

AMS Note 2005-07-01  
July 1, 2005  
Revised: July 27, 2006

# AMSRoot User Guide

J. Alcaraz (CIEMAT)

## Abstract

The AMSRoot package is a CVS standalone distribution for user analysis of AMS data. This Note presents in a hopefully simple way the installation procedure and the initial configuration steps. It also describes the analysis modes, utilities and libraries included in the package as of today.

# 1 Introduction

The standard AMS reconstruction output is a ROOT file [1], in which TTree [2] event structures are built. The AMSRoot package is a CVS standalone distribution for user analysis of AMS data. For the time being, it has only been tested on Linux and Mac OS X (> v10.4) machines. After downloading of the package, a subdirectory of the current directory called **AMSRoot** is created. We will denote this subdirectory by **\$AMSRoot** throughout the Note.

We have identified several modes to analyze AMS Root data:

1. CINT : an interactive analysis with CINT [3],
2. DYNAMIC: a compiled analysis,
3. STATIC : a compiled analysis with a fully static executable,
4. TSELECTOR : an analysis (compiled or interpreted) intended for parallel processing on several CPUs,
5. PYTHON : an analysis using the Python programming language [4].

The package contains some tools for fast inspection of AMS data:

1. jscan: a fast visualization program <sup>1)</sup>,
2. ams.inspect: graphic interface for one-dimensional histogramming,
3. Browse: script to examine AMS file contents vis standard Root browser facilities.

Finally, the package includes utility libraries. The available libraries at present are:

1. AMSRoot: the main AMSRoot library,
2. tracking: a library to make and study your own tracks in the AMS tracker,
3. unfold: a library to unfold distributions using different methods,
4. HistoMan: a library to manage histograms during analysis [5].

## 2 Getting started

### 2.1 Downloading AMSRoot

The package can be downloaded as follows:

```
cvs -d :pserver:anonymous@isscvb.cern.ch:/local/repos/amsroot checkout AMSRoot
```

---

<sup>1)</sup>There exists a more sophisticated scanning program, `amsed`, in the `display` directory of the official AMS software repository.

## 2.2 Configuration steps

1. Set the ROOTSYS variable:

- On sh or bash and SLC3 [6], gcc 3.2 or compatible (like pcamsf2 or lxplus):

```
export ROOTSYS=/afs/cern.ch/exp/ams/Offline/root/Linux/slc3
```

- On csh or tcsh and SLC3 [6], gcc 3.2 or compatible (like pcamsf2 or lxplus):

```
setenv ROOTSYS /afs/cern.ch/exp/ams/Offline/root/Linux/slc3
```

- If you have no AFS access, please follow the instructions in the file `$AMSRoot/README_install_Root`, in order to install a suitable working package. Also note that some of the packages may not work with Root versions below v5.

2. To avoid potential problems (like missing libraries at runtime), it is convenient to add `$ROOTSYS/lib` to the path for run-time libraries. For instance:

- On sh or bash:

```
export LD_LIBRARY_PATH=$ROOTSYS/lib
```

- On csh or tcsh:

```
setenv LD_LIBRARY_PATH $ROOTSYS/lib
```

3. It is also comfortable to have the `$AMSRoot` path defined. For instance, if you have downloaded AMSRoot into your `ams` subdirectory:

- On sh or bash:

```
export AMSRoot=$HOME/ams/AMSRoot
```

- On csh or tcsh:

```
setenv AMSRoot $HOME/ams/AMSRoot
```

4. It is a good practice to include all previous definitions in your `.bashrc` (on bash) or `.tcshrc` (on tcsh) shell initialization script. This way things are set automatically every time you open a window terminal.

5. Important Note: whenever the `$ROOTSYS` variable is changed or the Root installation is updated, you **MUST** go to the `$AMSRoot/lib` directory and run:

```
make all_clean
```

## 2.3 Additional details

The present recommended Root version for AMSRoot is v5.

We believe that any computer with a RedHat version  $\geq 7.3$  and access to a gcc version 3.2 is well suited for an AMSRoot installation. The code has been tested on RedHat versions 7.3, 8.0, 9.0 and Scientific Linux CERN (SLC3 [6]).

It is maybe useful to note that many of the official ROOT versions in `/afs/cern.ch/sw/root/...` are perfectly valid for running in a) CINT mode, b) dynamic or c) in tselector modes. Some

sub-packages and modes will not work with versions below v4.01/02: the Python mode and the libraries `unfold` and `tracking` are the examples. Finally, let us note that the `static` mode requires the presence of the `$ROOTSYS/lib/libRoot.a` static library, which is not included in the official Root versions distributed in binary mode.

## 2.4 Building the AMSRoot libraries

The analysis examples in the distribution are prepared to automatically build any missing AMSRoot library. Nevertheless, it is not a bad idea to build libraries beforehand. This can be done as follows:

Command	Effect
cd lib; make all cd lib; make all_shared cd lib; make all_static cd lib; make all_clean	make ALL shared+static libraries build ALL shared libraries build ALL static libraries clean ALL
cd lib; make cd lib; make shared cd lib; make static cd lib; make clean	make shared+static AMSRoot libraries build only the AMSRoot shared library build only the AMSRoot static library clean all AMSRoot stuff
cd lib; make utils cd lib; make utils_shared cd lib; make utils_static cd lib; make utils_clean	make all utility libraries build all utility shared libraries build all utility static libraries clean all utility libraries
cd lib; make XXXXXX cd lib; make XXXXXX_shared cd lib; make XXXXXX_static cd lib; make XXXXXX_clean	make shared+static XXXXXX libraries build only the XXXXXX shared library build only the XXXXXX static library clean all XXXXXX stuff

See `$AMSRoot/lib/README_to_add_a_library` for details on how to add your own library to the AMSRoot distribution.

## 2.5 Simplest examples of usage

---

```
cd $AMSRoot/cint; cat README;
root cint.C;
root select_entries.C;
```

---

```
cd $AMSRoot/dynamic; cat README;
run_dynamic;
```

---

```
cd $AMSRoot/static; cat README;
run_static;
```

---

```
cd $AMSRoot/tselector; cat README;
root run_MySelector1.C;
```

---

```
cd $AMSRoot/python; cat README;
python -i example.py;
```

---

```
cd $AMSRoot/scan; cat README;
make;
jscan ../files/test.root;
```

---

```
cd $AMSRoot/inspect; cat README;
make;
ams_inspect ../files/test.root;
```

---

```
cd $AMSRoot/browser; cat README;
Browse ../files/test.root;
```

---

```
cd $AMSRoot/unfold; cat README;
root test_unfold.C;
```

## 2.6 Documentation on AMS ROOT classes

- The Web way: the latest version of the documentation will be kept on the AMS Web pages. At present, you can find it at:  
<http://ams.cern.ch/AMS/Analysis/hpl3itp1/root02/AMSRoot/>
- The Html-private way: you can build your own HTML pages of documentation in the \$AMSRoot/doc directory. The pages are created with the Doxygen package [7]. If you do not have it on your machine, either install it (‘‘apt-get install doxygen’’ on SLC3) or download it from <http://www.doxygen.org>. To create the documentation do: `cd $AMSRoot/doc; make`. To start reading the documents, start from the index with something like: `mozilla file:$AMSRoot/doc/index.html`.
- The brute force way: read \$AMSRoot/lib/AMSRoot/root.h for the interface of each basic AMSRoot class and \$AMSRoot/lib/XXXXXX/\*.h for the interface of the utility library XXXXXX.

## 3 Running modes

### 3.1 CINT analysis mode

The file `$AMSRoot/cint/cint.C` is an example of program which can be run via the CINT [3] ROOT interpreter. To run it, just do: `cd $AMSRoot/cint; root cint.C`. A simplified version of the program is reproduced here:

---

```
1 {
2     gSystem->Exec("cd $AMSRoot/lib; gmake shared");
3     gSystem->Load("$AMSRoot/lib/libAMSRoot.so");
4
5     AMSChain* ams = new AMSChain;
6     ams->Add("$AMSRoot/files/*.root");
7
8     TFile* hfile = new TFile ("amstest.root", "RECREATE");
9
10    TH1F* hrig = new TH1F ("hrig", "Momentum (GeV)", 50, -10., 10.);
11
12    AMSEventR* pev = NULL;
13    while (pev = ams->GetEvent()) {
14        for (int i=0; i<pev->nParticle(); i++) {
15            ParticleR* part = pev->pParticle(i);
16            hrig->Fill(part->Momentum);
17        }
18    }
19
20    hrig->Draw();
21    hfile->Write();
22
23    printf("We have processed %d events\n", (int)ams->GetEntries());
24    printf("Histograms saved in '%s'\n", hfile->GetName());
25 }
```

File `cint.C`

---

Interpreted analyses are well suited for small or medium size data samples and small pieces of code, for which computer time is not a critical issue. In CINT you may even provide simplified C++ instructions, like:

```
hrig = new TH1F ("hrig", "Momentum (GeV)", 50, -10., 10.);
```

instead of:

```
TH1F* hrig = new TH1F ("hrig", "Momentum (GeV)", 50, -10., 10.);
```

Some additional comments:

- One has to load the specific AMS Root library containing all the ROOT class definitions, class attributes and utilities:

```
gSystem→Exec("cd $AMSRoot/lib; gmake shared");
gSystem→Load("$AMSRoot/lib/libAMSRoot.so").
```

- Root files are added with the `AMSCChain::Add("file")` method. The `AMSCChain` class inherits all methods of the `TChain` class and at the same time hides to the user some complicated operations, specific to the AMS implementation:

```
AMSCChain* ams = new AMSCChain;
ams→Add("$AMSRoot/files/*.root");
ams→Add("rfio:my_castor_file.root");
```

- The analysis starts from `AMSEventR` objects, which can be accessed with the `AMSCChain::GetEvent(...)` methods:

```
AMSEventR* pev = ams.GetEvent()
AMSEventR* pev = ams.GetEvent(entry_number)
AMSEventR* pev = ams.GetEvent(run_number, event_number)
```

- The current event and entry number are always available as:

```
AMSEventR* pev = ams.pEvent(); Long64_t entry = ams.Entry();
```

At initialization their values are `NULL` and `-1`, respectively.

- To redo a loop on the whole chain, you can rewind the data stream using the following:

```
ams.Rewind(); AMSEventR* pev = NULL; while (pev=GetEvent()) {...}
```

- In order to access in a simple way all relevant pieces of information many useful definitions, via pointers and vectors, are implemented. For example, the number of reconstructed particles is simply::

```
int npart = pev→nParticle();
```

The *i*th particle belongs to the class `ParticleR`, and it is accessed as:

```
ParticleR* part = pev→pParticle(i);
```

The full list of AMS classes and attributes can be found from the general documentation (see corresponding subsection above). For instance, in `ParticleR` the member "Momentum" can be accessed as:

```
float mom = pev→pParticle(i)→Momentum;
```

## 3.2 “Dynamic” analysis mode

`$AMSRoot/dynamic/dynamic.C` and `$AMSRoot/dynamic/run_dynamic` constitute a typical set of files for running in dynamic mode. The `run_dynamic` script includes the full compilation+link sequence required to build the executable, and `dynamic.C` the actual C++ code. To run it, just do: `cd $AMSRoot/dynamic; run_dynamic`.

The advantages of the “dynamic” mode are:

- The code is compiled, so execution is fast.
- Since the libraries are requested at run time, the executable is rather small (<100 kbytes in the example). This is ideal if you do not want to fill up your disk space with unnecessary stuff.
- The official ROOT binaries are distributed with “shared” libraries only.

There are also potential disadvantages:

- To run tests on small files and many times an interpreted mode, like CINT or Python, is more convenient.
- Since the executable is dynamic, you can not use it on another Linux machine, unless you are sure that the required run-time libraries exist (test if there are missing libraries with: `ldd dynamic.exe`) and are compatible (this happens when different ROOT versions are used: the best way to check the compatibility is by just running it). This can also be particularly a problem if you are going to submit your analysis to a batch queue.

Here below a simplified version of `dynamic.C` is shown. Note that the sequence of instructions is identical to the one used in the CINT mode, apart from the `include` statements, the existence of a main program and the interactive Root startup (`TRint* app = new TRint("",0,0);, ..., app->Run();`).

---

```
1 #include <root.h>
2 #include "TRint.h"
3 #include "TFile.h"
4 #include "TH1F.h"
5 int main(){
6
7     TRint* app = new TRint("", 0, 0);
8
9     AMSChain* ams = new AMSChain;
10    ams->Add("$AMSRoot/files/*.root");
11
12    TFile* hfile = new TFile ("amstest.root", "RECREATE");
13
14    TH1F* hrig = new TH1F ("hrig", "Momentum (GeV)", 50, -10., 10.);
15
16    AMSEventR* pev = NULL;
17    while (pev=ams->GetEvent()) {
18        for (int i=0; i<pev->nParticle(); i++) {
```

```

19         ParticleR* part = pev->pParticle(i);
20         hrig->Fill(part->Momentum);
21     }
22 }
23
24 hrig->Draw();
25 hfile->Write();
26
27 printf("We have processed %d events\n", (int)ams->GetEntries());
28 printf("Histograms saved in '%s'\n", hfile->GetName());
29
30 app->Run();
31 }

```

File dynamic.C

---

### 3.3 “Static” analysis mode

`$AMSRoot/static/static.C` and `$AMSRoot/static/run_static` constitute a typical set of files to run in `static` mode. The script `run_static` contains the full compilation+link sequence required to build a fully static executable, and `static.C` the actual C++ code. To run it, just do: `cd $AMSRoot/static; run_static`.

The advantages of the `static` mode are:

- The code is compiled, so execution is fast.
- Since the executable is static, you can create it anywhere, copy it to any other Linux machine and execute it as it is (no need of any machine-specific libraries at run time).
- It is an ideal mode for batch jobs, for which some run-time libraries might be unavailable.

There are also potential disadvantages:

- To run tests on small files and many times an interpreted mode, like CINT or Python is more convenient.
- The price to pay for a static executable is its size (many Mbytes). If disk space is an issue, linking with shared libraries (used at run-time), like in the `dynamic` mode is preferred. There are also some annoying features, like some warnings at link time.
- The official ROOT binaries are distributed only with “shared” libraries. This means that, in order to run in “static” mode, you’ll need ROOT static libraries installed. See `$AMSRoot/README_install_Root` for details.
- The static ROOT library does not have graphic support. Therefore, you can process, analyze and create histograms in `static` mode, but not visualize them simultaneously (do it afterwards in an interactive ROOT session, for instance).

Here below a simplified version of `static.C` is shown. The sequence of instructions is identical to the one used in the `dynamic` mode, apart from the absence of `TRint` or drawing statements.

---

```
1 #include <root.h>
2 #include "TFile.h"
3 #include "TH1F.h"
4
5 int main(){
6
7     AMSChain* ams = new AMSChain;
8     ams->Add("$AMSRooT/files/*.root");
9
10    TFile* hfile = new TFile ("amstest.root", "RECREATE");
11
12    TH1F* hrig = new TH1F ("hrig", "Momentum (GeV)", 50, -10., 10.);
13
14    AMSEventR* pev = NULL;
15    while (pev = ams->GetEvent()) {
16        if (pev==NULL) break;
17        for (int i=0; i<pev->nParticle(); i++) {
18            ParticleR* part = pev->pParticle(i);
19            hrig->Fill(part->Momentum);
20        }
21    }
22
23    hfile->Write();
24
25    printf("We have processed %d events\n", (int)ams->GetEntries());
26    printf("Histograms saved in '%s'\n", hfile->GetName());
27 }
```

File `static.C`

---

### 3.4 “TSelector” analysis mode

This mode is the suggested way to analyze large samples of AMS data in parallel, using the PROOF facility [8] (although PROOF is not publically released yet). It implies using two programs: a steering program and an analysis program. An example of steering program is `$AMSRooT/tselector/run_MySelector1.C`, which will initialize some ROOT stuff, define input and output files and process (compile+link+execute) the actual analysis program, `$AMSRooT/tselector/MySelector1.C`, containing the real stuff. Everything works together by doing: `cd $AMSRooT/tselector; root run_MySelector1.C`. If one wants to run in non-graphic mode and exit right after execution, use: `root -b -q run_MySelector1.C`.

The advantages of this approach are:

- Particularly for large data samples, a compiled analysis is mandatory.

- The approach will allow in the future to run your program in parallel on several processors (Parallel ROOT Facility - PROOF), with the consequent reduction in time.

The TSelector class [9] is used by the method TTree::Process in order to loop over TTree events and to make selections. TSelector contains the following virtual methods, to be filled by the user:

Method	Action
Init	Attach a new TTree during the loop.
Begin	Called every time a loop on the tree(s) starts. A convenient place to create your histograms.
Notify	Called the first time that a tree is read.
ProcessCut	Called at the beginning of each entry. Returns <code>true</code> if the entry is accepted.
ProcessFill	Called in the entry loop for all entries accepted by ProcessCut.
Terminate	Called at the end of a loop on a Tree. A convenient place to draw/fit your histograms.

The relevant features of the AMS implementation are:

- In the AMS framework, it is enough to create a class which inherits from AMSEventR and then implement the methods:

Method	Action
UBegin()	Called every time a loop on the tree(s) starts. A convenient place to create your histograms.
UProcessCut()	Called at the beginning of each entry. Returns <code>true</code> if the entry must be analyzed.
UProcessFill()	Called in the entry loop for all entries accepted by UProcessCut.
UTerminate()	Called at the end of a loop on a Tree. A convenient place to draw/fit your histograms.

- Note that the name MySelector1 is not mandatory. Everything works equally well if you change MySelector1 by anything. For instance, you may want to write `my_anal.C`, `class my_anal` and `chain.Process("my_anal.C+")` where appropriate.

Here below a simplified version of MySelector1.C is shown. Note that, although the output of the code is equivalent to the one obtained with the methods of the previous sections, the logic is totally different. Note also that the absence of pointers to the event in the code (everything is done inside the AMSEventR class itself):

---

```

1 #include <root.h>
2 #include "TFile.h"
3 #include "TH1.h"
4
5 class MySelector1 : public AMSEventR {
6 public :
```

```

7  MySelector1(){};
8  ~MySelector1(){};
9
10 void    UBegin();
11 Bool_t  UProcessCut();
12 void    UProcessFill();
13 void    UTerminate();
14
15 TFile*  h_file;
16 TH1F*  h_rig;
17
18 };
19
20 void MySelector1::UBegin(){
21   h_file = new TFile("amstest.root","RECREATE");
22
23   h_rig = new TH1F ("h_rig", "Momentum (GeV)", 50, -10., 10.);
24 }
25
26 Bool_t MySelector1::UProcessCut(){
27   return true;
28 }
29
30 void MySelector1::UProcessFill() {
31   for (int i=0; i<nParticle(); i++) {
32     ParticleR* part = pParticle(i);
33     h_rig->Fill(part->Momentum);
34   }
35 }
36
37 void MySelector1::UTerminate() {
38   h_rig->Draw();
39   h_file->Write();
40
41   printf("We have processed %d events\n", (int)Tree()->GetEntries());
42   printf("Histograms saved in '%s'\n", h_file->GetName());
43 }

```

File MySelector1.C

---

### 3.5 “Python” analysis mode

Python [4] is an interpreted, interactive, object-oriented programming language. It is often compared to Tcl, Perl, Scheme or Java.

The file `$AMSRoot/example.py` shows an example of a Python program which can be processed interactively as: `python -i example.py`. If you only need to execute the script and exit (which implies no graphics and no extra Python commands at the end), you may just type: `example.py`.

You can find simple introductions to Python in: <http://www.python.org/doc/Intros.html>. Python is naturally available in ROOT via PyROOT [10].

Please note that PyROOT is a rather new tool in ROOT. Specifically, the Python mode has been tested to work only with ROOT versions  $\geq 4.01$ . You could also find some problems or inconsistencies with it, compared to other better established products like CINT [3]. Finally, remember that interactive, interpreted analyses are well suited for small or medium size data samples and simple analyses, for which computer time is not a critical issue.

Some AMS/PyROOT specific issues follow:

- All necessary initializations concerning AMS classes are performed by the script `$AMSRoot/python/ams_utils.py`. The script can be easily rebuilt with `$AMSRoot/python/create_ams_utils.py`. It is enough to start your script with the line:

```
import ams_utils
```

See the `$AMSRoot/python/README` file for more details.

- The total number of entries in the chain "ams" can be determined as:

```
ndata = len(ams)
```

- You can loop over events and particles as follows:

```
for ev in ams:
    for part in ev.ParticleList():
        ...
```

Here below a simplified version of `example.py` is shown:

---

```
1#!/usr/bin/env python
2import ams_utils
3from ROOT import AMSChain, TFile, TH1F
4
5ams = AMSChain()
6ams.Add("$AMSRoot/files/*.root")
7
8hfile = TFile("amstest.root", "RECREATE")
9
10hrig = TH1F("hrig", "Momentum (GeV)", 50, -10.0, 10.0)
11
12for ev in ams:
13    for part in ev.ParticleList():
14        hrig.Fill(part.Momentum)
15
16hrig.Draw()
17hfile.Write()
18
19print "We have processed %d events" % len(ams)
20print "Histograms saved in '%s'" % hfile.GetName()
```

File `example.py`

---

## 4 Selection lists. Writing selected events.

The utility class `AMSEventList` contains methods to read and write event lists. In addition, the method `AMSEventList::Write(TTree* chain, TFile* output_file)` allows direct writing of the selected events into `output_file`. The example in `$AMSR00t/cint/select_entries.C`, approximately reproduced below, shows the main possibilities. To run it just do:

`cd $AMSR00t/cint; root select_entries.C`. Note that it is rather trivial to write just few branches of the original chain, allowing some disk space saving if necessary.

For more details, please consult the general documentation or the `AMSEventList` interface in `$AMSR00t/lib/AMSR00t/root.h`.

---

```
1 {
2 // Resetting and loading AMS library
3     gSystem->Exec("cd $AMSR00t/lib; gmake shared");
4     gSystem->Load("$AMSR00t/lib/libAMSR00t.so");
5
6 // Input AMS ROOT Chain
7     AMSChain *ams = new AMSChain;
8     ams->Add("$AMSR00t/files/*.root");
9
10 // Initialize selection list
11     AMSEventList list;
12 // Alternative if you have already a list
13     //AMSEventList other_list("amstest.list");
14
15 // Loop to analyze entries
16     int ndata = ams->GetEntries();
17     for (int entry=0; entry<ndata; entry++) {
18         AMSEventR* pev = ams->GetEvent();
19         if (pev==NULL) break;
20
21 // Examples of use, mainly for previously created lists
22         //if (other_list.Contains(pev)) continue;
23         //if (pev->nBeta(>1) { other_list.Remove(pev);}
24
25 // Typical example: add to list if it satisfies your requirements
26         if (pev->nBeta()==1) {
27             BetaR beta = pev->Beta(0);
28             if (fabs(beta.Beta-1.)>beta.Error) list.Add(pev);
29         }
30     }
31
32     // BEGIN of block: How to write just few branches ->
33     ams->SetBranchStatus("*",0); // this disables all branches by default
34     ams->SetBranchStatus("ev.fHeader",1); // HeaderR class information IN
35     ams->SetBranchStatus("ev.fParticle",1); // ParticleR class information IN
36     // END of block
37
38 // Write selected events from chain to a new AMS ROOT file
39     TFile* out_file = new TFile("amstest.root","RECREATE");
```

```

40     list.Write(ams,out_file);
41
42 //Write "run event" list to standard output
43     //list.Write();
44
45 //Write "run event" list to ASCII file
46     //list.Write("amstest.list");
47
48     cout << "Events in the old tree: " << ndata << endl;
49     cout << "Events in the new tree: " << list->GetEntries() << endl;
50 }

```

File select\_entries.C

---

## 5 Utilities for a fast look to AMS data

### 5.1 JScan program

The JScan utility allows a simple and fast visualization of AMS events. It can be found in the `$AMSRoot/scan` directory. The information here is an extract of what is written in the file `$AMSRoot/scan/README`.

To build the program, just do: `cd $AMSRoot/scan; make`. To run it, do:  
`jscan "AMS_Root_file(s)".` Examples:

```

jscan                               (will open a dialog to look for a file)
jscan $AMSRoot/files/test.root
jscan $AMSRoot/files/*.root

```

The program is still in a development phase. We are using some conventions:

- Colors for `TrTrackR` and `ParticleR` objects follow the GEANT4 default convention:

Color	Meaning
blue	positive charge
red	negative charge
green	neutral particle

- TOF and ANTI-coincidence clusters are shown in green. TRD clusters, TRD tracks, RICH hits at the photomultiplier plane and ECAL clusters not belonging to any shower are shown in cyan.
- The initial MC particle (if any) is shown as a dotted yellow line.
- TRD hits are assumed to have an uncertainty of the order of the TRD tube radius in both projections whenever it belongs to a reconstructed TRD track. Otherwise, one of the hits will be shown with an uncertainty of the order of the tube length in the non-measuring projection.

- Photon rays in RICH are shown with a given color (but not cyan), characterizing the RICH ring.
- Hits inside ECAL showers are shown with a given color (but not cyan), characterizing the shower.
- Passing the mouse over most tracks or hits in the picture triggers the writing of information about the object at the bottom pad.
- Clicking with the mouse over a track or hit in the picture freezes the information, so you can take your time to read what is written without caring about the mouse. This information is cleared when you pass the mouse over a different object.
- When a hit inside an ECAL shower is clicked, the corresponding shower information is written in the bottom pad. When a hit inside an ECAL not belonging to any shower is clicked, the hit information is written instead.

## 5.2 AMS\_Inspect program

The `ams_inspect` utility allows a simple and fast visualization of AMS file contents via one-dimensional histograms. It can be found in the `$AMSRoot/inspect` directory. The information here is an extract of what is written in the file `$AMSRoot/inspect/README`.

To build the program, just do: `cd $AMSRoot/inspect; make`. To run it, do: `ams_inspect "AMS_Root_file(s)"`. Examples:

```
ams_inspect                                (will open a dialog to look for a file)
ams_inspect $AMSRoot/files/test.root
ams_inspect $AMSRoot/files/*.root
```

## 5.3 Browser utilities

The `Browse` utility in the `$AMSRoot/browse` directory is a script to access and histogram the contents of AMS Root files via standard Root facilities like `TBrowser` and `TTreViewer`. The information here is an extract of what is written in the file `$AMSRoot/browse/README`. To run the script, do: `cd $AMSRoot/browse; Browse "any_Root_file(s)"`.

Some explanations concerning this utility:

- It only works with recent ROOT versions (from v4 on).
- It creates a temporary browsable root file on `/tmp`. So you should have enough disk space in order to store it.
- The idea is simple: the new root file is identical to the original one, but with split level 2 (see Root manual for details on the concept of `split level`). As a consequence, the whole data structure has to be read. The extra time involved is irrelevant for small root files, but it may be non negligible for production ones (it depends on the computer).
- The temporary file is deleted afterwards, so you may want to comment the last line in the `Browse` script to avoid deletion. The temporary file can be browsed directly with the script `browse_split_file.C`.
- STL vectors beyond the first split level are not browsable. Example: the array of `TrRecHits` inside `TrTracks`.

## 6 Libraries

Adding a new library to the AMSRoot package is a simple operation in most cases. For a given library called XXXXXX, the procedure essentially consists in creating a subdirectory XXXXXX, putting there your C++ and header files, and copying (almost without modifications) a couple of additional files. More detailed information can be found in `$AMSRoot/lib/README_to_add_a_library`.

Besides the general AMSRoot library, sitting in `$AMSRoot/lib/AMSRoot/`, only few libraries are included in the package at present. They are discussed in the following.

### 6.1 “Tracking” library

It is sitting in `$AMSRoot/lib/tracking/`. An example of usage, quoted below, is the file `$AMSRoot/tracking/my_tracks.C`. The central class of the library, `MyTrack`, allows the user to create his own tracks, adding or deleting hits (via the auxiliary class `MyHit`) at will, and fitting with or without magnetic field. In addition, the methods `MyTrack::Prediction(...)` and `MyTrack::PredictionStraightLine(...)` can be used in order to get the tracking vector information (`x,y,z,theta,phi,dx,dy`) close to any hit, using the rest of the track points.

For more details, please consult the general documentation or the `MyTrack` interface in `$AMSRoot/lib/tracking/tracking.h`.

---

```
1 {
2 // Resetting and loading AMS library
3     gROOT->Reset();
4     gSystem->Exec("cd $AMSRoot/lib; gmake shared");
5     gSystem->Exec("cd $AMSRoot/lib; gmake tracking_shared");
6     gSystem->Load("$AMSRoot/lib/libAMSRoot.so");
7     gSystem->Load("$AMSRoot/lib/libtracking.so");
8
9 // Input AMS ROOT Chain
10    AMSChain *ams = new AMSChain;
11    ams->Add("$AMSRoot/files/test.root");
12
13 // Output file and histograms
14    char* chfile = "amstest.root";
15    TFile* hfile = new TFile (chfile, "RECREATE");
16    TH1F* hrig = new TH1F ("hrig", "Rigidity change (GV)", 50, -1., 1.);
17
18 // Loop to analyze entries
19    int ndata = ams->GetEntries();
20    for (int entry=0; entry<ndata; entry++) {
21        AMSEventR* pev = ams->GetEvent();
22        if (pev==NULL) break;
23        for (int i=0; i<pev->nTrTrack(); i++) {
24            TrTrackR* ptrk = pev->pTrTrack(i);
25 // Instantiate your track
26            MyTrack my_track(ptrk);
27
28 // Remove some hits (just to use some additional functions)
```

```

29         for (int j=5; j<my_track.NHits(); j++) {
30             my_track.del_hit(j);
31         }
32         if (ptrk->NTrRecHit(>5) {
33             double* pred = my_track.Prediction(ptrk->pTrRecHit(5));
34             if (pred) printf ("X pred %f, measured %f\n"
35                 , pred[0], ptrk->pTrRecHit(5)->Hit[0]);
36         }
37
38         int stat = my_track.Fit();
39         if (stat==0) {
40             hrig->Fill(my_track.Rigidity-ptrk->RigidityWithoutMS);
41             //printf("Track number= %d, Old rig %f, New Rig %f\n"
42                 //      , i, ptrk->RigidityWithoutMS, my_track.Rigidity);
43         }
44     }
45 }
46
47 hrig->Draw();
48 hfile->Write();
49 printf("We have processed %d events\n", ndata);
50 printf("Histograms saved in '%s'\n", chfile);
51 }

```

File my\_tracks.C

---

## 6.2 “Unfold” library

This library contains several methods for distribution unfolding. The central class of the library, `AMSUnfold`, has the method `Unfold(int method)`, which triggers the unfolding procedure. Detailed instructions and a short explanation of the methods can be found in `$AMSRoot/unfold/README`.

An example of usage, quoted below, is the file `$AMSRoot/unfold/test_unfold.C`. To run it, use: `cd $AMSRoot/unfold; root test_unfold.C`.

---

```

1 {
2     gSystem->Exec("cd $AMSRoot/lib; gmake unfold_shared");
3     gSystem->Load("$AMSRoot/lib/libunfold.so");
4     gStyle->SetOptStat(0);
5
6     TFile* hfile = new TFile ("unfold_input_10000.root");
7
8     TH1F* hdata = (TH1F*)hfile->Get("hdata");
9     TH2F* hmigr = (TH2F*)hfile->Get("hmigr");
10    TH1F* hbkg = (TH1F*)hfile->Get("hbkg");
11    TH1F* hgen[5] = {NULL};
12    TH1F* htrue_binning = NULL;
13
14    AMSUnfold* unfold = new AMSUnfold();

```

```

15  unfold->SetData(hdata);
16  unfold->SetMigrationMatrix(hmigr);
17  unfold->SetBackground(hbkg);
18
19  TH1F* htest = unfold->GetTest();
20  TH1F* hdata_nobkg = (TH1F*)hdata->Clone();
21  hdata_nobkg->Add(hdata,hbkg,+1.,-1.);
22
23  char* chmethod[5] = {
24      "Exact Inversion",
25      "Bayesian Unfolding",
26      "Tikhonov regularization",
27      "CDelgado regularization",
28      "JAlcaraz unfolding"};
29  /// Unfold using a given method
30  /// 0: exact inversion
31  /// 1: Bayesian (also called D'agostini) method
32  /// 2: Tikhonov regularization from SVD decomposition and relevant #d.o.f.
33  /// 3: C. Delgado regularization
34  /// 4: J. Alcaraz unfolding
35  TCanvas* c1 = new TCanvas("c1","Unfolding tests",1024.,768.);
36  c1->Divide(2,2);
37  for (int method=1; method<=4; method++) {
38      c1->cd(method);
39      TH1F* hgen_tmp = unfold->Unfold(method);
40      hgen[method] = (TH1F*)hgen_tmp->Clone();
41      if (method==3) htrue_binning = unfold->>true_binning();
42
43      if (method==3) {
44          htrue_binning->SetMarkerStyle(20);
45          htrue_binning->Draw("e0");
46      } else {
47          hgen[method]->SetMarkerStyle(20);
48          hgen[method]->Draw("e0");
49      }
50
51      htest->Draw("histsame");
52      htest->SetXTitle("Unfolded variable");
53
54      hdata_nobkg->SetLineStyle(2);
55      hdata_nobkg->Draw("histsame");
56
57      TText *t = new TText();
58      t->SetTextSize(0.04);
59      t->SetTextAlign(12);
60      t->DrawText(0.15,1600,chmethod[method]);
61      t->DrawText(0.15,1400,"Theory (predicted)");
62      t->DrawText(0.15,1200,"Original data distribution");
63
64      TGraph *graph = new TGraph(1);

```

```

65         graph->SetMarkerStyle(20);
66         graph->SetMarkerSize(1);
67         graph->SetPoint(0,0.1,1600.);
68         graph->Draw("P");
69
70         TLine *line = new TLine(0.08,1400.,0.12,1400.);
71         line->SetLineStyle(1);
72         line->SetLineWidth(2);
73         line->Draw();
74
75         TLine *line2 = new TLine(0.08,1200.,0.12,1200.);
76         line2->SetLineStyle(2);
77         line2->SetLineWidth(2);
78         line2->Draw();
79     }
80 }

```

File test\_unfold.C

---

### 6.3 “HistoMan” library

The Perugia Tool library, implemented as the HistoMan library, consists in a set of utilities facilitating the task of histogramming when analyzing data. This is of particular importance when different sets of cuts have to be tried and one has to study, organize and keep track of all possibilities.

The command sequence `$AMSRoot/HistoMan/src; make doc` makes the documentation for the HistoMan package. It is created under the directory `$AMSRoot/HistoMan/doc`. A full description of the package is out of the scope of this Note. See AMS Note [5] for further details.

## References

- [1] <http://root.cern.ch>.
- [2] <http://root.cern.ch/root/html/TTree.html#TTree:description>.
- [3] <http://root.cern.ch/root/Cint.html>.
- [4] <http://www.python.org/>.
- [5] P. Zuccon and D. Caraffini, AMS Internal Note 2005-04-01.
- [6] <http://linux.web.cern.ch/linux/scientific3/>.
- [7] <http://www.doxygen.org/>.
- [8] <http://root.cern.ch/root/PROOF.html>.
- [9] <http://root.cern.ch/root/html/TSelector.html#TSelector:description>.
- [10] <http://cern.ch/wlav/pyroot/>.