern recognition Failure Codes

Failure Code	Description	Namelist
1	track > MaxTmpTracks in InsertCluster	
2	iPair = PrevPair(iFGTrack) in InsertCluster	
3	FGCl(iFGTrack) % nCl >= MaxPairs in InsertCluster	
4	Cluster does not fit circle in InsertCluster	
5	Less than three clusters in ResolveCircle	
6	ChiSquare unacceptably large in ResolveCircle	
7	nRows in matrices A and B are unequal in MatSolv (Matrix_mod)	
8	Unique solution does not exist for matrix equation in MatSolv (Matrix_mod)	
9	0 row in matrix in Triangulate (Matrix_mod)	
10	Failed to find pivot in Triangulate (Matrix_mod)	
11	ChiSquare unacceptably large in ResolvePhiLambda	
12	ChiSquare unacceptably large in ResolvePhiLambda	

Table 8 Pattern recognition failure codes.

X.2.3 $\chi 2$ Fit Failure codes

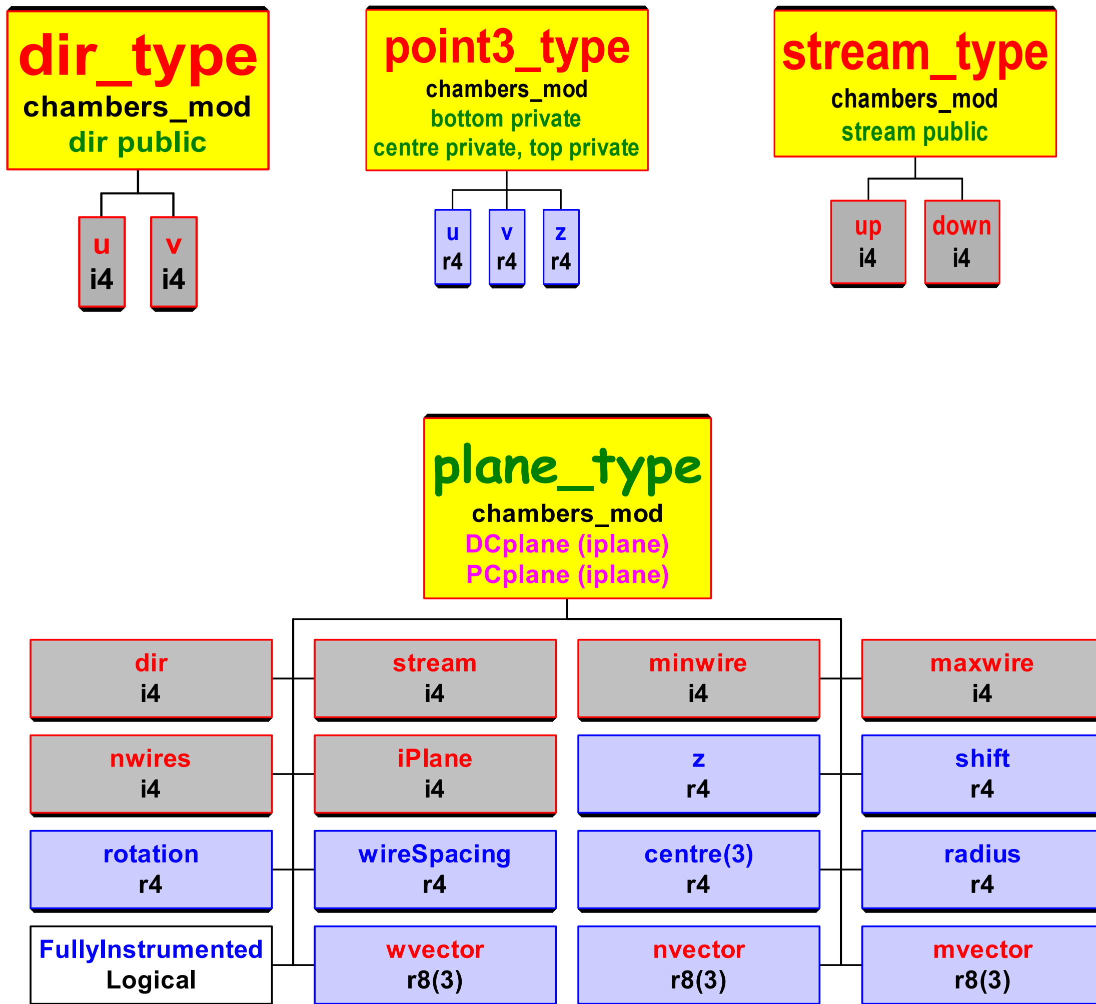
X.2.4 Kalman Filtering Failure Codes

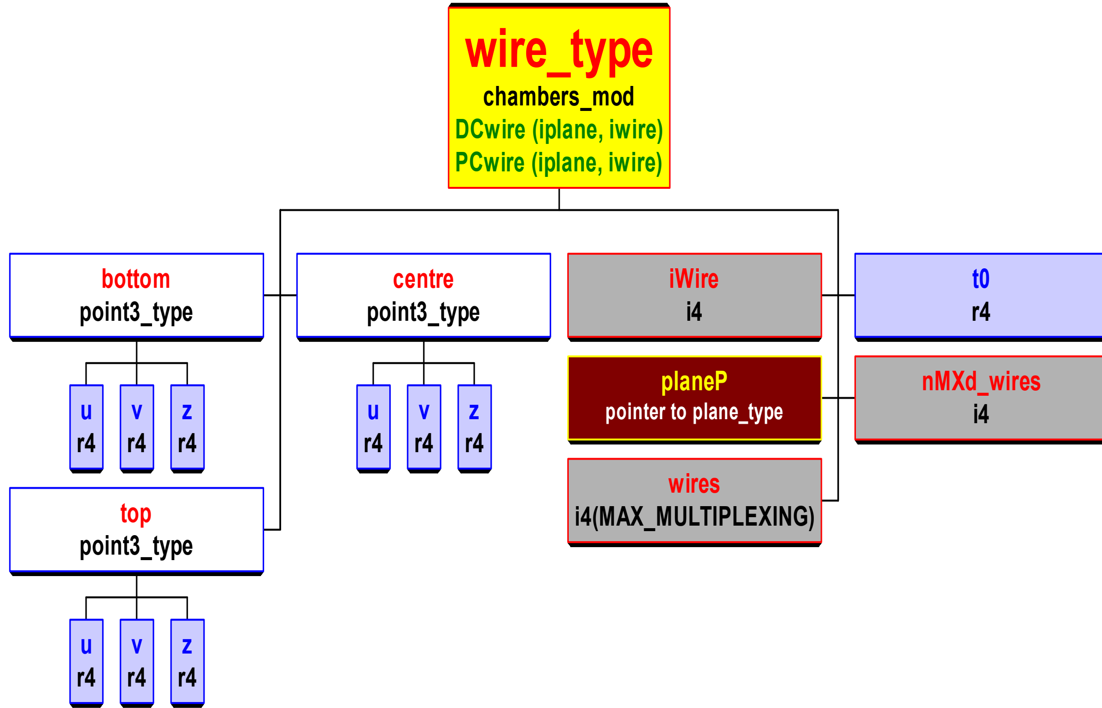
Failure Code	Description	Namelist
1	Not enough hits for tracking	
2	Filter failed in cluster fit	
3	Smoother failed in cluster fit	
4	Bad track Chi2 – cluster iteration	
5	Failed computing hit position	
6	Filter failed in timing fit	
7	Smoother failed in timing fit	
8	Bad tracking Chi2 - timing iteration	
9	Failed computing physics paramters	

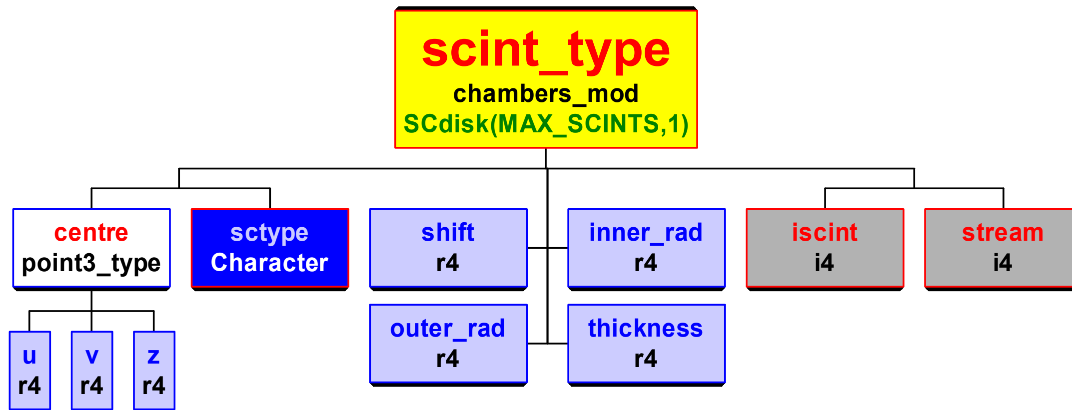
Table 9 Kalman filter failure codes.

X.3 Data Structures

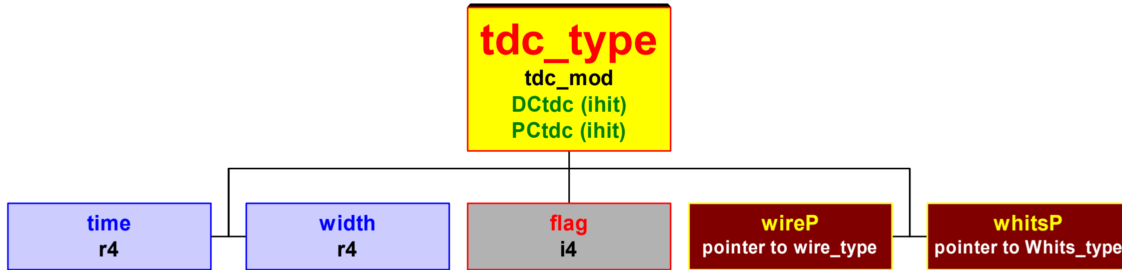
X.3.1 Geometry Structures

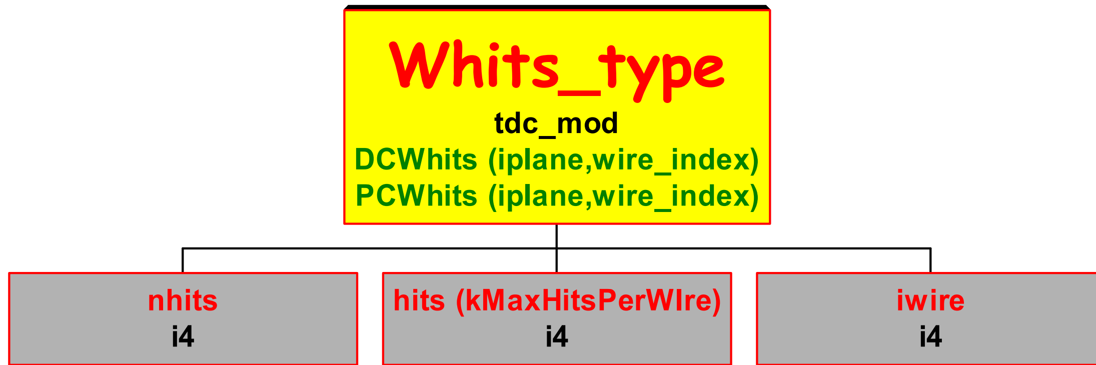


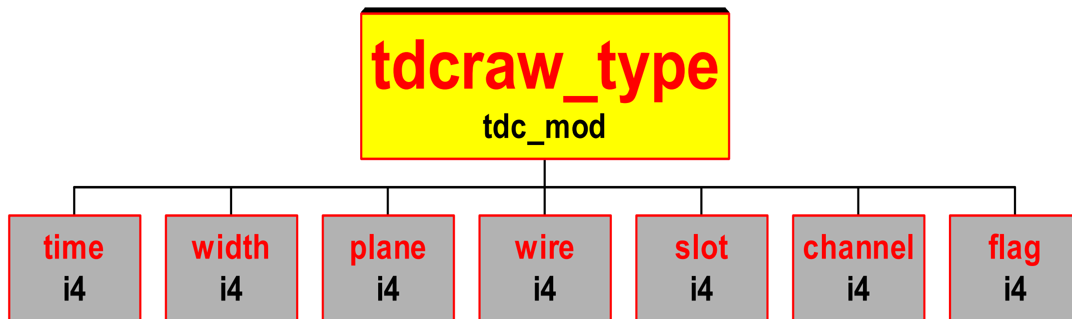

wire_type
chambers_mod
DCwire (iplane, iwire)
PCwire (iplane, iwire)
bottom
point3_type
u
r4
v
r4
z
r4
centre
point3_type
u
r4
v
r4
z
r4
iWire
i4
t0
r4
planeP
pointer to plane_type
nMXd_wires
i4
wires
i4(MAX_MULTIPLEXING)
top
point3_type
u
r4
v
r4
z
r4

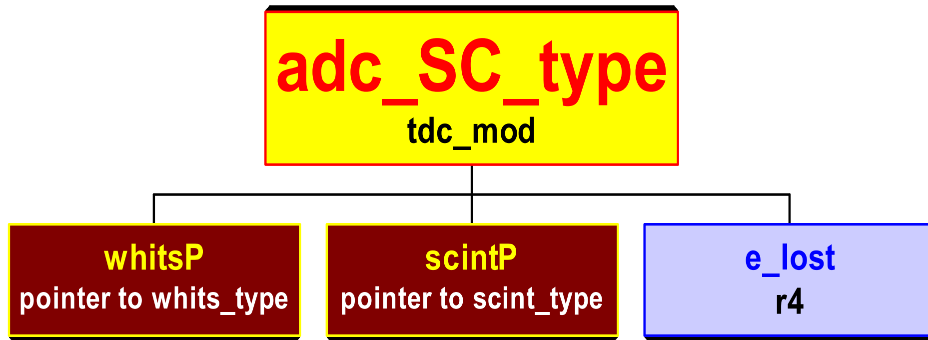

scint_type
chambers_mod
SCdisk(MAX_SCINTS,1)
centre
point3_type
sctype
Character
shift
r4
inner_rad
r4
iscint
i4
stream
i4
u
r4
v
r4
z
r4
outer_rad
r4
thickness
r4

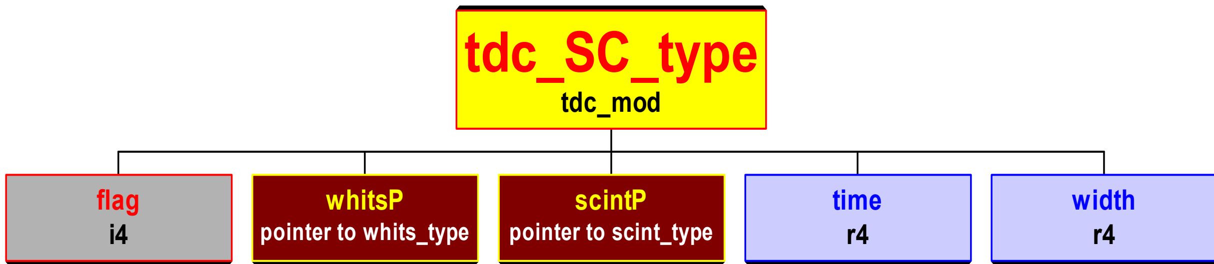
X.3.2 TDC Structures



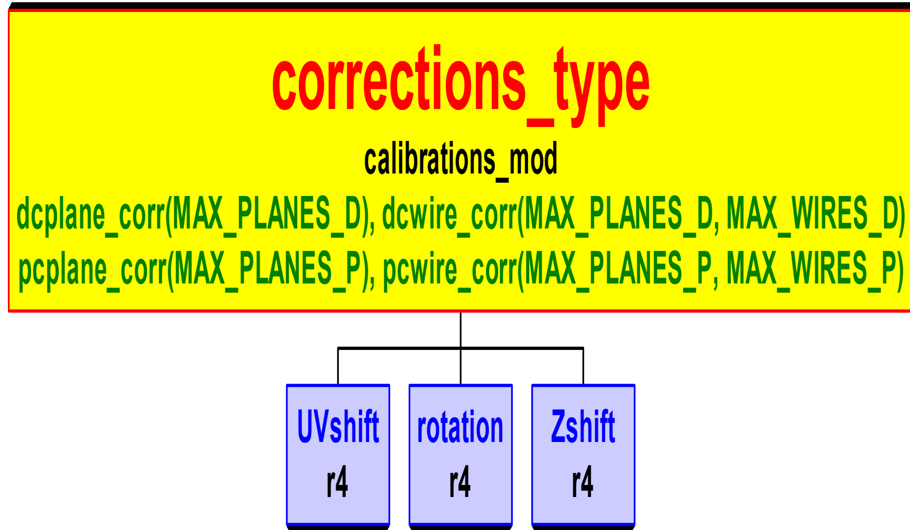


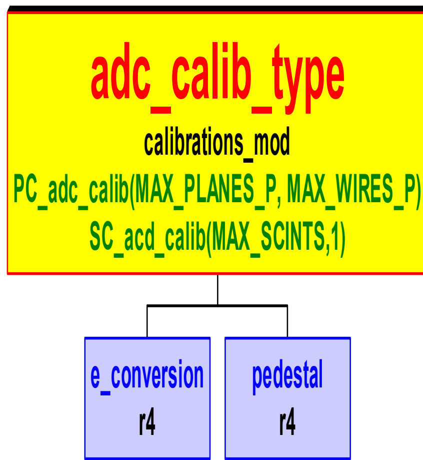


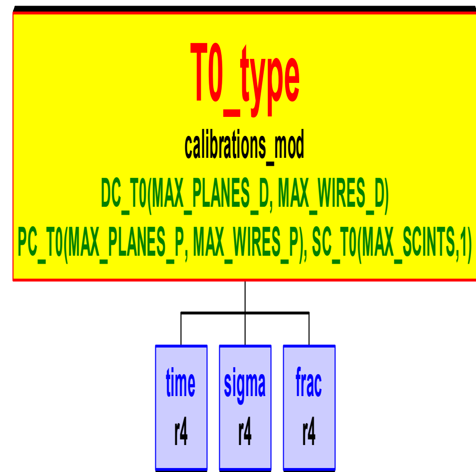

adc_SC_type
tdc_mod
whitsP
pointer to whits_type
scintP
pointer to scint_type
e_lost
r4


tdc_SC_type
tdc_mod
flag
i4
whitsP
pointer to whits_type
scintP
pointer to scint_type
time
r4
width
r4

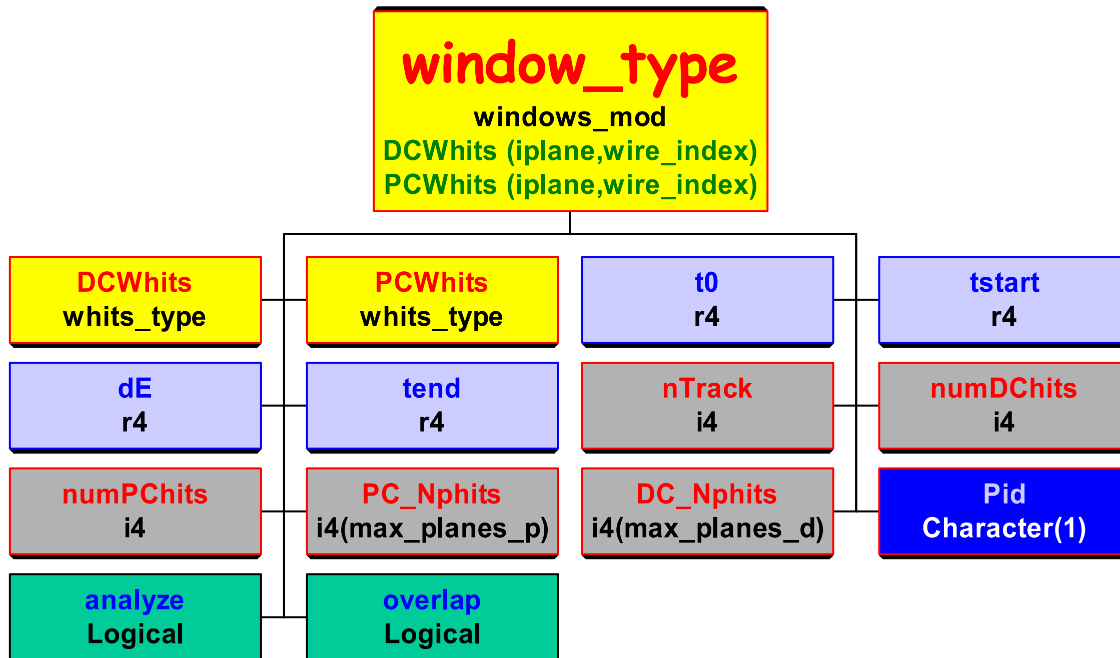
X.3.3 Calibrations Structures



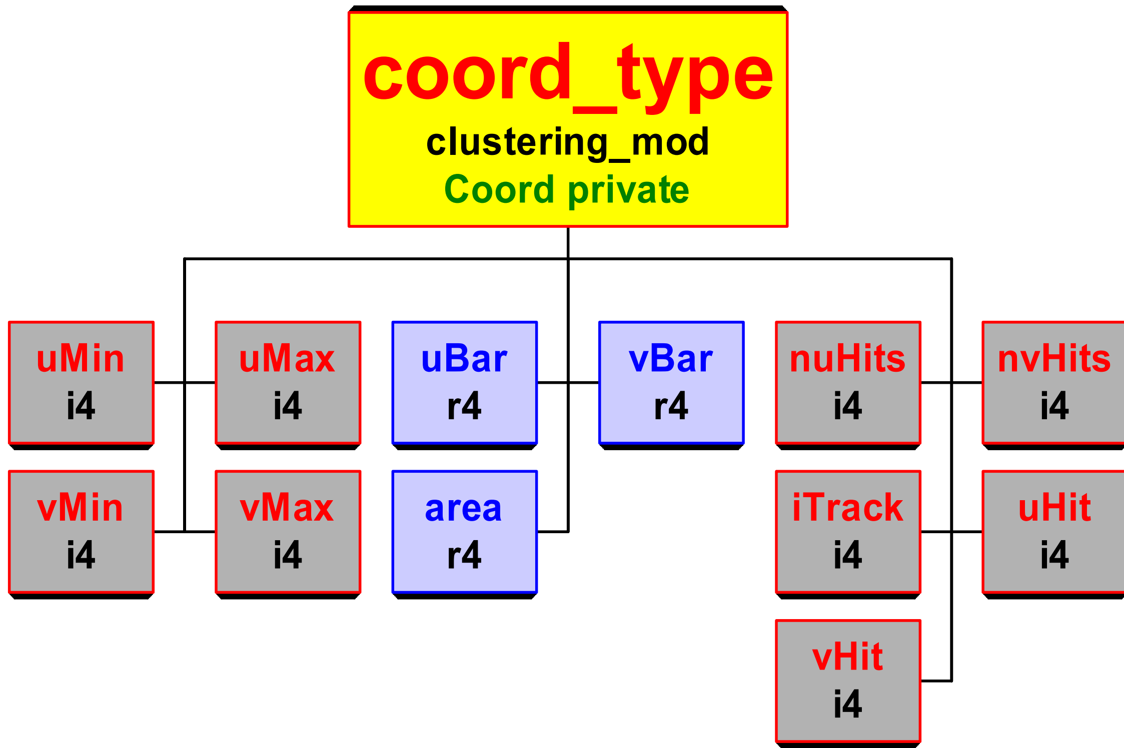


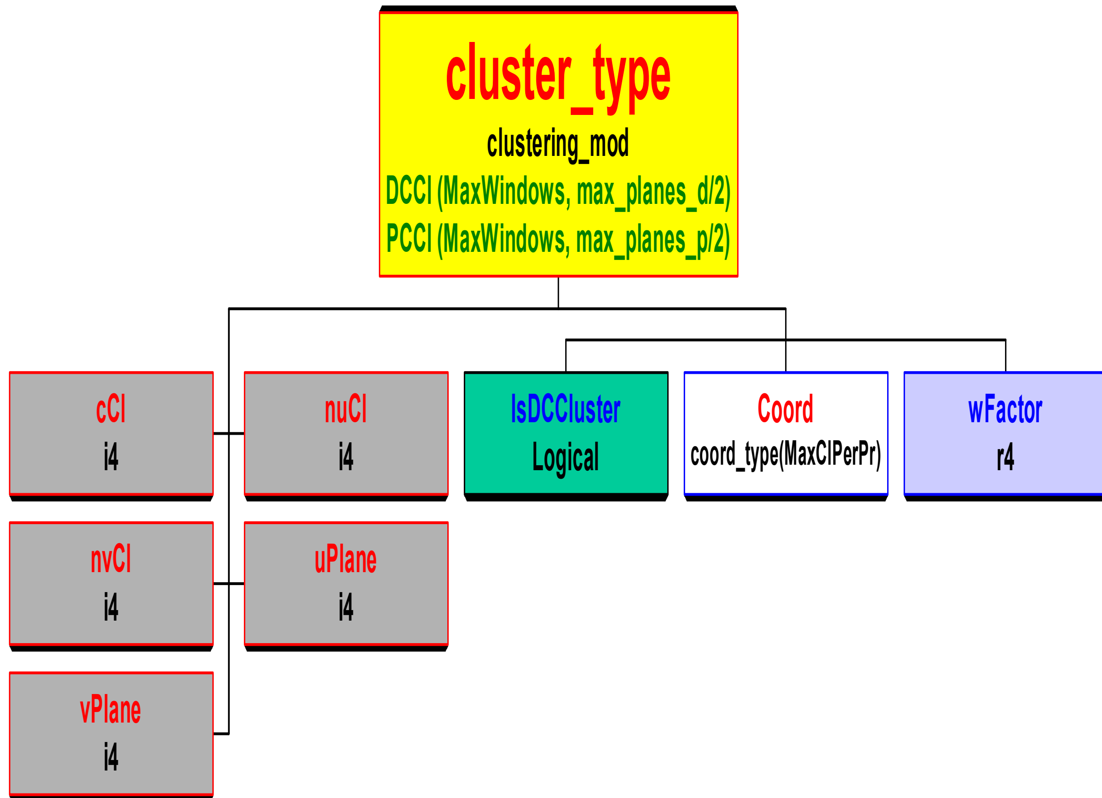


X.3.4 Windowing Structures

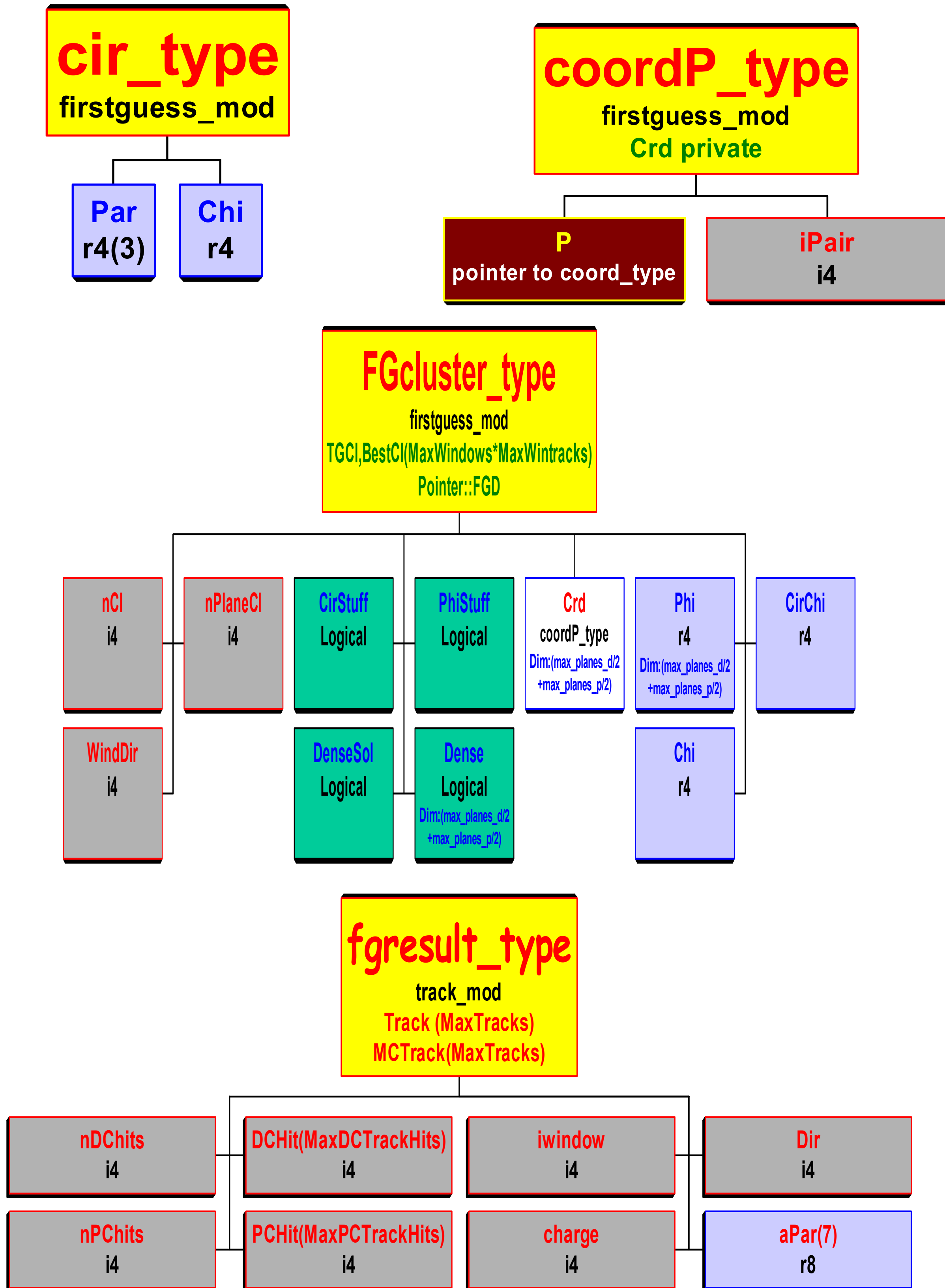


X.3.5 Clustering Structures

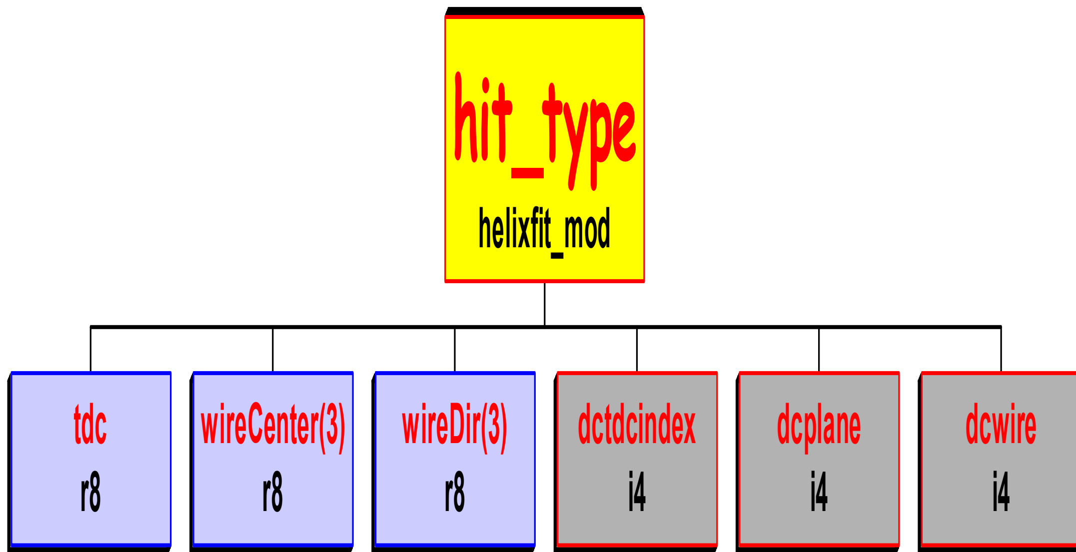


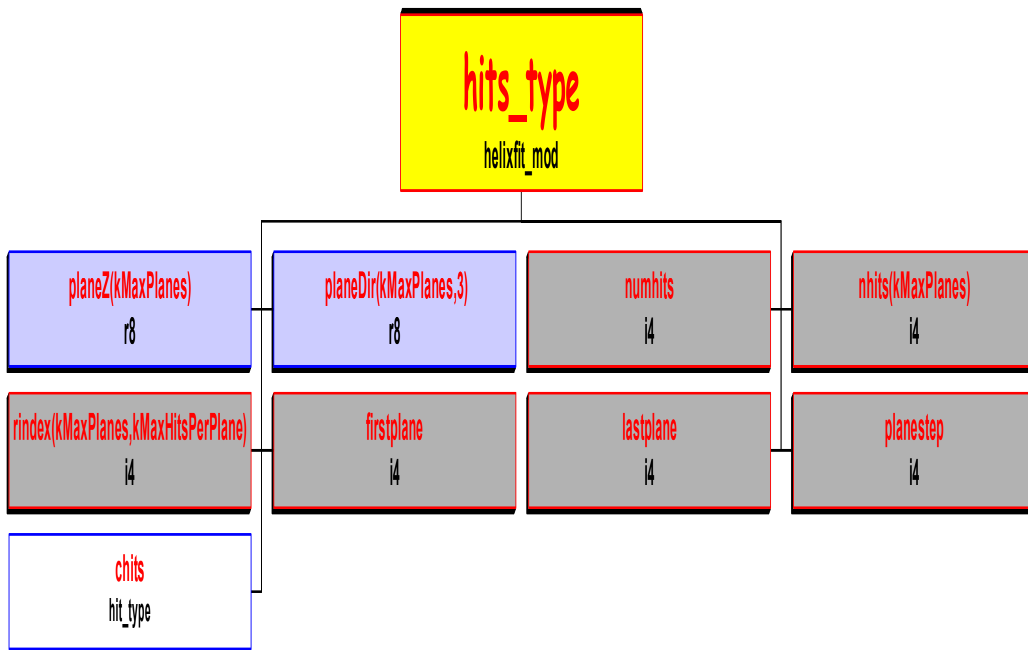


X.3.6 First Guess Structures

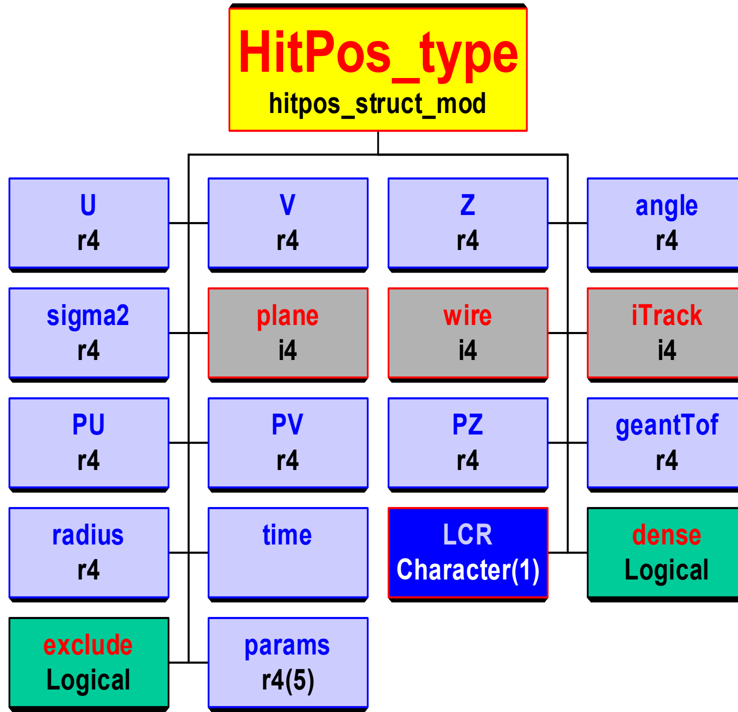


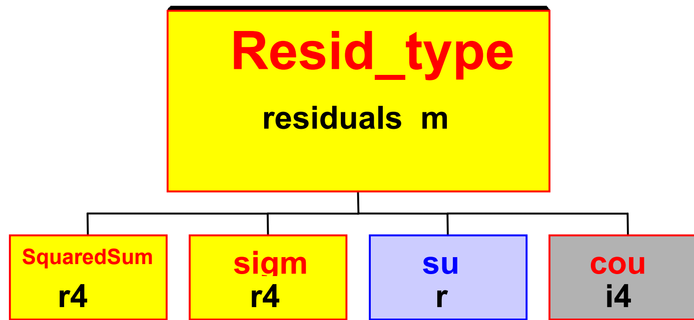
X.3.7 χ^2 Helix Fit Structures

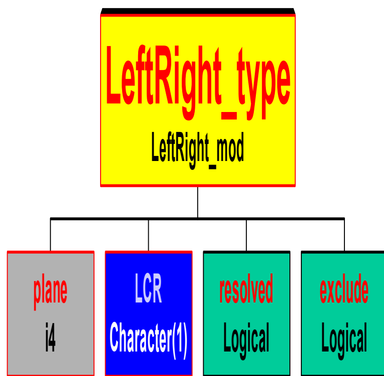


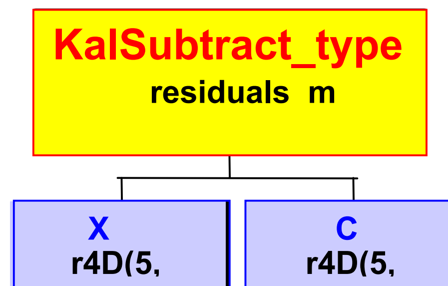


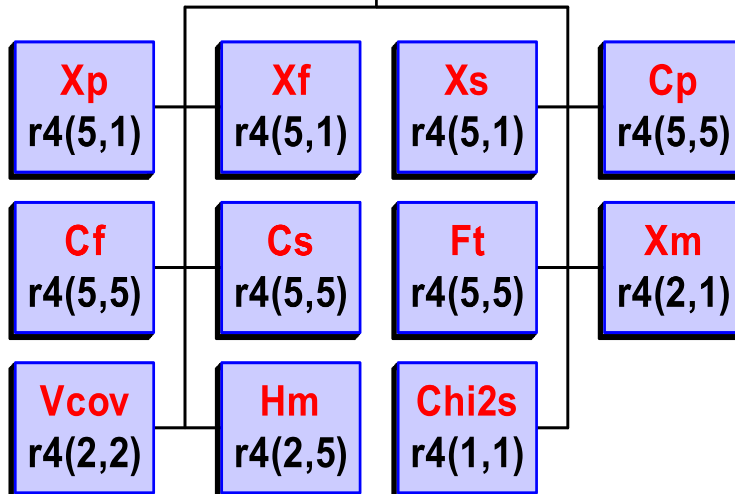
X.3.8 Kalman Filter Structures

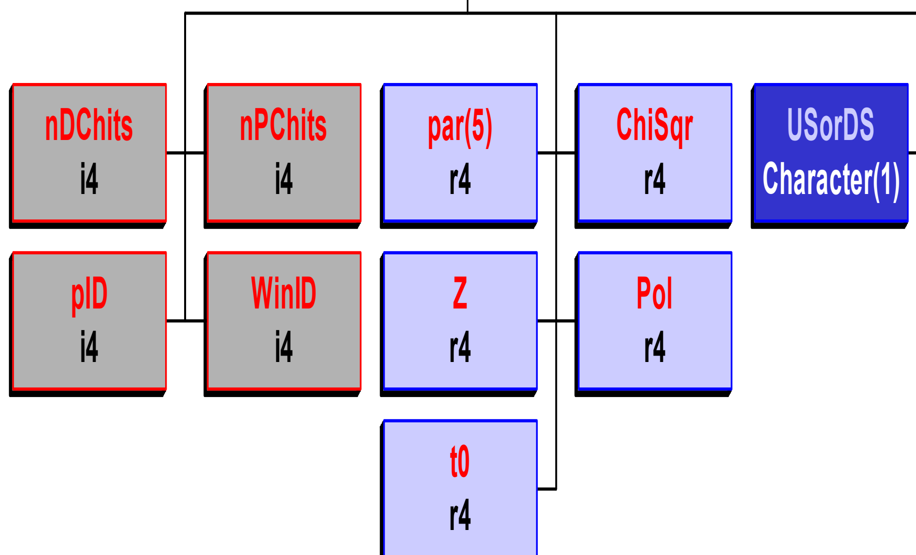




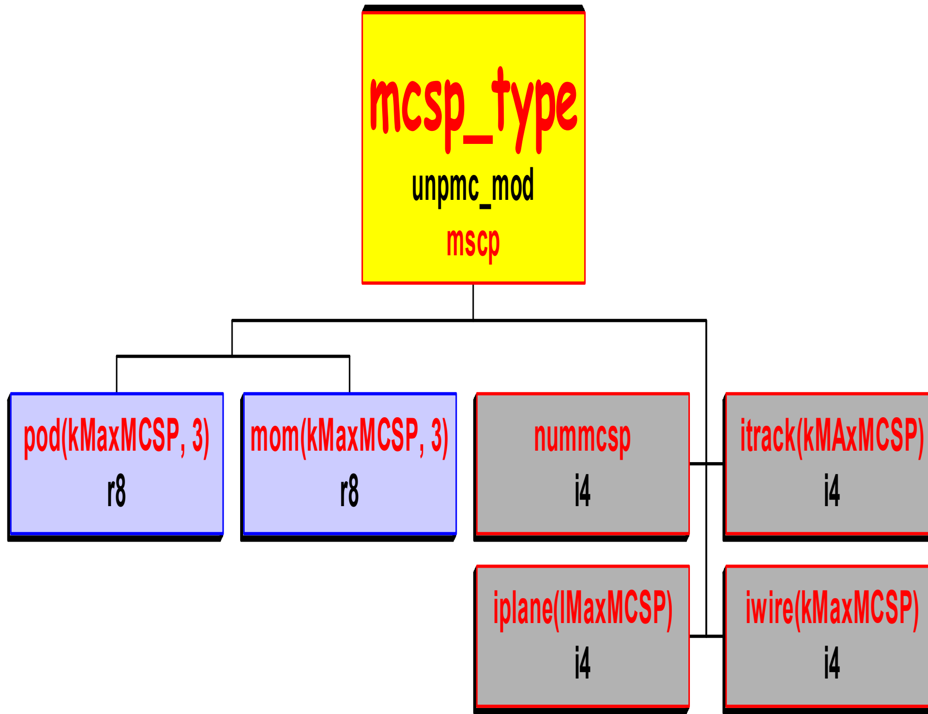


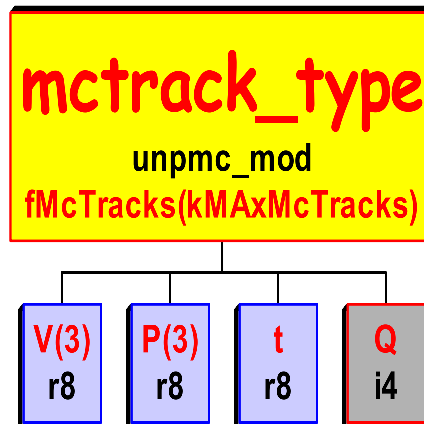



KalHit_type
kalman_mod
Xp
r4(5,1)
Xf
r4(5,1)
Xs
r4(5,1)
Cp
r4(5,5)
Cf
r4(5,5)
Cs
r4(5,5)
Ft
r4(5,5)
Xm
r4(2,1)
Vcov
r4(2,2)
Hm
r4(2,5)
Chi2s
r4(1,1)


track_type
track_mod
Track (MaxTracks)
MCTrack(MaxTracks)
nDChits
i4
nPChits
i4
par(5)
r4
ChiSqr
r4
USorDS
Character(1)
pID
i4
WinID
i4
Z
r4
Pol
r4
t0
r4

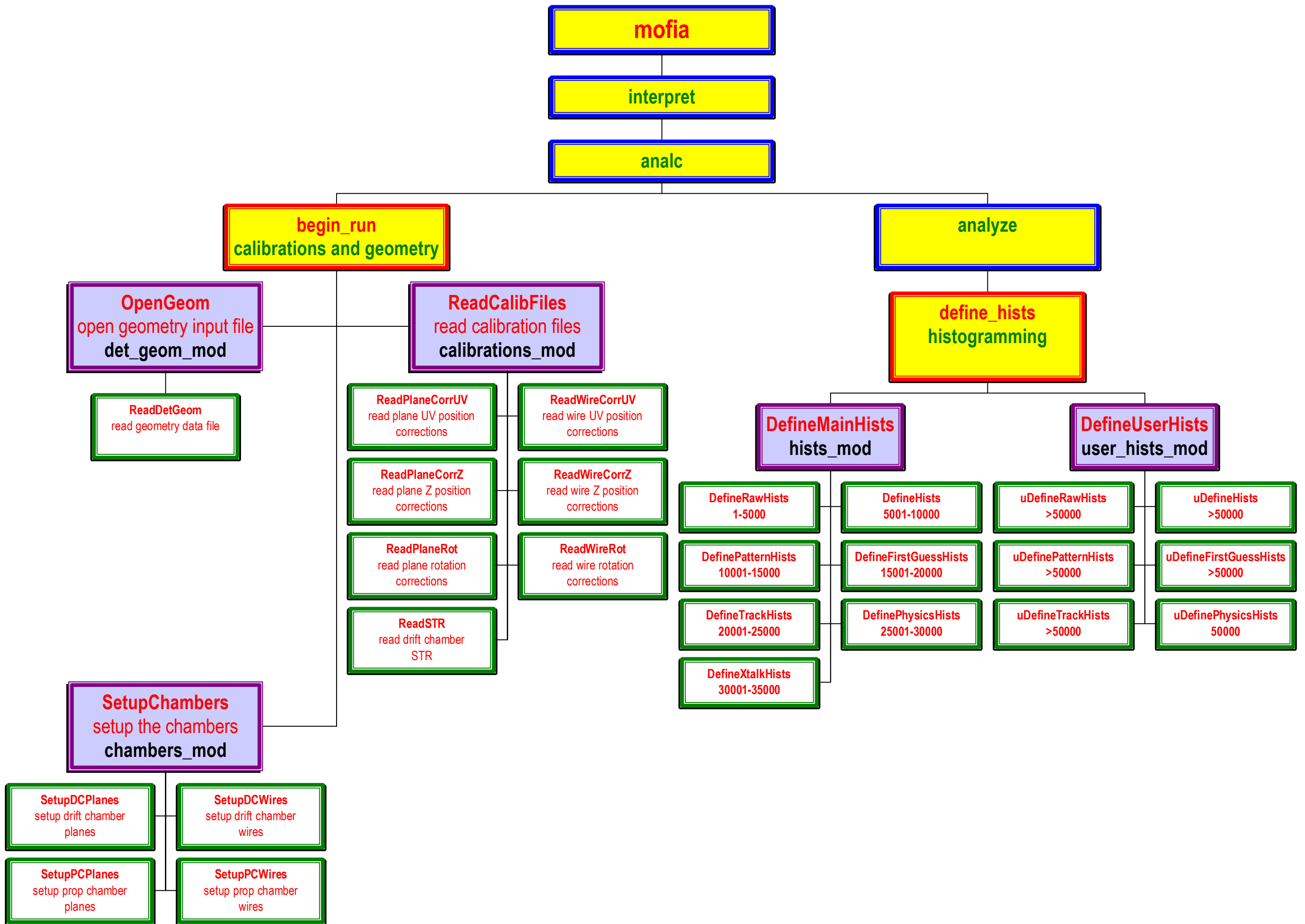
X.3.9 MC Banks Structures





X.4 Flowcharts

X.4.1 Initialization Branch



X.4.2 Analysis Branch