

AMS Note 2005-04-19

AMS Analysis Tool User and Reference Manual

P.Zuccon¹ and D.Caraffini²

¹INFN Sezione di Perugia

²Università degli Studi di Perugia

Contents

User Manual	3
1 AMS Analysis Tool	3
1.1 Introduction	3
1.2 The Histogram Manager	3
1.2.1 A more advanced use of DHF	5
1.2.2 Booking an Filling histograms by category	7
1.3 The Cut Manager	8
1.3.1 Cut Library	8
1.3.2 Evaluating the Cuts	9
1.3.3 The filtering option	10
1.3.4 Selecting the cuts: the datacard	10
1.4 The Analysis tool	10
1.4.1 The user point of view	12
1.4.2 Notes on the particle selection for the analysis	13
1.5 Conclusions and acknowledgements	15
Appendix A: quick start guide	17
Reference Manual	21
2 AMS Analysis Tool Hierarchical Index	21
2.1 AMS Analysis Tool Class Hierarchy	21
3 AMS Analysis Tool Class Index	23
3.1 AMS Analysis Tool Class List	23
4 AMS Analysis Tool Class Documentation	25
4.1 AMSAnaTool Class Reference	25
4.1.1 Detailed Description	26

4.1.2	Constructor & Destructor Documentation	27
4.1.3	Member Function Documentation	27
4.1.4	Member Data Documentation	28
4.2	Categories Class Reference	30
4.2.1	Detailed Description	31
4.2.2	Constructor & Destructor Documentation	32
4.2.3	Member Function Documentation	32
4.2.4	Member Data Documentation	33
4.3	CList Struct Reference	34
4.3.1	Detailed Description	34
4.3.2	Member Data Documentation	34
4.4	DCut Class Reference	36
4.4.1	Detailed Description	38
4.4.2	Constructor & Destructor Documentation	39
4.4.3	Member Function Documentation	39
4.4.4	Member Data Documentation	41
4.5	DHF Class Reference	42
4.5.1	Detailed Description	44
4.5.2	Constructor & Destructor Documentation	44
4.5.3	Member Function Documentation	44
4.5.4	Member Data Documentation	46
4.6	DHFU Class Reference	47
4.6.1	Detailed Description	47
4.6.2	Constructor & Destructor Documentation	47
4.6.3	Member Function Documentation	47
4.7	OneCut Class Reference	49
4.7.1	Detailed Description	50
4.7.2	Member Typedef Documentation	50
4.7.3	Constructor & Destructor Documentation	50
4.7.4	Member Function Documentation	51
4.7.5	Member Data Documentation	51

User Manual

Chapter 1

AMS Analysis Tool

1.1 Introduction

The AMS data are presented in ROOT format as collections of C++ objects, the most direct way to analyse them, is then to use the C++ language and the ROOT framework. In Perugia we developed a set of C++ classes that greatly simplify both the task to book and fill a required set of histograms at different steps of the analysis, and those to handle the selection criteria and their features (order and parameters).

In this note, we illustrate the tools we developed to help the analysis task. We start describing the Histogram Manager, then we pass to the Cut Manager, finally we show how these tools have been integrated to realise the Analysis Tool.

1.2 The Histogram Manager

Within the ROOT framework histograms are implemented as C++ objects and the standard way to use an histogram is to book it with a command like:

```
TH1F* h1= new TH1F("histo1","Test histogram No. 1",10,0.,10.);
```

the pointer `h1` to the histogram object must be used for all the subsequent operations like filling, cloning and saving.

The problem of the management of the histogram pointers is tedious and introduces a heavy overhead in writing the analysis code, the histogram manager cures this problem allowing to refer to an histogram by its name and introducing a large simplification of the analysis code.

The basic idea is to group a set of histograms in special container classes, as **TObjArray**, and exploit the capability of these containers to find the objects by their name. This is possible because the ROOT histogram classes are not only **TObject** objects¹ but also **TNamed** objects since they have this class in their inheritance tree.

In our histogram manager the histogram pointers are held in a **TObjArray** object, a vector like container of **TObject** objects, this allows to exploit the capability of the **TObjArray** container to find an element by its name. This feature is implemented by the method `TObjArray::FindObject(char * name);`.

¹Almost all ROOT classes inherit from the class `TObject`.

To keep ordered structures of histograms and to provide also a standard method to save them, the histogram contains also a pointer to a **TDirectory** object. The **TDirectory** class implements a tree-like structure where the **TObjects**, and hence the histograms, can be appended. A structure of this type allows to store the histograms in an ordered way, and it is navigable with the standard ROOT tools (ex. **TBrowser**).

Furthermore, since the ROOT file class **TFile** inherits from **TDirectory**, this provides an easy way to save the histograms either by "appending" a **TDirectory** to the **TFile** class, either using directly a **TFile** object instead of a **TDirectory** one.

Summarising two collectors are used at the same time to hold the histogram pointers: the **TObjArray** which is a flat structure that allows for an easy and fast search of the histograms, and the **TDirectory**, which has a tree structure, allows to keep an ordered view of the histograms and provides a standard method to save the histograms.

Our implementation of the histogram manager is represented by the class **DHF**, this class contains a **TObjArray** object to hold the histogram pointers and a pointer to a **TDirectory** object to which histograms and subdirectories are appended. It provides a method **DHF::Add()** that allows to add new histograms to the manager and a method **DHF::Fill()** that allows to fill the histogram. A complete description of the **DHF** class can be found in the reference manual. The following code shows an example of a basic use of the **DHF** class².

```
#include "DHF.h"
#include "TMath.h"

int main(){
  TFile f("Output.root","recreate");

  DHF* Histos= new DHF(&f);

  Histos->Add(new TH1F("histo1","Test histogram No. 1",10,0.,10.));

  for (int ii=0;ii<100;ii++){
    float tt=TMath::Rndm();
    Histos->Fill("histo1",tt);
  }
  f.Write();
  f.Close();
}
```

From the user point of view the operations on the histogram are reduced to the utilisation of:

```
Histos->Add( new TH1F("histo1","Test histogram No. 1",10,0.,10.));
```

to book the histogram, and:

```
Histos->Fill( "histo1",tt);
```

to fill it.

²A working example is provided in the distributed package (see Appendix A) under the directory `HManager.examples`.

1.2.1 A more advanced use of DHF

In a typical analysis, it is useful to monitor several quantities under different conditions, i.e. to have the same set of histograms filled up under different requests. The class **DHF** provides two methods `DHF::BookHistos()` and `DHF::FillAll()`, which can be used to book and fill a bunch of histograms in one go. These methods (defined in the file `DHF.cxx`) are void as default and can be customised by the user.

Since the pointers to the AMS event and particle are held as class members in the **DHF** class, they are available for the use in the `DHF::FillAll()` method, provided that at each event they are set via a call to the `DHF::SetCurrentPart()`.

However the suggested method is to not modify the original **DHF** class, but to exploit the C++ features and create a new class **DHFU** (user's **DHF**) which inherits from **DHF** and contains only the customised versions of `BookHistos()` and `FillAll()`. The definition of the **DHFU** class (`DHFU.h`) should look like³:

```
#ifndef DHFU_h
#define DHFU_h
#include "DHF.h"

class DHFU :public DHF {

public:
    DHFU(TDirectory* Dir=0,char *name="HManager",char *title="default HManager")
        :DHF(Dir,name,title){}
    void FillAll();
protected:
    void BookHistos();
    ClassDef(DHFU,0)
};

#endif
```

while the implementation part (`DHFU.cxx`) should look like:

```
#define DHFU_cxx
#include "DHFU.h"

ClassImp(DHFU)

void DHFU::BookHistos(){
    // User routine to book the histos

    fDir->cd();

    // //example histos
    //fHlist->Add some histos to the top directory level of this DHF Object
    fHlist->Add (new TH2F("RigVsMCMom","RigVsMCMon;MCMomentum;Rigidity",100,9.,210.,100,-11.,11.));
}
```

³A working example is provided in the distributed package (see Appendix A) under the directory `HManager.examples`.

```
void DHFU::FillAll(){
    // User routine to fill the histos

    Fill("RigidityVsMCMomentum",Partcl->Momentum,Event->MCEventg(0).Momentum);
}

```

The typical use of the **DHFU** class becomes:

```
#include "DHFU.h"

DHFU* Histos1=0; DHFU* Histos2=0;

int main(){
    TFile *f= new TFile("Output.root","recreate");

    f->cd();
    TDirectory * D_histos1=(TDirectory *) f->mkdir("HistosA");

    TDirectory * D_histos2=(TDirectory *) f->mkdir("HistosB");

    Histos1= new DHFU(D_histos1,"CaseA", "TestCase A");
    Histos2= new DHFU(D_histos2,"CaseB", "TestCase B");

    Histos1->Init(); //internally calls BookHistos
    Histos2->Init(); //internally calls BookHistos

    // some loop on the AMS events that calls
        processevent(AMSEventR * ev, ParticleR * part)
    //end of the loop

    f->Write();
    f->Close();
}

int processevent(AMSEventR * ev, ParticleR * part)

    Histos1->SetCurrentPart(ev,part);
    Histos2->SetCurrentPart(ev,part);

    if(ev->pMCEventg(0)->Momentum>10.) Histos1->FillAll();
    else Histos2->FillAll();

    return 0;
}

```

The class **DHF(U)** inherits from the class **TNamed** so the **DHF(U)** objects can have a name and a title; as shown in the example these attributes can be set using the full constructor. The most interesting consequence is that also the **DHF(U)** objects can be put in a collector (ex. **TObjArray**) and searched by their name.

1.2.2 Booking an Filling histograms by category

In some analyses it can be interesting to divide the data sample in different categories to separately study the same quantities.

In order to address this issue, the **DCut** class provides the method:

```
DHF::Define("histo_name","histo title",10,0.,10.);
```

as a means to book multiple copies of the same histogram with names of the form `histo_nameCategory_name`, thus mapping each one to a different event category. The defined histograms are appended to a directory with name `histo_name_D` which is created automatically by the command itself under the main directory of the **DHF**.

To fill such histograms it is enough to give the command:

```
DHF::FillSet("histo_name",value);
```

and the **DHF** object will automatically fill the histogram that corresponds to the category of the current event in the set of those with name starting with `histo_name`.

The implementation of this feature relies on the services of the static class **Categories** and requires to write some code before the creation of **DHF** objects.

The code must mimic the following example⁴:

```
#include "DHFU.h"

int GoodBadNeg(AMSEventR* ev,ParticleR* part);

int main(){

    Categories::SetNCat(3);
    Categories::SetCatName(0,"Good");
    Categories::SetCatName(1,"Bad");
    Categories::SetCatName(2,"Neg");
    Categories::SetFun(GoodBadNeg);

    TFile *f= new TFile("Output.root","recreate");

    f->cd();

    TDirectory * D_histos1=(TDirectory *) f->mkdir("HistosA");

    TDirectory * D_histos2=(TDirectory *) f->mkdir("HistosB");

    DHFU* Histos1= new DHFU(D_histos1,"CaseA", "TestCase A");
    ...
}

int GoodBadNeg(AMSEventR* ev,ParticleR* part){
```

⁴A working example is provided in the distributed package (see Appendix A) under the directory `HManager.examples`.

```

//Selects three categories
// good events with Delta P/P <0.12
// bad events with Delta P/P >0.12 but with the right sign assignment
// neg events with the wrong sign assignment

MCEventgR* Gen=ev->pMCEventg(0);
double res=fabs (1. - ((part->Momentum) / Gen->Momentum));
if (part->Momentum*Gen->Momentum <0) return 2;
else if (res < 0.12) return 0;
else return 1;
}

```

The user is requested to provide the **Categories** with the number and names of the planned categories that using the methods `Categories::SetNCat()` and `Categories::SetCatName()` respectively. It is also the user's responsibility to write a function returning the correct category number for each event; this function must have a prototype like:

```
int func(AMSEventR * ev, ParticleR * part);
```

The address of this function is passed to the class via the `Categories::SetFun()` command.

It is also possible to choose the histogram line colour associated to each category using the command:

```
Categories::SetCatColor(category_index,color_index);
```

1.3 The Cut Manager

The Cut Manager covers many tasks: it holds a library of the available cuts, it reads a datacard containing the cut configuration (order and parameters), it evaluates the cuts for each event, it does the bookkeeping of the events passing the single cuts and the whole cut chain.

The Cut Manager is implemented via the class **DCut**, in the following we discuss the most important features of this class. For a technical description of the implementation the reader is addressed to the reference manual and to ... the code itself.

1.3.1 Cut Library

The selection criteria, or cuts, can be represented as functions that taking as input the data and some parameters, return a boolean result like passed/not-passed.

We decided to standardise the interface for these functions and to "embed" it in a **TNamed** object. This carries some relevant advantages:

- The standard interface allows for a great simplification of the code and introduces the possibility to easily share the cut with other people,
- The "embedding" in a **TNamed** object allows to collect the cuts into a container class (like **TObjArray**) implementing a cut library,
- The "embedding" allows to refer to the cuts by their name (this is exploited in our datacard implementation).

Following our standardised interface a cut must be defined as a function of the form:

```
int <function_name>(AMSEvent *ev, Particle*part,float*parameters);
```

where the first two arguments are pointers to the AMS data and the third one represents the array of the cut parameters.

The "embedding" is realised via the introduction of the class **OneCut**. This class inherits from **TNamed** and contains: the pointer to the function actually implementing the cut, an array for the cut parameters and some useful information such as the number of the parameters. It also provides methods to evaluate the cuts and access the information, a detailed description of the class is available in the reference manual.

To add a cut to the cut library the user must edit the file `DCut.cxx` and add a line to the `DCut::Library_init()` method like:

```
_Cuts->Add( new OneCut("name","description",<function_name>,npars));
```

where the parameters for the **OneCut** constructor are in the order: the cut name, the cut description, the pointer to the function actually implementing the cut and the number of parameters used in the cut.

The name given at this stage will be used by the program to refer to this cut when parsing the datacard and in the bookkeeping output.

1.3.2 Evaluating the Cuts

The event by event evaluation of the cuts is the responsibility of the user who has to insert in his event analysis function the following lines:

```
// sets the pointers to the data
mydcut->SetParticle(curpart,event);
//evaluate the cuts
mydcut->EvalList();
```

where `mydcut` is the pointer to the actual **DCut** object, `curpart` and `event` are the pointers to the AMS data (particle and event).

Various methods are provided to know if a cut is passed or not, in particular:

- `int DCut::Passed(int mm)` returns 1 if the `mm`-th cut in the chain is passed as standalone,
- `int DCut::AllPassed(int mm)` returns 1 if all the cuts in the cut chain up to the `mm`-th position are passed,
- `int DCut::AllButThis(int mm)` returns 1 if the all cuts in the cut chain except the `mm`-th one are passed.

The user can decide what actions to take after the evaluation of the cuts, but the number of events passing each cut as standalone, and the number of events passing each step of the cut chain are stored by the cut manager anyway in two internal histograms.

In our implementation the first cut of the cut chain has a particular meaning: it must contain the very basic requests that are needed for the other cuts to be evaluated (such as the existence of an event!). As a consequence internally it is requested that the first cut is passed for the other cuts to be evaluated. If this feature is not needed or can create problems it is enough to put an always-true cut as first cut.

1.3.3 The filtering option

There is the possibility to save on an external file the events passing a cut in the cut chain. To enable this feature the user must pass the pointer to the AMS **TTree** using the method:

```
DCut::FilteringOn(TTree *intree),
```

this pointer is needed in order to create a copy of the AMS data structures within the new file(s).

To properly close the files the user must call the method:

```
DCut::CloseOutFiles(),
```

a method for saving the files during the loop is also provided:

```
DCut::SaveOutFiles().
```

The file name format is `<cut_name>_filtered.root`. The cut level(s) at which the events are saved are set via a flag in the datacard.

1.3.4 Selecting the cuts: the datacard

After the insertion of new cuts in the cut library, the user must choose which cuts he wants to use for a particular analysis, indicating also the cut order and the various cut parameters; this is done via a text datacard. Figure 1.1 shows an example of the datacard.

The data card is in free format and each line correspond to a cut: comment lines can be inserted using `"#"` as first character of the line⁵.

The format of the line follows this prototype:

```
<cut_name> <histo_level> <filterout> <par1> <par2> <par3> ... <par10>
```

- `cut_name` : the name of the cut as defined in the cut library
- `histo_level` : this flag is not used inside **DCut** it is provided for an integration with an histogram facility (as done in **AMSAnaTool** described in the next section).
- `filter_out` : if `> 0` an output file is filled with the events passing the cut in the cut chain. The file name is of the format `<cut_name>_filtered.root`.
- `<par1> <par2> <par3> ... <par10>` : parameters for the cut 1 to 10.

The first cut in the cut chain has a special meaning since if it is false the other cuts are not evaluated, this responds to the logic to have a precut which selects the basic conditions to evaluate the other cuts.

1.4 The Analysis tool

The class **AMSAnaTool** integrates the Cut and the Histogram Managers into a complete Analysis Tool.

With the use of this tool an user can exploit all the features of the cut manger, in particular a user can:

⁵Note that the `"#"` must be the *very first* character of the line, without any leading whitespace, tab or other non-printable character.

```

#Example datacard # first field: a string with the name of the cut
# # second field: 0 no histos for this cut #          1
histos if the cut is satisfied in the cut chain #          2
also histo for the cut as single/first(after precut) cut #
3 also histo for the cut as last(all but this) cut # third field:
0 no output data file #          1 output file
<cut_name>_filtered.root with the events passing #
the cut in the cut chain # fourth field and above: cut parameters
#####
#cut name |do histos|Filter out| Parameters #
Emin  Emax PreCut          1          0          0.  10000. #
Number of tracks less than EvNTrack          1          0          3
MyTrd          1          1          2 #
Mmin  Mmax PartMassRange  3          0          0.5  100.

```

Figure 1.1: Example of a datacard

- define his own cuts or import them from other users editing the `DCut.cxx` file,
- define a set of histograms via his own **DHFU** class⁶,
- individuate a set of cuts to be applied via a text datacard,
- decide to have instances of his own **DHFU** for the various step of the cut chain, or/and for each cut as single or as last cut still simply editing the data card,
- decide whether to save in a file the events passing a given cut or not by setting its `filter_out` flag accordingly in the datacard.

At the end of the analysis job, the user will obtain a root file containing the required histograms disposed in directories, as shown in fig 1.3. In the main directory of the output file there are also two histograms named `Book` and `BookSeq`(fig. 1.2) showing respectively the number of events passing each cut as single and those passing the various steps in the cut chain.

The class **AMSAnaTool** contains an instance of the class **DCut** and a **TObjArray** to hold multiple instances of the **DHFU** class.

The user is required to use only three methods from the **AMSAnaTool** class: `AMSAnaTool::BookHistos()`, `AMSAnaTool::ProcessEvent()` and `AMSAnaTool::UTerminate()`.

In the `AMSAnaTool::BookHistos()` method the initialisation is done in this way:

1. (**DCut** Initialisation)the datacard is read⁷, the existence of the required cuts is verified and in the case of failure the program stops, the `book` and `bookseq` histograms are created and all the relevant counter(s) are initialised,
2. The output file is opened⁸, and the appropriate directory structure is created. Multiple instances of the **DHFU** class are created according to the value of the histogram flags in the datacard, the name of each instantiated object is built from the cut name and the cut position in the cut chain, in order to provide an easy way to fill the right set of histograms,

⁶Also initialising **Categories** if need be.

⁷As default the data card file name is `dcut.conf`, a different file name can be passed as argument in the `AMSAnaTool` constructor.

⁸The default output file name is `Output.root` a different file name can be passed as argument in the `AMSAnaTool::BookHistos()` method.

	PreCut	EvNTrack	NTkHits	PartMassRange
Neg	95	88	75	3
Bad	499	455	387	87
Good	4230	4228	3791	565

Figure 1.2: Example of cut history bookkeeping (with the categories feature).

3. the book and bookseq histograms are appended to the output file,
4. the file(s) for the output of the selected events are opened⁹.

At each event the user is required to call the method `AMSAnaTool::ProcessEvent()` and to pass as argument the pointer to the AMS event class `AMSEventR`.

At each call of the `AMSAnaTool::ProcessEvent()` method the selected cuts are evaluated and, depending on the result, the actions required via the data card flags are performed. These actions corresponds to the filling of the appropriate sets of histograms and to the saving of the events in an external file.

At the end of the loop on the events the user must call the method `AMSAnaTool::UTerminate()` to properly save the histograms and close the opened files.

1.4.1 The user point of view

Summarising the actions that must be done by the user to customise and run the analysis tool are:

1. edit the file `DCut.cxx` to add new cuts in the cut library,

⁹For this action to be successful the user must provide the pointer to the AMS data `TTree` via the method `AMSAnaTool::SetInputTree()`. This pointer is needed to create a clone of the AMS `TTree` within the output file.

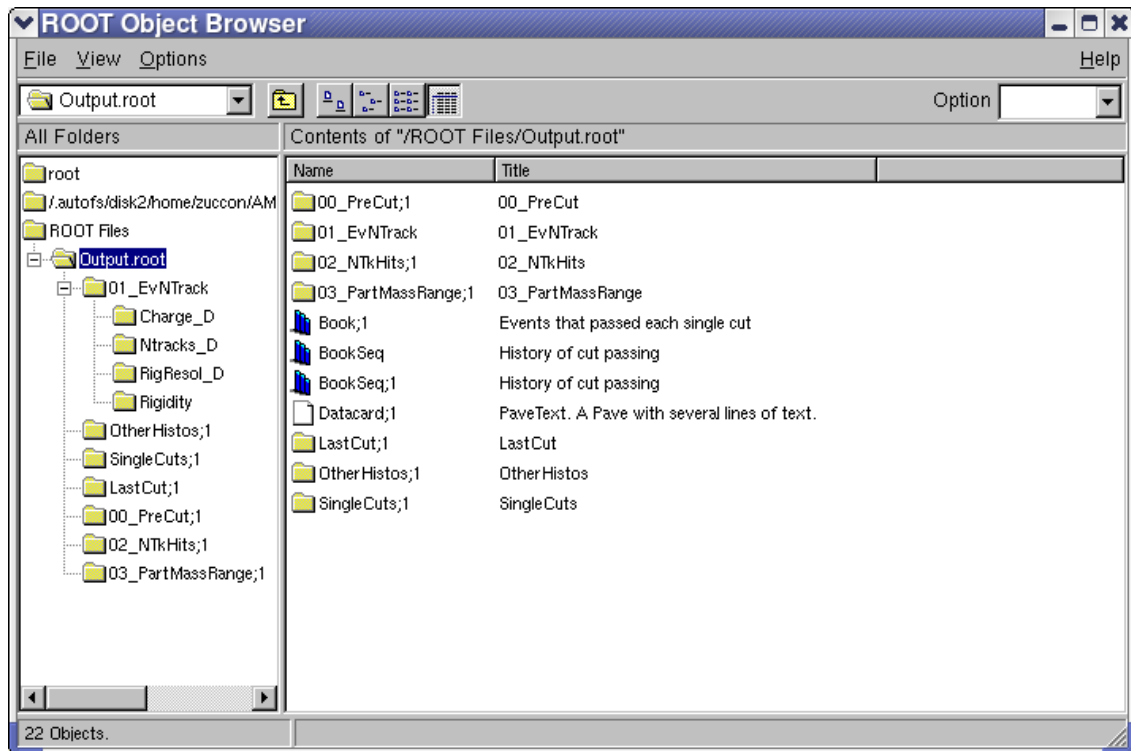


Figure 1.3: Example of an output file.

2. edit the file `DHFU.cxx` to book the histograms and to add the code that define how to fill it,
3. write a program that opens an AMS data file and loops on the events. This program, that we call `looper`, must contain the three essential calls to the **AMSAnaTool** methods. An example of the `looper` is shown in fig. 1.4.1,
4. edit a custom datacard.

The analysis can be done in different ways, corresponding to (slightly) different versions of the `looper`. Within the distributed software package (see appendix) examples on how to perform an analysis are provided¹⁰.

1.4.2 Notes on the particle selection for the analysis

The bulk of the AMS data is expected to have one single reconstructed particle in the event, corresponding to a single particle impinging on the detector. In the AMS reconstruction, quite loose criteria are used to define what a "Particle" is, in particular it can exist a reconstructed "Particle" without a reconstructed track. As a consequence in a reconstructed AMS event there is usually more than one reconstructed particle. However, when applying very basic requirements as for a definition of a "Normal Particle" (i.e. asking for a reconstructed track + a charge determination + a beta measurement), the particle multiplicity drops drastically. In fact only a few per mille of the events contains more than one reconstructed particle.

Following these considerations in our tool the events with more than one "Normal Particle" are

¹⁰Interactive with CINT, interactive with TSelector and batch analysis compiled versus static/shared libraries.

```
#include "root_RVS.h"

#include "AMSAnaTool.h"

int main() {

    int maxev=10000;
    AMSChain* ams = new AMSChain();
    ams->Add(".././files/test.root");

    AMSAnaTool *myana=new AMSAnaTool();

    myana->SetInputTree((TTree*) ams);
    myana->BookHistos("Output.root");

    //loop on the events
    Int_t ntot=(int)ams->GetEntries();
    cout <<"Total entries found="<<ntot<<endl;
    float perc=0;
    float pperc=0;
    int nentries=min(maxev,ntot);
    for (int ii=0;ii<nentries;ii++){
        perc=ii/(nentries*1.);
        if (perc>=pperc){printf("Processed %5.0f%%\n",pperc*100);pperc+=0.1;}
        ams->GetEvent(ii);
        myana->ProcessEvent(ams->pEvent());
    }

    myana->UTerminate();

    printf("We have processed %d events\n",nentries );
    return 0;
}
```

Figure 1.4: Example of a loop on AMS events

internally discarded; this behaviour should not affect the analysis of the cosmic rays components, however for other analysis like gamma rays measurements this behaviour must be modified.

1.5 Conclusions and acknowledgements

A new software tool for the analysis of the AMS data has been presented. This tool is integrated within the AMS analysis framework and provides an easy way to do a cut based analysis.

A standalone use of the histogram manager has also been presented.

A quick start guide and a reference manual are provided in the following of this note.

We want to acknowledge the contribution of B.Bertucci and of the AMS Perugia group for the help provided in designing, optimising and debugging this analysis tool.

Appendix A: quick start guide

The AMS Analysis Tool is distributed as a gzipped tar file, that can be downloaded at the address:

`http://ams.pg.infn.it/~zuccon/anatool/`

The tool and the examples are meant to be used within standard **AMSRoot2** framework. The tar archive file must be untarred in the main directory of the **AMSRoot2** framework, the directory `analysis-tool` where can found the code and various examples will be created.

The **AMSRoot2** framework can be obtained following the link:

`AMS Root Analysis Documentation`

on the main AMS web page:

`http://ams.cern.ch`

Reference Manual

Chapter 2

AMS Analysis Tool Hierarchical Index

2.1 AMS Analysis Tool Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AMSAnaTool	25
Categories	30
CList	34
DCut	36
DHF	42
DHFU	47
OneCut	49

Chapter 3

AMS Analysis Tool Class Index

3.1 AMS Analysis Tool Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AMSAnaTool (Standard AMS Analysis Class)	25
Categories (Class Categories)	30
CList (Structure CList)	34
DCut (Cut Manager)	36
DHF (Class DHF Histogram Manager)	42
DHFU (User version of the Histogram Manager class DHF (p. 42))	47
OneCut (Class OneCut)	49

Chapter 4

AMS Analysis Tool Class Documentation

4.1 AMSAnaTool Class Reference

Standard AMS Analysis Class.

```
#include <AMSAAnaTool.h>
```

Public Member Functions

- **AMSAAnaTool** (char *datacard="dcut.conf")
Default constructor the argument is the name of the datacard file.
- **~AMSAAnaTool** ()
Default destructor.
- void **BookHistos** (char *filename="Ouptut.root")
Initializes the Histograms and opens the output file.
- void **ProcessEvent** (AMSEventR *event)
Routine to be called at each event to actually calculate the cut and fill the histograms. It requires in input the pointer to the AMS event.
- void **UTerminate** ()
Saves the histograms and close the opened files.
- void **SetInputTree** (TTree *input)
Set the Input TTree pointer and enable the filtering option (needed for filterout).
- void **SaveFiles** ()
Saves the opened files.

Private Member Functions

- void **FillOther1** (char *hname, float Xin, float w=1)
Fills the 1D service histograms.
- void **FillOther2** (char *hname, float Xin, float Yin, float w=1)
Fills the 2D service histograms.

Private Attributes

- **DCut * Taglio**
Pointer to the Cut Manager.
- **TFile * Output**
Pointer to the Output file.
- char **OutFileName** [200]
Name of the OutputFile.
- **TObjArray * cuthistos**
*Container of the **DHFU**(p.47) Classes.*
- **TObjArray * OtherHistos**
Container of the service histograms.
- float **EventProcessed**
Number of Events Processed.
- float **EventLevel1**
Number of Events with a TOF Trigger.
- float **EventConPart**
Number of Events with at least 1 Particle (a la Vitaly).
- float **EventConPartN**
Number of Events with at least 1 NormalParticle (with a track).
- float **EventConPartN1**
Number of Events with only 1 NormalParticle (with a track).

4.1.1 Detailed Description

Standard AMS Analysis Class.

The Class AMSAnaTool provides the entry points to use this analysis tool. To use this tool you should use a looper on the AMS trees, examples are available in this package.

In general you must use four functions:

1) INITIALIZATION

You must create a new AMSAnaTool passing as argument the name of the datacard file (default is dcut.conf):

```
AMSAnaTool *myana=new AMSAnaTool("datacard_filename");
```

to complete the initialization you must book the histograms providing the name of the output file (default is Output.root) with the command:

```
myana->BookHistos("name_of_the_output_file");
```

2) PROCESSING

At each event you must call:

```
myana->ProcessEvent( pointer_to_amsevent);
```

providing the pointer to the AMSEventR Class

3) TERMINATION

When the loop ends you must call:

```
myana-> UTerminate()(p. 28);
```

If you choose to enable the filterout option in the datacard you must provide a pointer to the input AMS tree before booking the histograms with the command:

```
myana-> SetInputTree(pointer_to_AMSTree);
```

Optionally you can call:

```
myana->SaveFiles()(p. 28);
```

every let's say 100000 events during the loop to save the output files.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AMSAnaTool::AMSAnaTool (char * *datacard* = "dcut.conf")

Default constructor the argument is the name of the datacard file.

4.1.2.2 AMSAnaTool::~~AMSAnaTool ()

Default destructor.

4.1.3 Member Function Documentation

4.1.3.1 void AMSAnaTool::BookHistos (char * *filename* = "Ouptut.root")

Initializes the Histograms and opens the output file.

4.1.3.2 void AMSAnaTool::FillOther1 (char * *hname*, float *Xin*, float *w* = 1) [private]

Fills the 1D service histograms.

4.1.3.3 void AMSAnaTool::FillOther2 (char * *hname*, float *Xin*, float *Yin*, float *w* = 1) [private]

Fills the 2D service histograms.

4.1.3.4 void AMSAnaTool::ProcessEvent (AMSEventR * *event*)

Routine to be called at each event to actually calculate the cut and fill the histograms. It requires in input the pointer to the AMS event.

4.1.3.5 void AMSAnaTool::SaveFiles ()

Saves the opened files.

4.1.3.6 void AMSAnaTool::SetInputTree (TTree * *input*)

Set the Input TTree pointer and enable the filtering option (needed for filterout).

4.1.3.7 void AMSAnaTool::UTerminate ()

Saves the histograms and close the opened files.

4.1.4 Member Data Documentation**4.1.4.1 TObjArray* AMSAnaTool::cuthistos [private]**

Container of the DHFU(p.47) Classes.

4.1.4.2 float AMSAnaTool::EventConPart [private]

Number of Events with at least 1 Particle (a la Vitaly).

4.1.4.3 float AMSAnaTool::EventConPartN [private]

Number of Events with at least 1 NormalParticle (with a track).

4.1.4.4 float AMSAnaTool::EventConPartN1 [private]

Number of Events with only 1 NormalParticle (with a track).

4.1.4.5 float AMSAnaTool::EventLevel1 [private]

Number of Events with a TOF Trigger.

4.1.4.6 float AMSAnaTool::EventProcessed [private]

Number of Events Processed.

4.1.4.7 TObjArray* AMSAnaTool::OtherHistos [private]

Container of the service histograms.

4.1.4.8 char AMSAnaTool::OutFileName[200] [private]

Name of the OutputFile.

4.1.4.9 TFile* AMSAnaTool::Output [private]

Pointer to the Output file.

4.1.4.10 DCut* AMSAnaTool::Taglio [private]

Pointer to the Cut Manager.

4.2 Categories Class Reference

Class Categories.

```
#include <Categories.h>
```

Public Member Functions

- **Categories** ()
Default constructor. It defines a single category named "AllEvents".

Static Public Member Functions

- int **GetNCat** ()
Returns the number of categories.
- char * **GetCatName** (int ii)
Returns the name of the n-th category.
- short int **GetCatColor** (int ii)
Return the color index associated to a category.
- void **SetNCat** (int ii)
Sets the number of categories.
- void **SetCatName** (int ii, char *name)
Sets the Name of the n-th category.
- void **SetCatColor** (int ii, short int col)
Sets the color index associated to a category.
- void **SetFun** (CATFUN oo)
Sets the pointer to the function (function name) which returns the category an event belong to.
- int **GetCatIndex** (AMSEventR *ev, ParticleR *part)
Returns the category number an event belong to.
- char * **GetCategory** (AMSEventR *ev, ParticleR *part)
Returns the category name an event belong to.

Static Private Attributes

- int **_ncat**
Number of categories to classify the events.
- char **_catname** [MAXCAT][MAXLEN]
Names of the categories used to classify the events.

- short int `_color` [MAXCAT] = {1}

Color index associated to a category.

- **CATFUN** `_SelFun`

Pointer to the function (function name) that returns the category an event belong to.

4.2.1 Detailed Description

Class Categories.

The Class Categories is used to implement an additional (and optional) feature of the Histogram Manager described in the manual. If you want to use this feature you must:

1) Write a function that returns the category of a given event, the function must be compliant to the prototype:

```
typedef int (*CATFUN) (AMSEventR *, ParticleR *);
```

Example:

```
int GoodBadNeg(AMSEventR* ev, ParticleR* part){
MCEventgR* Gen=ev->pMCEventg(0);
double res=fabs (1. - ((part->Momentum) / Gen->Momentum));
if (part->Momentum*Gen->Momentum <0) return 2;
else if (res < 0.12) return 0;
else return 1;
}
```

2) Since the Categories class is static you do not need to call its constructor. It is enough to include the header in your looper and to set the values of the class elements as sketched in the example. The definition of the elements must be done before the creation of the **DHF**(p. 42) or **AMSAnaTool**(p. 25) object.

Example:

```
Categories::SetNCat(3);
Categories::SetCatName(p. 32)(0,"Good");
Categories::SetCatName(p. 32)(1,"Bad");
Categories::SetCatName(p. 32)(2,"Neg");
Categories::SetFun(GoodBadNeg);
```

3) Create a **DHF**(p. 42) or an **AMSAnaTool**(p. 25) object using the correct constructor

Examples:

```
DHF(p. 42) *histos= new DHF(gDirectory);
AMSAnaTool(p. 25) *myana=new AMSAnaTool(p. 25)("dcut.conf");
```

As result you will get the break down of the events passing the cuts separately for each category and you will be allowed to use the functions **DHF::Define**(p. 45)(...) and **DHF::FillSet**(p. 45)(...) to automatically have different plots for each category.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Categories::Categories ()

Default constructor.It defines a single category named "AllEvents".

4.2.3 Member Function Documentation

4.2.3.1 short int Categories::GetCatColor (int *ii*) [inline, static]

Return the color index associated to a category.

4.2.3.2 char * Categories::GetCategory (AMSEventR * *ev*, ParticleR * *part*) [static]

Returns the category name an event belong to.

4.2.3.3 int Categories::GetCatIndex (AMSEventR * *ev*, ParticleR * *part*) [static]

Returns the category number an event belong to.

4.2.3.4 char* Categories::GetCatName (int *ii*) [inline, static]

Returns the name of the n-th category.

4.2.3.5 int Categories::GetNCat () [inline, static]

Returns the number of categories.

4.2.3.6 void Categories::SetCatColor (int *ii*, short int *col*) [inline, static]

Sets the color index associated to a category.

4.2.3.7 void Categories::SetCatName (int *ii*, char * *name*) [static]

Sets the Name of the n-th category.

4.2.3.8 void Categories::SetFun (CATFUN *oo*) [inline, static]

Sets the pointer to the function (function name) which returns the category an event belong to.

4.2.3.9 void Categories::SetNCat (int *ii*) [static]

Sets the number of categories.

4.2.4 Member Data Documentation

4.2.4.1 `char Categories::catname` [static, private]

Names of the categories used to classify the events.

4.2.4.2 `short int Categories::_color = {1}` [static, private]

Color index associated to a category.

4.2.4.3 `int Categories::_ncat` [static, private]

Number of categories to classify the events.

4.2.4.4 `CATFUN Categories::_SelfFun` [static, private]

Pointer to the function (function name) that returns the category an event belong to.

4.3 CList Struct Reference

Structure CList.

```
#include <DCut.h>
```

Public Attributes

- char **Name** [80]
Name of the Cut.
- **OneCut * Cut**
Pointer to the Cut in the library.
- int **Passed**
Is this cut passed?
- int **DoHistos**
Do we need to book and fill histos for this cut?
- int **FilterEv**
Do we need to filter out the events passing the cut.
- TFile * **FileOut**
Pointer to the output file (for filtering).
- TTree * **TreeOut**
Pointer to the output tree (for filtering).
- int **Selected**
Counter of particles that passed the cut.
- int **Rejected**
Counter of particles that didn't pass the cut.
- float **ParList** [MAXPAR]
Vector of the parameters of the cut.

4.3.1 Detailed Description

Structure CList.

The structure CList represents a line of the datacard and holds the pointer to the matched **OneCut**(p. 49) Object

4.3.2 Member Data Documentation

4.3.2.1 OneCut* CList::Cut

Pointer to the Cut in the library.

4.3.2.2 int CList::DoHistos

Do we need to book and fill histos for this cut?

4.3.2.3 TFile* CList::FileOut

Pointer to the output file (for filtering).

4.3.2.4 int CList::FilterEv

Do we need to filter out the events passing the cut.

4.3.2.5 char CList::Name[80]

Name of the Cut.

4.3.2.6 float CList::ParList[MAXPAR]

Vector of the parameters of the cut.

4.3.2.7 int CList::Passed

Is this cut passed?

4.3.2.8 int CList::Rejected

Counter of particles that didn't pass the cut.

4.3.2.9 int CList::Selected

Counter of particles that passed the cut.

4.3.2.10 TTree* CList::TreeOut

Pointer to the output tree (for filtering).

4.4 DCut Class Reference

Cut Manager.

```
#include <DCut.h>
```

Public Member Functions

- **DCut** (char *input="dcut.conf")
Default Constructor.
- **~DCut** ()
Default Destructor.
- ParticleR * **PtoC** ()
Function that return the pointer to the Particle.
- AMSEventR * **Event** ()
Function that return the pointer to the Event.
- void **EvalList** ()
Evaluate the cuts.
- int **GetNFound** ()
Returns the number of configured cut.
- void **SetParticle** (ParticleR *p, AMSEventR *eVENT)
Sets the pointer to the event and to the particle.
- int **Passed** (int ii)
Is the cut ii passed ?
- int **AllPassed** (int ii)
Are all cuts up to ii passed ?
- int **AllButThis** (int mm)
Checks all cuts excluding the argument.
- int **DoHistos** (int ii)
Returns 1 if histos for this cut have been booked 0 otherwise.
- int **FilterEv** (int ii)
Returns 1 if filter out for this cut is requested 0 otherwise.
- char * **GetCutName** (int ii)
Returns the cut name.
- int **GetCutPar** (int ii, int parnum)
Returns parameter parnum of cut ii.

- **TTree * GetCutOutTree** (int ii)
Returns the pointer to the OutTree (for filtering).
- **TFile * GetCutOutFile** (int ii)
Returns the pointer to the OutFile (for filtering).
- **void FilteringOn** (TTree *intree)
Enables the filtering option (and opens the requested files).
- **void SaveOutFiles** ()
Saves the output files.
- **void CloseOutFiles** ()
Saves and close the output files.

Public Attributes

- **TH2F * Book**
Table of events passing each single cut.
- **TH2F * BookSeq**
Table of events passing the cuts in the given order.
- **TPaveText * DCard**
Text object containing a copy of the datacard.

Private Member Functions

- **void Library_init** ()
Initialize The library of cuts.
- **void List_init** ()
Initialize The list of cuts.
- **int parseconf** (char *filename)
Read and parse of config file.
- **void OpenFileForExit** (int cutnum)
Open the files for the saving of filtered events.
- **void SetCutOutTree** (int ii, TTree *tt)
Set the pointer to the OutTree (for filtering).
- **void SetCutOutFile** (int ii, TFile *file)
Set the pointer to the OutFile (for filtering).

Private Attributes

- TObjArray * **_Cuts**
Array of the library of cuts.
- int **_NFound**
Num of cuts found in config.
- CList **_CutList** [MAXCUTS]
Vector of structures CList(p.34) representing the datacard.
- int **_globalFilter**
Internal Flag for filtering.
- TTree * **_InputTree**
Pointer to the input TTree (needed for filtering).

Static Private Attributes

- TObjArray * **_gCuts**
Array of the library of cuts.

4.4.1 Detailed Description

Cut Manager.

The class DCut is the implementation of the cut manager.

Their principal components are:

1. A cut library (implemented via a TObjectArray of objects **OneCut**(p.49))
2. The image of the datacard (implemented with a vector of **CList**(p.34) structures)
3. A bookeping of the events passing the cuts (implemented with two histos ans with some counter in the **CList**(p.34) structure)

When the constructor is called this chain of events is triggered:

1. The cut library is initialized with the cuts defined in **Library_init**()(p.40)
2. The datacard (default name dcut.conf) is read and checked by **parseconf(char *filename)**(p.40);
3. The selected cuts are initialized by **List_init**()(p.40)

At each event the user is expected to call:

SetParticle(ParticleR * p,AMSEventR * event)(p.41)

to set the pointers and:

EvalList()(p.39)

to trigger the evaluation of the cuts.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 DCut::DCut (char * *input* = "dcut.conf")

Default Constructor.

4.4.2.2 DCut::~~DCut ()

Default Destructor.

4.4.3 Member Function Documentation

4.4.3.1 int DCut::AllButThis (int *mm*)

Checks all cuts excluding the argument.

4.4.3.2 int DCut::AllPassed (int *ii*)

Are all cuts up to *ii* passed ?

4.4.3.3 void DCut::CloseOutFiles ()

Saves and close the output files.

4.4.3.4 int DCut::DoHistos (int *ii*) [inline]

Returns 1 if histos for this cut have been booked 0 otherwise.

4.4.3.5 void DCut::EvalList ()

Evaluate the cuts.

4.4.3.6 AMSEventR * DCut::Event ()

Function that return the pointer to the Event.

4.4.3.7 int DCut::FilterEv (int *ii*) [inline]

Returns 1 if filter out for this cut is requested 0 otherwise.

4.4.3.8 void DCut::FilteringOn (TTree * *intree*)

Enables the filtering option (and opens the requested files).

4.4.3.9 char* DCut::GetCutName (int *ii*) [inline]

Returns the cut name.

4.4.3.10 `TFile* DCut::GetCutOutFile (int ii)` [inline]

Returns the pointer to the OutFile (for filtering).

4.4.3.11 `TTree* DCut::GetCutOutTree (int ii)` [inline]

Returns the pointer to the OutTree (for filtering).

4.4.3.12 `int DCut::GetCutPar (int ii, int parnum)` [inline]

Returns parameter parnum of cut ii.

4.4.3.13 `int DCut::GetNFound ()` [inline]

Returns the number of configured cut.

4.4.3.14 `void DCut::Library_init ()` [private]

Initialize The library of cuts.

4.4.3.15 `void DCut::List_init ()` [private]

Initialize The list of cuts.

4.4.3.16 `void DCut::OpenFileForExit (int cutnum)` [private]

Open the files for the saving of filtered events.

4.4.3.17 `int DCut::parseconf (char * filename)` [private]

Read and parse of config file.

4.4.3.18 `int DCut::Passed (int ii)` [inline]

Is the cut ii passed ?

4.4.3.19 `ParticleR * DCut::PtoC ()`

Function that return the pointer to the Particle.

4.4.3.20 `void DCut::SaveOutFiles ()`

Saves the output files.

4.4.3.21 `void DCut::SetCutOutFile (int ii, TFile * file)` [inline, private]

Set the pointer to the OutFile (for filtering).

4.4.3.22 void DCut::SetCutOutTree (int *ii*, TTree * *tt*) [inline, private]

Set the pointer to the OutTree (for filtering).

4.4.3.23 void DCut::SetParticle (ParticleR * *p*, AMSEventR * *eVENT*)

Sets the pointer to the event and to the particle.

4.4.4 Member Data Documentation

4.4.4.1 CList DCut::_CutList[MAXCUTS] [private]

Vector of structures CList(p.34) representing the datacard.

4.4.4.2 TObjArray* DCut::_Cuts [private]

Array of the library of cuts.

4.4.4.3 TObjArray* DCut::_gCuts [static, private]

Array of the library of cuts.

4.4.4.4 int DCut::_globalFilter [private]

Internal Flag for filtering.

4.4.4.5 TTree* DCut::_InputTree [private]

Pointer to the input TTree (needed for filtering).

4.4.4.6 int DCut::_NFound [private]

Num of cuts found in config.

4.4.4.7 TH2F* DCut::Book

Table of events passing each single cut.

4.4.4.8 TH2F* DCut::BookSeq

Table of events passing the cuts in the given order.

4.4.4.9 TPaveText* DCut::DCard

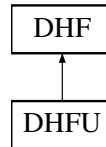
Text object containing a copy of the datacard.

4.5 DHF Class Reference

Class DHF Histogram Manager.

```
#include <DHF.h>
```

Inheritance diagram for DHF::



Public Member Functions

- **DHF** (TDirectory *Dir=0, char *name="HManager", char *title="default HManager")
Default constructor.
- **DHF** (const **DHF** &)
Copy constructor.
- **~DHF** ()
Default destructor.
- virtual void **Init** ()
Initialize the class (call BookHistos plus some other init).
- void **Add** (TObject *hh)
Add an histogram to the manager.
- void **SetCurrentPart** (AMSEventR *ev, ParticleR *part)
Sets the pointers to the event and to the current particles.
- void **SetDrawOption** (char *basename, char *option)
Set some standard draw option for the histograms.
- TObject * **GetHist** (char *name)
Returns the pointer to the histogram with the required name.
- TDirectory * **GetDir** () const
Returns the pointer to the directory holding the histograms.
- virtual void **FillAll** ()
Fill all histograms.
- TDirectory * **Define** (char *name, char *title, int binx, Axis_t lowx, Axis_t upx, int biny, Axis_t lowy, Axis_t upy, char *dirname=0)
Books 2d histo sets.

- TDirectory * **Define** (char *name, char *title, int binx, Axis_t lowx, Axis_t upx, char *dirname=0)
Books 1d histo sets.
- TDirectory * **Define** (char *prof, char *name, char *title, int binx, Axis_t lowx, Axis_t upx, char *dirname)
Books Profile histo sets.
- void **Fill** (char *histo, Axis_t X1, Axis_t X2=1., Stat_t w=1.)
Fills a 1D or 2D or Profile histogram.
- void **FillSet** (char *histo, Axis_t X1, Axis_t X2=1., Stat_t w=1.)
Fills a 1D or 2D or Profile histos sets.

Public Attributes

- AMSEventR * **Event**
Pointer to AMS Event.
- ParticleR * **Partcl**
Pointer to the particle.

Protected Member Functions

- TObjArray * **GetHlist** () const
Returns the pointer to the container (TObjArray) of the histograms.
- void **Sumw2** ()
Request the error calculation for all the histograms.
- virtual void **BookHistos** ()
User routine to book the histos.

Protected Attributes

- TDirectory * **fDir**
Main directory where to store the histos.
- TObjArray * **fHlist**
Tobject array containing the histos.

4.5.1 Detailed Description

Class DHF Histogram Manager.

The class DHF implements the histogram manager. In the method **BookHistos()**(p. 44) the user can define a set histograms he want to fill at each cut level.

The method **Define()**(p. 45) provides a way to book automatically a many of histograms versions of the same histograms corresponding to the different event classes defined in the Class **Categories**(p. 30) these histograms are putted in a subdirectory

In the method **FillAll()**(p. 45), the user is expected to define the rules to fill the histograms. The method **FillSet()**(p. 45) is provided to fill automatically the histos booked with the method **Define()**(p. 45).

4.5.2 Constructor & Destructor Documentation

4.5.2.1 DHF::DHF (TDirectory * Dir = 0, char * name = "HManager", char * title = "default HManager")

Default constructor.

Default constructor, the arguments are a TDirectory where to store the histos, the name and the title of DHF Object

If the TDirectory argument is not passed or not valid the current directory(gDirectory) is set as fDir

4.5.2.2 DHF::DHF (const DHF &)

Copy constructor.

4.5.2.3 DHF::~DHF ()

Default destructor.

4.5.3 Member Function Documentation

4.5.3.1 void DHF::Add (TObject * hh) [inline]

Add an histogram to the manager.

4.5.3.2 void DHF::BookHistos () [protected, virtual]

User routine to book the histos.

Reimplemented in **DHFU** (p. 47).

4.5.3.3 TDirectory * DHF::Define (char * prof, char * name, char * title, int binx, Axis_t lowx, Axis_t upx, char * dirname)

Books Profile histo sets.

4.5.3.4 `TDirectory * DHF::Define (char * name, char * title, int binx, Axis_t lowx, Axis_t upx, char * dirname = 0)`

Books 1d histo sets.

4.5.3.5 `TDirectory * DHF::Define (char * name, char * title, int binx, Axis_t lowx, Axis_t upx, int biny, Axis_t lowy, Axis_t upy, char * dirname = 0)`

Books 2d histo sets.

4.5.3.6 `void DHF::Fill (char * histo, Axis_t X1, Axis_t X2 = 1., Stat_t w = 1.)`

Fills a 1D or 2D or Profile histogram.

4.5.3.7 `void DHF::FillAll () [virtual]`

Fill all histograms.

Reimplemented in **DHFU** (p. 48).

4.5.3.8 `void DHF::FillSet (char * histo, Axis_t X1, Axis_t X2 = 1., Stat_t w = 1.)`

Fills a 1D or 2D or Profile histos sets.

4.5.3.9 `TDirectory* DHF::GetDir () const [inline]`

Returns the pointer to the directory holding the histograms.

4.5.3.10 `TObject * DHF::GetHist (char * name)`

Returns the pointer to the histogram with the required name.

4.5.3.11 `TObjArray* DHF::GetHlist () const [inline, protected]`

Returns the pointer to the container (TObjArray) of the histograms.

4.5.3.12 `void DHF::Init () [virtual]`

Initialize the class (call BookHistos plus some other init).

4.5.3.13 `void DHF::SetCurrentPart (AMSEventR * ev, ParticleR * part) [inline]`

Sets the pointers to the event and to the current particles.

4.5.3.14 `void DHF::SetDrawOption (char * basename, char * option)`

Set some standard draw option for the histograms.

4.5.3.15 void DHF::Sumw2 () [protected]

Request the error calculation for all the histograms.

4.5.4 Member Data Documentation**4.5.4.1 AMSEventR* DHF::Event**

Pointer to AMS Event.

4.5.4.2 TDirectory* DHF::fDir [protected]

Main directory where to store the histos.

4.5.4.3 TObjArray* DHF::fHlist [protected]

Tobject array containing the histos.

4.5.4.4 ParticleR* DHF::Partcl

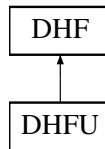
Pointer to the particle.

4.6 DHFU Class Reference

User version of the Histogram Manager class **DHF**(p. 42).

```
#include <DHFU.h>
```

Inheritance diagram for DHFU::



Public Member Functions

- **DHFU** (TDirectory *Dir=0, char *name="HManager", char *title="default HManager")
Standar Constructor as in DHF(p. 42).
- void **FillAll** ()
Implementation of the DHF(p. 42) **FillAll**()(p. 48).

Protected Member Functions

- void **BookHistos** ()
Implementation of the DHF(p. 42) **BookHistos**()(p. 47).

4.6.1 Detailed Description

User version of the Histogram Manager class **DHF**(p. 42).

4.6.2 Constructor & Destructor Documentation

- 4.6.2.1 DHFU::DHFU** (TDirectory * *Dir* = 0, char * *name* = "HManager", char * *title* = "default HManager") [inline]

Standar Constructor as in **DHF**(p. 42).

4.6.3 Member Function Documentation

- 4.6.3.1 void DHFU::BookHistos** () [protected, virtual]

Implementation of the **DHF**(p. 42) **BookHistos**()(p. 47).

Reimplemented from **DHF** (p. 44).

4.6.3.2 void DHFU::FillAll () [virtual]

Implementation of the **DHF**(p. 42) **FillAll**(p. 48).

Reimplemented from **DHF** (p. 45).

4.7 OneCut Class Reference

Class OneCut.

```
#include <OneCut.h>
```

Public Member Functions

- **OneCut** ()
Default dummy constructor.
- **OneCut** (const **OneCut** &orig)
Copy constructor.
- **OneCut** (char *name, char *title, **PF** function, int npar)
Standard constructor.
- virtual **~OneCut** ()
Standard destructor.
- void **Clear** ()
Reset the content of the object.
- void **SetPar** (float *par)
Set the values of the parameters.
- int **Eval** ()
Evaluate the cut (1=passed, 0=failed,>1 error).
- int **GetNPar** ()
Return the number of parameters of the cuts.
- ParticleR * **GetPart** ()
Return the pointer to the Particle.
- AMSEventR * **GetEvent** ()
Return the pointer to the Event.

Static Public Member Functions

- void **SetPointers** (ParticleR *PtoC, AMSEventR *Event)
Set the pointers to the event.

Private Types

- typedef int(* **PF**)(ParticleR *, AMSEventR *, float *)
Function pointer type for functions describing the cuts.

Private Attributes

- **PF _CutFun**
Pointer to the function which implement the cut.
- **int _Npar**
Number of parameters of the cut.
- **float _Param [MAXPAR]**
Vector of the cut parameters up to 10.

Static Private Attributes

- **ParticleR * _PtoC**
Pointer to the particle of the event.
- **AMSEventR * _Event**
Pointer to the AMS Event.

4.7.1 Detailed Description

Class OneCut.

The class OneCut provides the abstraction of a selection criterium, it represents the object that is contained in the "cut library", it contains the pointer to the real cut function that should be of the form: `int myfun(ParticleR * part , AMSEventR * event, float * param)`

4.7.2 Member Typedef Documentation

4.7.2.1 `typedef int(* OneCut::PF)(ParticleR *,AMSEventR *,float *) [private]`

Function pointer type for functions describing the cuts.

4.7.3 Constructor & Destructor Documentation

4.7.3.1 `OneCut::OneCut ()`

Default dummy constructor.

4.7.3.2 `OneCut::OneCut (const OneCut & orig)`

Copy constructor.

4.7.3.3 OneCut::OneCut (char * *name*, char * *title*, PF *function*, int *npar*)

Standard constructor.

Arguments are the name and the title of the OneCut Object, the name of the function implementing the cut and the number of the parameters of the cuts

4.7.3.4 virtual OneCut::~~OneCut () [inline, virtual]

Standard destructor.

4.7.4 Member Function Documentation

4.7.4.1 void OneCut::Clear ()

Reset the content of the object.

4.7.4.2 int OneCut::Eval ()

Evaluate the cut (1=passed, 0=failed,>1 error).

4.7.4.3 AMSEventR* OneCut::GetEvent () [inline]

Return the pointer to the Event.

4.7.4.4 int OneCut::GetNPar () [inline]

Return the number of parameters of the cuts.

4.7.4.5 ParticleR* OneCut::GetPart () [inline]

Return the pointer to the Particle.

4.7.4.6 void OneCut::SetPar (float * *par*)

Set the values of the parameters.

4.7.4.7 void OneCut::SetPointers (ParticleR * *PtoC*, AMSEventR * *Event*) [inline, static]

Set the pointers to the event.

4.7.5 Member Data Documentation

4.7.5.1 PF OneCut::_CutFun [private]

Pointer to the function which implement the cut.

4.7.5.2 AMSEventR* OneCut::_Event [static, private]

Pointer to the AMS Event.

4.7.5.3 int OneCut::_Npar [private]

Number of parameters of the cut.

4.7.5.4 float OneCut::_Param[MAXPAR] [private]

Vector of the cut parameters up to 10.

4.7.5.5 ParticleR* OneCut::_PtoC [static, private]

Pointer to the particle of the event.

Index

- ._CutFun
 - OneCut, 51
- ._CutList
 - DCut, 41
- ._Cuts
 - DCut, 41
- ._Event
 - OneCut, 51
- ._InputTree
 - DCut, 41
- ._NFound
 - DCut, 41
- ._Npar
 - OneCut, 52
- ._Param
 - OneCut, 52
- ._PtoC
 - OneCut, 52
- ._SelFun
 - Categories, 33
- ._catname
 - Categories, 33
- ._color
 - Categories, 33
- ._gCuts
 - DCut, 41
- ._globalFilter
 - DCut, 41
- ._ncat
 - Categories, 33
- ~AMSAnaTool
 - AMSAnaTool, 27
- ~DCut
 - DCut, 39
- ~DHF
 - DHF, 44
- ~OneCut
 - OneCut, 51
- Add
 - DHF, 44
- AllButThis
 - DCut, 39
- AllPassed
 - DCut, 39
- AMSAnaTool, 25
 - AMSAnaTool, 27
- AMSAnaTool
 - ~AMSAnaTool, 27
 - AMSAnaTool, 27
 - BookHistos, 27
 - cuthistos, 28
 - EventConPart, 28
 - EventConPartN, 28
 - EventConPartN1, 28
 - EventLevel1, 28
 - EventProcessed, 28
 - FillOther1, 27
 - FillOther2, 27
 - OtherHistos, 28
 - OutFileName, 28
 - Output, 29
 - ProcessEvent, 27
 - SaveFiles, 28
 - SetInputTree, 28
 - Taglio, 29
 - UTerminate, 28
- Book
 - DCut, 41
- BookHistos
 - AMSAnaTool, 27
 - DHF, 44
 - DHFU, 47
- BookSeq
 - DCut, 41
- Categories, 30
 - ._SelFun, 33
 - ._catname, 33
 - ._color, 33
 - ._ncat, 33
 - Categories, 32
 - GetCatColor, 32
 - GetCategory, 32
 - GetCatIndex, 32
 - GetCatName, 32
 - GetNCat, 32
 - SetCatColor, 32
 - SetCatName, 32

- SetFun, 32
- SetNCat, 32
- Clear
 - OneCut, 51
- CList, 34
 - Cut, 34
 - DoHistos, 34
 - FileOut, 35
 - FilterEv, 35
 - Name, 35
 - ParList, 35
 - Passed, 35
 - Rejected, 35
 - Selected, 35
 - TreeOut, 35
- CloseOutFiles
 - DCut, 39
- Cut
 - CList, 34
- cuthistos
 - AMSAnaTool, 28
- DCard
 - DCut, 41
- DCut, 36
 - _CutList, 41
 - _Cuts, 41
 - _InputTree, 41
 - _NFound, 41
 - _gCuts, 41
 - _globalFilter, 41
 - ~DCut, 39
 - AllButThis, 39
 - AllPassed, 39
 - Book, 41
 - BookSeq, 41
 - CloseOutFiles, 39
 - DCard, 41
 - DCut, 39
 - DoHistos, 39
 - EvalList, 39
 - Event, 39
 - FilterEv, 39
 - FilteringOn, 39
 - GetCutName, 39
 - GetCutOutFile, 39
 - GetCutOutTree, 40
 - GetCutPar, 40
 - GetNFound, 40
 - Library_init, 40
 - List_init, 40
 - OpenFileForExit, 40
 - parseconf, 40
 - Passed, 40
 - PtoC, 40
 - SaveOutFiles, 40
 - SetCutOutFile, 40
 - SetCutOutTree, 40
 - SetParticle, 41
- Define
 - DHF, 44, 45
- DHF, 42
 - ~DHF, 44
 - Add, 44
 - BookHistos, 44
 - Define, 44, 45
 - DHF, 44
 - Event, 46
 - fDir, 46
 - fHlist, 46
 - Fill, 45
 - FillAll, 45
 - FillSet, 45
 - GetDir, 45
 - GetHist, 45
 - GetHlist, 45
 - Init, 45
 - Partcl, 46
 - SetCurrentPart, 45
 - SetDrawOption, 45
 - Sumw2, 45
- DHFU, 47
 - BookHistos, 47
 - DHFU, 47
 - FillAll, 47
- DoHistos
 - CList, 34
 - DCut, 39
- Eval
 - OneCut, 51
- EvalList
 - DCut, 39
- Event
 - DCut, 39
 - DHF, 46
- EventConPart
 - AMSAnaTool, 28
- EventConPartN
 - AMSAnaTool, 28
- EventConPartN1
 - AMSAnaTool, 28
- EventLevel1
 - AMSAnaTool, 28
- EventProcessed
 - AMSAnaTool, 28
- fDir

- DHF, 46
- fHlist
 - DHF, 46
- FileOut
 - CList, 35
- Fill
 - DHF, 45
- FillAll
 - DHF, 45
 - DHFU, 47
- FillOther1
 - AMSAnaTool, 27
- FillOther2
 - AMSAnaTool, 27
- FillSet
 - DHF, 45
- FilterEv
 - CList, 35
 - DCut, 39
- FilteringOn
 - DCut, 39
- GetCatColor
 - Categories, 32
- GetCategory
 - Categories, 32
- GetCatIndex
 - Categories, 32
- GetCatName
 - Categories, 32
- GetCutName
 - DCut, 39
- GetCutOutFile
 - DCut, 39
- GetCutOutTree
 - DCut, 40
- GetCutPar
 - DCut, 40
- GetDir
 - DHF, 45
- GetEvent
 - OneCut, 51
- GetHist
 - DHF, 45
- GetHlist
 - DHF, 45
- GetNCat
 - Categories, 32
- GetNFound
 - DCut, 40
- GetNPar
 - OneCut, 51
- GetPart
 - OneCut, 51
- Init
 - DHF, 45
- Library_init
 - DCut, 40
- List_init
 - DCut, 40
- Name
 - CList, 35
- OneCut, 49
 - OneCut, 50
- OneCut
 - _CutFun, 51
 - _Event, 51
 - _Npar, 52
 - _Param, 52
 - _PtoC, 52
 - ~OneCut, 51
 - Clear, 51
 - Eval, 51
 - GetEvent, 51
 - GetNPar, 51
 - GetPart, 51
 - OneCut, 50
 - PF, 50
 - SetPar, 51
 - SetPointers, 51
- OpenFileForExit
 - DCut, 40
- OtherHistos
 - AMSAnaTool, 28
- OutFileName
 - AMSAnaTool, 28
- Output
 - AMSAnaTool, 29
- ParList
 - CList, 35
- parseconf
 - DCut, 40
- Partcl
 - DHF, 46
- Passed
 - CList, 35
 - DCut, 40
- PF
 - OneCut, 50
- ProcessEvent
 - AMSAnaTool, 27
- PtoC
 - DCut, 40
- Rejected

- CList, 35
- SaveFiles
 - AMSAAnaTool, 28
- SaveOutFiles
 - DCut, 40
- Selected
 - CList, 35
- SetCatColor
 - Categories, 32
- SetCatName
 - Categories, 32
- SetCurrentPart
 - DHF, 45
- SetCutOutFile
 - DCut, 40
- SetCutOutTree
 - DCut, 40
- SetDrawOption
 - DHF, 45
- SetFun
 - Categories, 32
- SetInputTree
 - AMSAAnaTool, 28
- SetNCat
 - Categories, 32
- SetPar
 - OneCut, 51
- SetParticle
 - DCut, 41
- SetPointers
 - OneCut, 51
- Sumw2
 - DHF, 45
- Taglio
 - AMSAAnaTool, 29
- TreeOut
 - CList, 35
- UTerminate
 - AMSAAnaTool, 28