Calib:
a package for MDT calibration studies - User Manual

A.Baroncelli, M.Iodice, D.Orestano

INFN Sezione di Roma III and
Dipartimento di Fisica Università degli Studi di Roma Tre

P.Bagnaia, L.Pontecorvo

INFN Sezione di Roma and
Dipartimento di Fisica Università degli Studi di Roma "La Sapienza"

ATL-MUON-2005-013
17 January 2005

Abstract

This note describes the *calib* package, developed for the optimization of the MDT calibration procedures but also used as a general MDT reconstruction and performance evaluation tool. This package has been used to analyze data both from Roma Tre cosmic test and from H8 test beam. The structure of the program is presented together with a simplified user guide.

1 Introduction

The MDT (monitored drift tubes) calibration procedures, and in particular the possibility of autocalibration, have been studied extensively in the framework of ATLAS muon activities. The experience gained from the analysis of test beam data [1] and simulations [2] lead to the development of a software package called *calib* , used in the studies of data collected in H8 [3] and in Roma Tre cosmic ray test stand, but flexible enough to accommodate any simple geometry. The *calib* framework provides the essential steps in the analysis of MDT information: data decoding, conversion of TDC counts into space information, pattern recognition and track fit without magnetic field, and allows the insertion of user analysis tasks. The most used user tasks developed for the H8 test beam data analysis are included in the distribution of the code to be used as a starting point for new users.

2 Program description

The *calib* program is a stand-alone package, based only on few standard external libraries: STL [4], CLHEP [5], HBOOK/CERNLIB [6]. It is organized in sub-packages in order to be modular and, hopefully, to simplify its migration to the official ATLAS framework in a near future. Each sub-package is represented by a sub-directory in the calib top directory.

The following functionalities have been identified and used in the domain decomposition to define the sub-packages:

Data flow control associated to sub-package *control* .

Handling of database information associated to sub-package *database* .

Detector description associated to sub-package *detector*

Handling of events and different data formats associated to sub-package *event*

Pattern recognition and track fit associated to sub-package *recon*

User analysis tasks collected in sub-package *user*

Statistical analysis associated to sub-package *his*

Tools of various type associated to sub-package *tools*

Handling of job options, performed by each user task for the task specific options and attributed to *control* for the global options.

3 Framework

3.1 Sub-package *control*

The *calib* framework is based upon the class *CalibrationManager* in the sub-package *control* . *CalibrationManager* is a singleton instantiated in the main program (*calib.cxx*) whose tasks are

- load the global job options from an external file, *calib.datacards* , and provide access to them through public methods from all the program classes. Global job options include the input data format, the number of events to be processed, skipped, displayed etc... The contents of file *calib.datacards* will be described in detail in paragraph 6.2.1.
- load from *calib.datacards* the sequence of user tasks to be processed and handle in its *CalibrationManager::execute* method, a loop over the tasks, within which an even loop is performed for each task. *CalibrationManager::eventLoop* loops over the full data set in input as many times as requested by the user task.

It is important to underline that for each task a separate loop over the data is performed, and for some tasks more than a loop is required. This flexibility, allowing multiple iterations over the same data, was the main requirement which lead to the development of a stand-alone framework. Some changes in the ATLAS framework will be required in order to implement a similar functionality.

In its constructor *CalibrationManager* instantiates all the singletons used to store data of different types:

- the *GeometryHandler* in sub-package *database* for the access to the geometry (cf. 3.2);
- the *RTRelHandler* in sub-package *database* for the access to the space-time relation and to the resolution function (cf. 3.2);
- the *T0Handler* in sub-package *database* for the access to the T0 value of each MDT tube (cf. 3.2);
- one of the classes inheriting from *EventHandler* in sub-package *event* (cf. 3.3 and 6.2.3) for the manipulation of events;
- the *HisFile* in sub-package *his* for the booking and storage of histograms (cf. 3.4).

3.2 Sub-package *database*

The *GeometryHandler* keeps the list (in the form of STL standard vectors) of the pointers to the basic geometrical detector objects. For MDT the basic object is a *GeoMDTMultiLayer* , for RPC the basic object is a *GeoRPCStripPlane* , both belonging to sub-package *database* .

The constructor invokes the *GeometryHandler::load* method, which looks for a file named *geofiles* in the top directory and loops over the files listed in *geofiles* (cf. 6.2.2) and loads their contents into the basic geometrical objects.

The geometrical description provided by *GeoMDTMultiLayer* and *GeoRPC-StripPlane* is a tree going down to the single MDT tube (*GeoMDTTube*) and single RPC strip (*GeoRPCStrip*). A parallel structure is used to store the event data and links allow to cross-navigate through the two structures. Accessing the geometry of a tube or a strip allows also to retrieve through appropriate links the electronics and calibration information associated to the tube or strip. These links are updated at the beginning of each new run by a call to *GeometryHandler::updateLinks* in *CalibrationManager::runInit()* .

The *RTRelHandler* keeps a list of space-time relations *RTRel* . The space-time relations are given by points in a set of files listed in file *rtfiles* (cf. 6.2.2) in the top directory. Each space-time relation is valid for a set of tubes listed inside the file and for the tubes portions selected in the file header.

The *T0Handler* keeps a list of *T0File* , where each *T0File* contains the T0 values for a list of tubes. The *T0Handler* looks for file *t0files* (cf. 6.2.2) in the top directory and loads the contents of the files listed in it.

3.3 Sub-package *event*

Events are read-in from a list of input files taken from top directory file *input* and are decoded by specific objects inheriting from EventHandler to implement the decoding of different data formats. These objects, like *DAQEvent* for H8 DAQ-1 data (cf. 6.2.3), are instantiated by *CalibrationManager* accordingly to the value of data card DataFormat in *calib.datacards* . Some of these Event objects use in the decoding the mapping of electronic channels to tubes provided by class *EltxHandler* in sub-package *database* .

The class *EventHandler* , in sub-package *event* , is a singleton used to store a list of pointers to detector specific *SubEvent* objects (like *MDTMultiLayer* and *RPC-StripPlane* in sub-package *detector*). The pointer to the specific Event object can be accessed by all other objects using the *EventHandler::getEventHandler* method. All the hits can be retrieved navigating through the *SubEvent* list stored in the Event object.

3.4 Sub-package *his*

The interface towards the histogramming package (actually HBOOK) is provided by classes *HisFile* and *Histo* , in which all the standard operations upon histograms have been implemented through the *hbook.h* cfortran file from CERNLIB. This file had been copied locally in subdirectory *his* to get rid of the problems due to few differences found between different CERNLIB releases.

3.5 Sub-package *tools*

All the header files in *calib* must include *tools/Utils.h* which provides a couple of precompiler *define*s and the inclusion of *MemoryChecker.h* . *MemoryChecker* is a singleton used to keep track of the number of objects created and deleted all over the program. Developers should invoke *MemoryChecker::increase* in all the object constructors and *MemoryChecker::decrease* in all the object destructors. At the end of calib execution a check is performed on the number of objects still present and an error message is issued in case of mismatch. In presence of a memory leak users can call *MemoryChecker::objects* anywhere in the program to find out whether there are pending links or attempts to delete non-existing objects.

4 Reconstruction

The sub-package *recon* contains the classes handling the event reconstruction. The objects involved in MDT reconstruction are

- the *MDTReconstructor*
- the *PRDetector*
- the *Track*

while

- *RPCReconstructor*
- *RPCTrack*

must be developed for RPCs reconstruction.

Finally within *recon* a straight track generator, *Generator* has been implemented to help in development and debugging of reconstruction, as well as to cross check the autocalibration, the resolution and the efficiency studies techniques (cf. 5.8).

4.1 MDTReconstructor

The pattern recognition acts on one or more *GeoMDTMultiLayers* (sub-detectors) forming a *PRDetector* element, accordingly to what specified in the datacards. A *MDTReconstructor* object contains the hit list of the current event for a *PRDetector* . On this MDT hit list a pattern recognition method (still belonging to the MDTReconstructor class) is applied to search for tracks. As a general rule, a straight track is found when the reduced chi squared of the hit residuals is below the value specified in the task datacard file.

A *checkAmbiguities* method is also implemented to identify possible ambiguities on the reconstructed tracks: two different tracks can share a maximum number of hits specified in the datacards.

4.2 PRDetector

The class name comes from "Pattern Recognition Detector" element. A PRDetector element is usually created when a *taskInit* method is invoked within a task where tracks reconstruction is required.

More than one *PRDetector* can be constructed. In this case they will be treated as different elements on which tracks are independently searched for (as for example in the *CalAlign* alignment task).

At *PRDetector* initialization a list of possible MDT tube patterns, for tracks generated within the angular acceptance specified on the datacards file, is produced by the *buildCandidates* method. The list of the produced *CandidateTracks* (consisting in a list of tube identifiers) is kept in memory to be used as a look-up table by the pattern recognition algorithm. A single "reference pattern" is considered for all those patterns differing only by a simple increment of tube number, when the increment is the same for all layers (translation along z of the chamber).

The main method for track search purposes is *matchCandidate* allowing to match the current event MDT hit list with the tube patterns listed in the look-up table. Once a pattern is matched, a *fitTrack* method of the *Track* class is called to define the track parameters and to check whether the track is acceptable (reduced chi squared less than a threshold) or not. If accepted, the hits are removed from the hit list and other tracks are searched for in the updated hit list. If not accepted, the same hits can later be assigned to an incomplete track, when one or more (bad) hits are removed from the list. The method first scans all the *CandidateTracks* of the look-up table to find "complete" tracks (number of hits = number of layers of the *PRDetector*). The list is scanned again to look for tracks with missing hits.

Finally, it is worth mentioning that a method *mergeTracks* has been implemented to properly handle those tracks at large angles (about 30 degrees w.r.t. the normal to the chamber) which can give hits on two adjacent tubes in the same layer (configurations not included in the look-up table).

4.3 Track

The minimal content of a track is a list of aligned MDT hits choosen as described above. The method *fitTrack* is used to find the best straight line tangent to the

circles corresponding to the hit drift radii. At present two methods are available to find the best slope and intercept of the straight line:

- a minimization procedure on the residual chi squared function, which makes use of the MINUIT package [7],
- an analytic method which finds all the tangent lines to a pair of subsequent hits and follows these tangents to the next pair, finally merging those closer in slopes and intercepts.

Tests performed show there are no significant differences between the two methods.

5 User tasks

User tasks, collected in sub-package *user* , are objects derived from *CalibrationTask* in *control* . The user should implement the following methods:

- *void taskInit()* , called at the beginning of the task execution;
- *bool taskEnd()* , called at the end of the task execution, it returns true when the task is completed, and false when an additional iteration over the data sample is needed;
- *void runInit()* , called at each new run;
- *void runEnd()* , called at the end of a run;
- *void evtProc()* , called for each event.

In addition user tasks can define their own set of datacards to set task dependent parameters.

Tasks already developed are described in this section.

5.1 *CalT0Det*

The user task *CalT0Det* builds the TDC counts spectrum histogram of each MDT tube. From the datacards it is possible to change the histograms binning and to decide whether to fit the distributions. At the end of the unique iteration performed by this task an output file (*newt0.dat*) with the fitted T0 of each tube is produced.

Be aware that the parameter called T0 above is just one of the parameters obtained from the fit corresponding approximatively to the center of the rising slope of the spectrum. The effective time for zero drift distance must be obtained subtracting a constant from it, the same for all the tubes. The constant is defined by the datacard T0offset in *calib.datacards* **.**

This same task can provide in output a rough space-time relation (*newrt.dat*) obtained from the integration of the TDC spectrum for a given tube, selected in *CalT0Det.datacards* through the datacard **tubeSel**.

5.2 *CalAutoCal*

The user task *CalAutoCal* is the most important *calib* task. Starting from an input space-time relation (obtained from the integration of the TDC spectrum, from a simulation or from the analysis of another data set) it reconstructs tracks in the full data sample and accumulates in a bi-dimensional histogram the residuals between each drift circle attributed to the track and the distance of minimum approach of the fitted track to the tube center, as a function of the measured drift time. In *CalAutoCal::taskEnd* the information retrieved from the histogram is used to correct the input space-time relation so that a new one can be used in the next iteration. *CalAutoCal::taskEnd* stops the process returning true when a predefined maximum number of iterations is reached but can be modified to stop when the space-time relation of one iteration is equal within errors to the previous one. The value of the correction can be extracted from the histograms either by taking the average residual in a drift time bin as computed by HBOOK through a profile histogram or by taking the mean of a Gaussian fitted to the data in a drift time slice. The selection between the two methods is done through the datacard **defResidual** in *CalAutoCal.datacards* to be set to 1 or 2 respectively.

CalAutoCal can be run on a given data sample once the T0s have been determined (and organized in files listed in *t0files*), starting from a first approximation space-time relation provided through the files listed in *rtfiles* . In output *CalAutoCal* produces the new space-time relation saved to file *newrt.dat* .

5.3 *CalResol*

Task *CalResol* reconstructs tracks using the correct T0 values and the best space-time relation and computes the MDT tubes resolution. For each space bin, the standard deviation of the track residual distribution on a plane excluded from the track fit is computed. The resolution is obtained from this value after subtraction, in quadrature, of the extrapolation error at that layer. Resolution is computed separately for positive and negative residuals and the final result is given by the average value between these two data sets. To reduce the dependence of the result from the resolution assumed in the track fit the task can iterate over the data set more than once. Typically two iterations are used. The final resolution is saved to *newres.dat* together with the space-time relation.

5.4 *CalAlign*

Given the correct T0 values, the space-time relations and resolution functions for two sub-detectors or sets of sub-detectors (for example two multilayers from the same MDT chamber or two different MDT chambers) this task reconstructs separately two track segments in the two detectors and compares them, thus allowing the user to check their relative alignment and possibly correct it by acting on the geometry files.

5.5 *CalRecon*

This task simply uses the correct T0 values, the best space-time relation and the resolution, to reconstruct tracks in the MDTs.

5.6 *CalMonitor*

This task is an example of a higher level task, which for each event calls the *evtProc* methods of *CalT0Det* and *CalRecon* and updates the histograms regularly, with a frequency defined through the task datacards.

5.7 *CalEffi*

This task evaluates the global efficiency and efficiency as a function of the radius of a number of MDT tubes listed in the *CalEffi.datacards* file. The determination of the efficiency for each tube is performed by searching for complete tracks (number of hits = number of layers) or for tracks with just one missing hit. If the tube under investigation is included in the complete track, fulfilling a chi-squared cut, then it is counted as "efficient". If for an incomplete track the missing tube is the one under analysis, it is considered as "inefficient". However it can be "recovered" if its residual is less than $n\sigma$, σ being the space resolution of the tube at the hit radius and n a number to be specified in the *CalEffi.datacards* file.

The final efficiency results are not directly produced by the *CalEffi* task but the PAW [8] macro *caleffi.kumac* should be run on calib output.

5.8 *CalRayTrace*

This task invokes a straight track generator to produce simple simulated events and saves them in the ASCII format compatible with class *PBEvent* . Tracks are generated in the PRDetector element specified in the datacard file with a given angular spread.

For each firing tube the radius is then smeared with a resolution function which is hard coded in class *Generator* . The usual space-time relation written in the *rtfiles* is then used to convert radii into times. The *t0files* data are also taken into account. A uniform level of noise as well as the efficiency as a function of the radius can be added.

5.9 *RPCAnalysis*

This task in under development for the reconstruction of tracks in the RPC detectors.

6 Program usage

6.1 Installing and compiling the program

The program, maintained under CVS, is available in the form of a compressed tar file at the address /afs/cern.ch/user/d/domizia/public/calib. It was developed on

Linux PC but should work on any Unix platform using GNU compiler gcc version egcs-2.91.66. You will need access to afs or a local copy of the CERNLIB (version 99 or later) and CLHEP.

6.2 Customizing the setup

The program still lacks a configuration tool allowing to define coherently geometry, number of elements (tubes or strips) for which electronics and calibration information is needed, data format and so on, so the customization of the setup is a critical point, requiring changes in many different files. In the following special emphasis will be put on settings which imply coherent changes elsewhere.

6.2.1 Data cards

In the top directory you will have to modify *calib.datacards* inserting

- the maximum number of events to be processed (must be 0 for the *CalRayTrace* task);
- the number of events to be skipped if needed;
- the number of events to be saved on file *display.dump* , to be used in PAW for MDT events visualization (PAW tools are collected in the *display* directory);
- a global debug flag, used to control the verbosity of reconstruction algorithms (for $0 < Debug < 3$) and event decoding ($Debug > 3$);
- the minimum and maximum number of TDC counts for the MDT physical window;
- the number of counts to be subtracted from the T0 quoted from the spectrum fit to MDT data to get the effective zero time;
- the data format, which has changed and will continue to change from site to site and between a test and the next one;
- the tasks to be activated and a number specifying the calling sequence.

Please note that the data format, being site and time dependent, is certainly related also to the geometrical setup and of course to the input files.

This is an example of file *calib.datacards* in top directory:

```
# max number of events per run
nev 100
# events to be skipped
# nsk 0
# events to be dumped for the display
display 10
# global debug flag (for decoding and reconstruction)
```

```
Debug  0
# physical range for TDC counts (MDT)
TDCrange 0 2000
# quantity to be subtracted for T0s
T0offset 20
# data format type
# 0 ascii
# 1 binary from DAQ-1
# 2 binary from Roma Tre LAB
# 3 ascii from Roma Tre LAB
# 4 binary from July 2001 Bundle Test in H8
DataFormat 1
# available tasks and their calling sequence number
# CalT0Det 0 # T0 determination from TDC spectra
CalAutoCal 1 # autocalibration
# CalResol 2 # resolution
# CalAlign      3       # alignment task
# CalRecon 4 # bare reconstruction
# CalRayTrace 5 # generator
# CalEffi       6 # Efficiency
# CalMonitor    7       # monitoring task
# RPCAnalysis   0       # RPCs tasks
```

Accordingly to the selected tasks the task specific datacards will have to be modified. Usually they contain cuts for the analysis to be performed, a local debug flag and often a list of SubEvents (MDTMultiLayers for MDT tasks) to be included in the reconstruction and analysis. **The selected SubEvents must of course be among the ones defined in the geometry**.

This is an example of *CalT0Det.datacards* in top directory:

```
#CalT0Det task configuration default values:
#verbosity level
debug 1
#number of bins in TDC spectra histograms (lower an upper abscissa are
#fixed to the values given to TDCrange in calib.datacards
binNum 300
#flag to activate (1) or deactivate (0) the fit to the spectra
fitTDC 0
#default T0 value assumed in case of fit failure
default 350
#initial values of parameters used in TDC spectrum fit
#those with physical meaning are (from 1 to 8)
#parameter 1 noise level outside the drift time window,
#parameter 5 related to T0
#parameter 6 related to Tmax
#parameter 7 related to the spectrum slope around T0
#parameter 8 related to the spectrum slope around Tmax
params  0. 15. 5. 100. 350. 1350. 2.5 8.
```

```
#maximum value of reduced chi2 to accept the fit result
chi2max 1.5
#list of MultiLayers to be used in the analysis
multilayers 11 12 21 22 0 0 0 0 0 0
#tube to be used for the evaluation of space-time relation from
#spectrum integration
tubeSel 11302
```

This is an example of file *CalAutoCal.datacards* in top directory:

```
#CalAutoCal task configuration default values:
#selection of the method to be used to compute the correction to space-time rel
defResidual 1
#maximum number of iterations to be performed
maxIterations 10
#verbosity level
debug   1
#minimum number of hits in accepted tracks
minHits   6
#maximum number of tracks in the event
maxTracks   1
#list of chi2 cuts to be applied on reconstructed track at each iteration
# (foresee maxiterations values!)
chiCut 50000. 5000. 1000. 500. 200. 100. 50. 20. 20. 20.
#minimum and maximum angle for accepted tracks
angMin   -0.35
angMax   0.35
#maximum number of missing hits for accepted tracks
missHits   0
#maximum number of tubes shared among different tracks
sharedTubes   0
#list of MDTMultiLayers to be used in the analysis
multilayers 11 12 0 0 0 0 0 0 0 0
#assign ml 11 and 12 to the same pattern recognition unit, i.e. look for tracks
#in both multilayers
prdetectors 1  1  0 0 0 0 0 0 0 0
```

This is an example of file *CalResol.datacards* in top directory:

```
#CalResol task configuration default values:
#verbosity level
debug   1
#minimum number of hits in accepted tracks
minHits   6
#maximum number of tracks per event
maxTracks   2
#chi2 cut of accepted tracks
chiCut   10.
#minimum and maximum angle for accepted tracks
```

```
angMin   -0.35
angMax   0.35
#maximum number of missing hits in accepted tracks
missHits   0
#maximum number of tubes shared by different tracks
sharedTubes   0
#number of points used in the resolution function
resolBins 51
#list of MDT MultiLayers to be used in analysis
multilayers 11 12 0 0  0 0 0 0 0 0
```

6.2.2 Other files

In the top directory the following files should be present

- *geofiles* , containing the list of geometry files to be loaded by the *Geometry-Handler* ,
- *rtfiles* , containing the list of space-time relation files to be loaded by the *RTRelHandler* ,
- *t0files* , containing the list of T0 files to be loaded by the *T0Handler* ,
- *eltxfiles* , containing the list of channels mapping files to be loaded by the *EltxHandler* .

For the format and contents of the files to be loaded please refer to the description included in the example files distributed with calib under sub-directory *data* .

6.2.3 Data format

The data formats implemented up to now are

- the ASCII format produced for 1998 H8 test beam data and used since then in some MC productions. For this format use the EventHandler sub-class *PBEvent* in sub-package *event* ;
- the binary format used in Roma Tre test site, decoded by the EventHandler sub-class *LABEvent* in sub-package *event* ;
- the ASCII format used in Roma Tre test site, decoded by the EventHandler sub-class *LABAsciiEvent* in sub-package *event* ;
- the DAQ-1 format for events in taken H8 2001 test beam, decoded by the EventHandler sub-class *DAQEvent* in sub-package *event* .

To account for the frequent setup changes during the data taking and simplify the decoding in the latter case the *EltxHandler* object, loading electronic channels maps for external files, has been introduced and its use will probably be propagated to the other Event classes in the near future.

This is an example of an electronic channels map file:

```
# this file contains the mapping tdc, channel -> tube for the
# specified run range (not used yet!)
elx 0 10000
# channel -> tube mapping in the first layer of a 3 layers mezzanine
# first & second multilayer
mz3 8 7 6 5 4 3 2 1 8 7 6 5 4 3 2 1
# channel -> tube mapping in the first layer of a 4 layers mezzanine
# first & second multilayer
mz4 3 6 2 5 1 4 1 4 2 5 3 6
# association mezzanine type -> tdc number -> tube id for channel 0
# (with ml = 0 for the trigger)
tdc 4 0 0
tdc 3 1 21316
tdc 3 2 21308
tdc 3 3 22316
tdc 3 4 22308
tdc 4 5 11109
tdc 4 6 11103
tdc 4 7 12107
tdc 4 8 12101
```

The above file describes the mapping of the TDC channels for 3 and 4 layers mezzanines and provides the association between tube number and channel 0 of a TDC with given number.

6.3 Developing your own task

If you wish to develop your own task remember that

- the source code should be inserted in the *user* sub-directory;
- it should inherit from *CalibrationTask* and implement the methods *taskInit* ,*taskEnd* ,*runInit* ,*runEnd* and *evtProc* ;
- method *taskEnd* should return true to stop iterating over data and false to go through another iteration;
- the new task should be added to *calib.datacards* and therefore to the *CalibrationManager::jobOpt* method which reads them in. The header file should be included in *CalibrationManager.cxx* .

6.4 Running *calib*

1. Compile the program by issuing the command **make** from the calib top directory. The executable *calib* will be produced.
2. Choose the job parameters and in particular the sequence of tasks to be executed from *calib.datacards* .

3. Check and eventually modify the datacards of the selected tasks. In particular define the list of sub-detectors to be used by the task.

4. Select the data sample and list the input files in file *input* .

5. Verify the geometry files listed in *geofiles* : these should correspond to the setup used in the data taking and should describe at least all the sub-detectors selected in step 3.

6. Verify the T0 files listed in *t0files* : these should list the tube identifier and the T0 values for all the tubes in the required sub-detectors. The T0 values can be dummy values when running *CalT0Det* but must be reliable ones for all the other tasks.

7. verify the space-time relation files listed in *rtfiles* : all the tubes in the selected sub-detectors should be listed in these files. The actual space-time and resolution functions can be just initial ones when running respectively *CalAutoCal* and *CalResol* tasks, but their binning as a function of the drift time should be the one chosen for the resulting function.

The program allows running in the same job a sequence of tasks each using the results from the previous one, for example the sequence: *CalT0Det* producing an output file *newt0.dat* listed in *t0files* in order to be used by *CalAutoCal* . Nevertheless this use is not recommended and users are encouraged to carefully check the results from a task before using them in input for another task.

References

[1] C.Bacci et al. - ATLAS MUON-97-135 (10 January 1997).

A.Biscossa et al. - ATLAS MUON-97-136 (14 January 1997).

A.Negri et al. - ATLAS MUON-97-153 (5 May 1997).

P.Creti et al. - ATLAS MUON-97-196 (29 June 1997).

C.Bini et al. - ATLAS MUON-97-204 (16 July 1997).

A.Biscossa et al. - Nucl. Instr. and Meth., A419, 331 (21 December 1998).

P.Creti et al. - ATLAS MUON-2000-005 (4 November 1999).

[2] R.Veenhof - Garfield, Cern Program Library W5050.

[3] G.Avolio et al., ATL-COM-MUON-2001-022: *First results of the 2001 MDT chambers beam test*, submitted on October 20th 2001, 18p.

[4] Silicon Graphics Computer System Inc. and Hewlett-Packard Company, Standard Template Library, http://www.sgi.com/tech/stl/

[5] L.Lönnblad, Comput. Phys. Comm. 84(1994) 307.

http://wwwinfo.cern.ch/asd/lhc++/clhep/

[6] R.Brun - Hbook, Cern Program Library Y250.

[7] F.James, M.Roos - Minuit, Cern Program Library D506.

[8] R.Brun - Paw, Cern Program Library Q121.