The Calorimeter Quick–look Software

Emiliano Mocchiutti

5th December 2005

(Software v. 3.00)

Introduction

The aim of this document is to describe the software for the quick–look analysis of the calorimeter detector. In the next sections each program will be briefly described, it will be shown how to call it and what are the input parameters. It will follow a description of the expected output, hence it will be discussed how to recognize possible problems in the detector and some constraints to the distribution shown will be shown. The calorimeter software described here has been written in C and it is supposed to be run with ROOT using as input the YODA generated files.

The installation procedure is described in the calorimeter common package readme.

1 List of quick–look functions

Two types of functions have been written for the calorimeter analysis: main functions and programs which can be called directly or from ROOT and subroutines which are used by the main functions and require input data that cannot be given by hand.

The name of main functions is composed by the word "Calo" followed by a uppercase name. Functions are contained in files usually of the same name and extension ".c". This is the list of functions with a brief description of their use:

CaloQLOOK (Version 3.00) Performs a quick–look analysis of the calorimeter data. As output three canvas are drawn with informations about the calorimeter status. No calibration data are needed to run this program. Included in the automate version of quick–look.

CaloCHKCALIB (Version 3.00) Given a calibration it draws three canvas with the RMS, thresholds, pedestals, bad strips, variance for each strip of the calorimeter anb with a report of possible errors occurred during calibration. Included in the automate version of quick–look.

These programs included in the automate version of the PAMELA quick–look. The programs that can be used to better understand the calorimeter behaviour if in trouble or to perform a more dedicated analysis are listed and described later in this document.

2 Description of quick–look programs

Programs must be executed from the command line or using root and loading the compiled library.

2.1 CaloQLOOK file from_event to_event output_dir figure_format (Version 3.00)

Input variables:

- *file*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run.

- *from_event* = 0
  Integer, first event to be analyzed. Default value is 0 (= all the events). If different from zero and with *to_event* zero the program will process only the event number *fromevent*.

- *to_event* = 0
  Integer, last event to be analyzed. Default value is 0 (= all the events). It can differ from zero only if *from_event* is not zero.

- *output_dir* = “”
  String, path to a directory where to save the output figures. Default is empty, it will save in *file*.

- *figure_format* = “ps”
  Type of figures to save. Deafult are “ps” figures, can be any format recognized by ROOT.

Example of a standard call:

```
bash> CaloQLOOK /home/pamela/filesfromyoda/dw_050301_00100/ 0
0 /tmp/ ps
```

More examples:

```
CaloQLOOK /home/data/yodafiles/dw_041126_00400/ 146 0 /tmp/ ps
```

Analyze only event number 146.

```
CaloQLOOK /home/data/yodafiles/dw_041126_00400/ 146 1000 /tmp/
ps
```

Analyze events from 146 to 1000.

```
CaloQLOOK /home/data/yodafiles/dw_041126_00400/ 69 0
/home/myhome/myyodafigures/ ps
```

Analyze event number 69 and put figures in the directory “/home/myhome/myyodafigures/”.

```
CaloQLOOK /home/data/yodafiles/dw_041126_00400/ 0 0
/home/myhome/myyodafigures/ eps
```

Analyze all events and put "eps" figures in the directory "/home/myhome/myyodafigures/".

The output is reported in three canvas which have as title "Calorimeter_Detector_Report_x/3" with "x" from one to three.

Calorimeter_Detector_Report_1/3 :

This canvas is divided into two main zone, left and right. An example of this canvas is shown in figure 1.

FIGURE 1 – RIGHT: on the right there could be one or two panels depending on the acquisition mode of the calorimeter. In the case data were acquired only in COMPRESS or RAW mode one panel will be shown reporting in a histogram the number of strip hit. In this case a strip is considered hit if its value is not zero, hence in RAW mode there must be a delta function centered on the number of working strips; in the case of COMPRESS mode there must be a distribution which average is the mean number of strips above the DSP threshold. In the case of FULL mode or in any case in which there is a mixture of acquisition modes two panels will be shown, in the upper one there will be the COMPRESS mode distribution while in the lower one the RAW mode distribution will be shown. If there is no cosmic ray trigger the mean of the COMPRESS mode must be of about 20 in the case of the full working calorimeter, with a RMS of about 8. In the case of ground muons there should be a distribution centered at about 60 plus a distribution of pulser events at about 20. If everything is working correctly the mean of RAW mode must be 4223 (one strip died during integration in Rome) and no events must be out of the delta.

FIGURE 1 – LEFT: on the left is shown, if no errors occurred, the counter coming from the calorimeter section YE (*calevnum(2)*)versus a counter in YODA (*iev*). In the case of RAW mode there is no DSP information, hence no figure. In FULL and COMPRESS mode a linear dependence must appear between the two variables. Notice that in case of mixed acquisition the line could start at iev greater than zero.
The program check also the counters coming from the other three sections. If any difference between the four counters is found instead of the single graph just discussed above four figures are drawn. From lower right clockwise they represent: the counter coming from the calorimeter section YE versus a counter in YODA, the absolute difference between counters from section XE and YO, the absolute difference between counters from section XE and XO and the absolute difference between counters from section YE and YO.
Another check performed by CaloQLOOK is the correct corrispondence of RAW and COMPRESS data in the case of FULL acquisition. If any difference is found in the left side of the canvas only a figure is shown with the difference between RAW and COMPRESS data instead of counter figure(s).
Only the calevnum(2)%iev figure must be shown in standard conditions and at least one FULL or COMPRESS event. In case of pure RAW mode acquisition there will be written "No calevnum infos from DSP".

Calorimeter_Detector_Report_2/3 :

Four figures are plotted in this canvas, an example can be found in figure 2. From top right clockwise:

DEXYC: the energy distribution in ADC channels in the case of COMPRESS mode acquisition. If there are no COMPRESS data this pad will remain empty. In a standard situation there must be a peak at zero and a distribution of data at about 3000/5000 ADC channels. Since there is a dead strip there will be a small peak at about 32000 ADC channels.
The distribution should not spread over more than about 3000 ADC channels, if it does there must be some noisy strips (has the calorimeter just been turned on? is the power supply working correctly?).

DEXY: the energy distribution in ADC channels in the case of RAW mode acquisition. If there are no RAW or FULL data this pad will remain empty. In a standard situation there must be a distribution of data at about 3000/5000 ADC channels.
There should not be any high signal at about 32000 ADC channels, that would mean a latch–up alarm in one or more views. Again, since there is a dead strip, it could be possible to see a small peak at about 32000 ADC channels. The distribution should not spread over more than about 3000 ADC channels, if it does there must be some noisy strips (has the calorimeter just been turned on? is the power supply working correctly?).

CALSTRIPHIT: this is the total number of strips above threshold in the case of COMPRESS mode acquisition as calculated by the DSP. In case of pure RAW data this figure will be empty. This distribution must be very similar (could be also identical) to the one presented in the report sheet number one (right, on the top).
The mean must be of about 10 in the case of the full working calorimeter, with a RMS of about 10, if there is no trigger. In the case of ground muons there should be a distribution centered at about 50/60.

BASELINES: in this figure is reported the distribution of the common noise of preamplifiers as calculated by the DSP (no information in RAW mode). In a standard situation there must be a distribution of data at about 3000/5000 ADC channels.
There must be no signal at about 32000 ADC channels, that would mean a latch–up in one or more views. The distribution should not spread over more than about 3000 ADC channels.

Calorimeter_Detector_Report_3/3 :
The last canvas summerizes the calorimeter status, checking for errors coming from the DSP and errors occurred during the unpacking of data, see figure 3. The information from the four sections of the calorimeter is displayed in four different pads. The first four lines, written in green colour, show how many times the calorimeter acquired data in RAW, COMPRESS or FULL mode and how many time a preamplifier was fully transmitted. All the other lines, written in red colour, represent the possible errors occurred. The lines of errors coming from the DSP of the calorimeter start with a star to distinguish them by error coming from the YODA unpacking program. The possible errors are:

- “* DSP ack error”: the DSP was not able to answer to the CPU.

- “* Temp. alarm”: temperature alarm.
- “* Latch up alarm”: latch–up alarm, one or more views have been turned off. The number of planes with latch–up is reported. Below under parenthesis there is the multiplicity of the latch–up as calculated by CaloQLOOK looking at the data. Notice that sometimes it could be a latch–up alarm that CaloQLOOK does not recognize on data or a latch–up alarm seen by CaloQLOOK but not reported by the DSP. The last case as been proved to be a “false” latch–up and to be the case in which all strips of one plane gave signal zero (even in RAW mode!). This seems to be due to the fact that the section had just been turned on and it needed some time to warm up.
- “* CMD length error”: the DSP received a command of an unexpected length.
- “* Execution error”: the DSP was not able to execute the command.
- “* CRC error (st. word)”: the CRC of the command received was wrong.
- “View or command not recognized” (YODA error code = 128): YODA was not able to recognize the calorimeter view or command.
- “Missing section” (YODA error code = 129): YODA did not find the calorimeter section.
- “CRC error (data)” (YODA error code = 132): the calculated CRC on data was not equal to the CRC transmitted by the DSP.
- “Length problems in RAW mode” (YODA error code = 133): the length of RAW data was not 1064 words.
- “Length problems in COMPRESS mode” (YODA error code = 134): the length given by the DSP is not compatible with COMPRESS mode data.
- “Length problems in FULL mode” (YODA error code = 135): the length given by the DSP is not compatible with FULL mode data.
- “Acquisition mode problems” (YODA error code = 136): the event seems to be not RAW nor COMPRESS nor FULL.
- “Problems with coding” (YODA error code = 139): in COMPRESS or FULL mode YODA cannot distinguish the word containing the signal from the word containing the common noise (baseline).
- “Pedestal checksum wrong” (YODA error code = 140): the table containing the pedestals which are needed to compress the data and to compute the baseline has been corrupted (a new calibration procedure is needed!).
- “Thresholds checksum wrong” (YODA error code = 141): the table containing the thresholds which are needed to compress the data and to compute the baseline has been corrupted (a new calibration procedure is needed!).
- “Packet length is zero (yoda, input error), skipped” (YODA error code = 142): the input buffer given to the unpacking routines of the calorimeter had length equal to zero.

Moreover other two kind of errors are coming from checks made by CaloQLOOK and can appear in this canvas:

- "Calevnum jump": in FULL or COMPRESS mode the counter of the section was incremented by two between to neighbour events. Notice that this can be due to a CRC error in the previous event.
- "Full mode, differences between raw and compress mode": in full mode the program detected differences between the RAW and COMPRESS mode acquisition (when the compress data are above threshold).
- "WARNING! DEXYC < 0": negative compressed signal.

There should be no red errors. The number of events must be the same for the four different sections and the preamplifiers can be fully transmitted only in COMPRESS or FULL mode. In that case the number of preamplifier transmission should not exceed half of the total number of COMPRESS and FULL mode events on each section.

2.2 CaloCHKCALIB file calib_number output_directory matra figure_format (version 3.00)

Input variables:

- *file*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run.

- *calib_number* = 0
  Integer, the number of the calibration to check. The default value is zero, it will show one by one all the calibration in the input file.

- *output_directory* = ""
  String, path to a directory where to save the output figures. Default is empty, it will save in *file*.

- *matra* = 0
  When set to 1 display the colour figure of RMS values.

- TString *figure_format* = "ps"
  Type of figures to save. Deafult are "ps" figures, can be any format recognized by ROOT.

Example of a standard call:

```
bash> CaloCHKCALIB /home/data/yodafiles/dw_041126_00400 0 /tmp/
0 ps
```

More examples:

```
CaloCHKCALIB /home/data/yodafiles/dw_041126_00400/ 3 /tmp/ 0 ps
```

Analyze calibration number 3.

```
CaloCHKCALIB /home/data/yodafiles/dw_041126_00400/ 0
/home/myhome/myyodafigures/calorms/ 0 ps
```

Analyze all the calibrations and write the figures in directory "/home/myhome/myyodafigures/calorms/".

The output is reported in different canvas.

In canvas which has the title "Calorimter:_strip_RMS", see figure 5, two panels are shown: in the upper panel the RMS of each strip of each Y–plane is drawn. The same in the lower one but for X-planes. Tiny white lines between strips are a normal visualization effect. There should be no white planes (latch–up) and no white strips (dead strips). The number of "bad" strips can vary and could be a maximum of about 25/30 in each panel (50/60 in total). Usually the two panels must be bluish, some green strips (about 15/20 in each panel), some red strips (about 5/10 in each panel), some black strips (about 5/10 in each panel). No violet strips should be seen.

Other figures:

FIGURE C14:

Description:
The pedestals of calorimeter strips (ADC channels) obtained during a calibration procedure.
NOMINAL:
Most of values inside yellow region.
STANDARD situations:
- some hits outside yellow region;
- about hundred consecutive values at zero; latch–up alarm from a plane during calibration?
ACTION: check a following calibration if problem persist contact specialist.
NON–STANDARD situations:
- eleven set of hundred planes at zero. One section is missing.
- all zero.
ACTION: call specialist.

FIGURE C15:

Description:
The RMS of calorimeter strips (ADC channels) obtained during a calibration procedure.
NOMINAL:
Most of values inside yellow region.
STANDARD situations:
- some hits outside yellow region;
- about hundred consecutive values at zero; latch–up alarm from a plane during calibration?
ACTION: check a following calibration if problem persist contact specialist.
NON–STANDARD situations:
- eleven set of hundred planes at zero. One section is missing.
- all zero.

ACTION: call specialist.

FIGURE C16:

Description:
Bad strips during a calibration procedure (not used in baseline calculation).
NOMINAL:
No more than 50 hits.
NON–STANDARD situations:
- all black.
- no hits.
ACTION: call specialist.

FIGURE C17:

Description:
The thresholds (ADC channels) used during a calibration procedure.
NOMINAL:
Most of values inside yellow region.
STANDARD situations:
- some hits outside yellow region;
- about six consecutive values at zero or 255; latch–up alarm from a plane during calibration?
- about six consecutive values at higer values; noise on one plane during calibration?
ACTION: check a following calibration if problem persist contact specialist.
NON–STANDARD situations:
- eleven set of hundred planes at zero or 255. One section is missing.
- all zero.
ACTION: call specialist.

FIGURE C18:

Description:
Strip variance (ADC channels) during a calibration procedure.
NOMINAL:
Most of values inside yellow region.
STANDARD situations:
- some hits outside yellow region;
- about six consecutive values at zero or 255; latch–up alarm from a plane during calibration?
- about six consecutive values at higer values; noise on one plane during calibration?
ACTION: check a following calibration if problem persist contact specialist.
NON–STANDARD situations:
- eleven set of hundred planes at zero or 255. One section is missing.
- all zero.
ACTION: call specialist.

FIGURE C18:

Description:
Baselines (ADC channels) obtained during a calibration procedure.
NOMINAL:
Most of values inside yellow region. **STANDARD situations:**
- some hits outside yellow region;
- about six consecutive values at zero or 32000; latch–up alarm from a plane during calibration?
- about six consecutive values at higer values; noise on one plane during calibration?
ACTION: check a following calibration if problem persist contact specialist.
NON–STANDARD situations:
- eleven set of hundred planes at zero or 32000. One section is missing.
- all zero.
ACTION: call specialist.

FIGURE C20:

Description:
Calorimeter calibration status, checking for errors coming from the DSP and errors occurred during the calibration procedure.
NOMINAL:
No “red” errors. The number of calibrations must be the same for the four different sections.
NON–STANDARD situations:
- different number of calibrations for different sections.
- any “red” error.
ACTION: call specialist.

3 List of other functions and subroutines

CaloPULSE (Version 1.05) Shows the distribution of ADC channels transmitted during the pulse calorimeter calibration. Included in the automate version of quick–look.

CaloLEVEL1 (Version 1.21) Calibrate the calorimeter (first order calibration) and save data in a rootple inside the YODA structure (under /Physics/Level1/). It search for calibrations using CaloFindCalibs subroutine, if it fails because there are no calibrations in the file and no previous files processed it requires as input a calibration file to be used.

CaloPLANES (Version 2.13) Shows the number of hit for each strip and each plane of the calorimeter in a data file. It requires at least one calibration in the data file. It can be used as tool to see tracks in the calorimeter.

CaloMIP (Version 2.12) Calls the calibration program and print out the energy distribution together with the number of strip hit (*nstrip*) and the distribution of the energy released (*qtot*) for each event.

CaloMATRA (Version 3.00) Tools to look at calorimeter tracks. It requires at least one calibration in the data file.

CaloFINDCALIBS.c (Version 2.06) This macro has been moved in the package "COMMON". It provides:

- **CaloFINDCALIBS** Search for calibration in the data file and print out on the terminal the results. Look also in previous files to associate correctly at each event a calibration.
- **CaloOLDFINDCALIBS** Search for calibration in the data file and print out on the terminal the results.
- **CaloMySQLFINDCALIBS** Search for calibration in the database and print out on the terminal the results.
- **CaloMySQLFILLCALIBS** Fill the calibrations in the database and print out on the terminal the results.

CaloADC2MIP.c (Version 4.01) This macro has been moved in the package "UTILITIES". This file contains calorimeter functions that can be used to determine the ADC to MIP conversion value for each strip. This file provides:

- **CaloADC2MIP** Given a file list it will put in 4224 histograms the ADC value for each strip and at the end it will perform a convoluted Landau–Gaussian fit for each strip.
- **Calo4224BAK** Shows the 4224 histograms from backup files.
- **Calo4224FIT** Shows the 4224 histograms from final file.
- **Calo4224STATUS** Shows the error on the fitted peak for each strip.
- **Calo4224MIPVALUES** Shows 22 histograms with the value of the conversion factor as function of the strip number.
- **CaloBAKFIT** Performs the fit on backup figures.
- **CaloRAWADC2MIPDATA** Save in a rootple the ADC values for each strip for each event of a list of files.
- **CaloLOOKATSTRIP** Shows the strip ADC distribution as function of the event number or OBT (only for RAW and FULL data).
- **CaloRAWADC2MIPPLOT** Save in a rootple histograms with the ADC values for each strip for each event of a list of files.

CaloCALIBSCAN (Version 1.05) Moved to "UTILITIES". Save in a rootple calorimeter calibration values and time, to check how calibration values vary on time.

CaloTRKCALOALIG (Version 1.01) Moved to "UTILITIES". Given an input file determine alignement parameters between tracker and calorimeter.

Notice that CaloPLANES, CaloMIP, and CaloMATRA requires the output of CaloLEVEL1 function. If CaloLEVEL1 has not been run before launching one of that programs CaloLEVEL1 will be automatically called.

The name of subroutines is composed again by the word “Calo” but it is not followed by all upper case letters. Subroutines are written in the file “CaloFunctions.h” (version 3.04, moved to “COMMON”) which is included in the program files. The subroutines are:

CaloCompressData Compression algorithm for RAW data.

CaloFindBaseRaw Determines the baseline starting from raw data. The resulting baseline must be identical to the one computed by the DSP in the case of FULL mode acquisition.

CaloFindBaseRawNC Determines the baseline starting from raw data. The resulting baseline must be identical to the one computed by the DSP in the case of FULL mode acquisition. Same as before but do not compress data.

CaloFindBase Determines the baseline starting from raw data but using relaxed condition on the minimum number of strips needed and discarding the lowest energy strip.

CaloFindCalibs Finds the calibration inside the data file and determine time limits for which each calibration has to be used. Looks also in previous file if the case. Associates events to the previous calibration in time.

OLDCaloFindCalibs Finds the calibration inside the data file and determine time limits for which each calibration has to be used. To be used in conjuction with CaloOLDFINDCALIBS. Associates events to the closer calibration.

Calo1stCalib Calls the first calibration on the file and store data into memory. To be used with CaloFINDCALIBS.

OLDCalo1stCalib Calls the first calibration on the file and store data into memory. To be used with CaloOLDFINDCALIBS.

CaloPede Given the calibration that must be used it returns the pedestal for each strip of the calorimeter.

getFilename Given the path to the YODA directory it returns the .dat filename.

getLEVname Given the path to the YODA directory and the data level number returns a filename formatted as YODA does.

ColorMip Given the energy in MIP it returns the colour to be used in figure for hit of the given energy.

fetchpreviousfile looks for a file containing a good calibration for a given section and returns a set of parameters. Used in CaloFindCalibs.

whatnamewith given the set of parameters from “fetchpreviousfile” returns the filename of the file which match that set. Used in CaloFindCalibs.

WhatToDo user interactive subroutine.

PrintFigure prints a figure with a special formatting.

langaufun Gaussian–Landau convoluted function.

langaufit Gaussian–Landau convoluted fit.

langaupro Finds the peak of the Gaussian–Landau fitted function.

delay Given the TDC channels of the self–trigger calorimeter delay it returns a time in milliseconds.

stringcopy copies strings.

stringappend appends strings.

getEmiFile almost the same as getFile in yodautility.c but it does not crash when running over a large number of files.

fitraw exponential function to be used in fitting routines.

The classes defined in caloclasses.h are:

CalorimeterLevel1 calorimeter Level1 class.

CalorimeterADCRAW contains ADC values for RAW mode data.

CalorimeterCalibration class to contain the 4224 conversion values.

CalorimeterCalibScan contains calibration data values.

4 Description of the other programs

In the following session all examples are given starting from ROOT but now each program can be called as a compiled executable without entering ROOT. If you still want to use ROOT please use the compiled version loading libraries with ".x CaloNAMEOFPRORAM.C".

4.1 CaloPULSE(TString *filename*, TString *outDir* = "", Int_t *tosave* = 0, TString *saveas* = "eps") (Version 1.05)

Input variables:

- TString *filename*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run.

- TString *outDir* = ""
  String, path to a directory where to save the output figures. Default is empty, it will save in *filename*.

- Int_t *tosave* = 0
  Flag to tell the function to save figures or not. Default is 0, do not save figures.

- TString *saveas* = “eps”
  Type of figures to save. Deafult are “eps” figures, can be any format recognized by ROOT.

Example of a standard call:

```
bash> root
root [1] .L CaloPULSE.c
root [2] CaloPULSE("/home/data/yodafiles/dw_041126_00400/");
```

More examples (the first two steps are mandatory):

```
root [2] CaloPULSE("/home/data/yodafiles/dw_041126_00400/",
"/home/mocchiut/myfigures/",1,"gif");
```

Save “gif” figures in the directory “/home/mocchiut/myfigures/”.

The output is reported canvases which has the title “calpulse”, see figure 4.

Due to a known bug the pulse type and strip number reported in figures are wrong when a file contains more than one calibration.

Usually when injecting with pulse type 8005 a distribution at about 3000/5000 ADC channel should be seen. When injecting with pulse type 8015 it should be possible to see another small distribution at higher ADC channel values. If the distribution is centered at about zero with values spreading from about -32000 to about 32000 ADC channels do nothing, it is a known bug in CPU software. If the last case, just report to calorimeter people (LOW severity).

For each calibration four figures are produced, one for each section; each figure shows as zero also the values of the other sections.

4.2 CaloLEVEL1(TString *filename*, TString *calcalibfile* = “”, Int_t *FORCE* = 0) (Version 1.21)

Input variables:

- TString *filename*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run.

- TString *calcalibfile* = “”
  String, path to a file which contains calorimeter calibration. Default is empty.

- Int_t *FORCE* = 0
  Flag to force the reprocessing of data overwriting existing files. Default is 0, the program will exit if it will find the LEVEL1 rootple.

Example of a standard call:

```
bash> root
root [1] .L CaloLEVEL1.c
root [2] CaloLEVEL1("/home/data/yodafiles/dw_041126_00400/");
```

More examples (the first two steps are mandatory):

```
root [2] CaloLEVEL1("/home/data/yodafiles/dw_041126_00400/",
"",1);
```

Force the reprocessing of data.

Due to a known bug (to be solved in next version) the processing time could become high if the files contains events which calibration has to be found in a previous (in time) file but this file has not been processed with yoda. In that case “CaloLEVEL1” will process anyway the data if it will find at least one calibration in the processed file or if a calibration file has been given as input. It will exit if no calibration are found.

No output is produced but the Level1 rootple.

4.3 CaloPLANES(TString *filename*, TString *viewxy*=“both”, TString *parity*=“both”, Int_t *plane* = 0, Int_t *fromevent* = 0, Int_t *toevent* = 0, TString *outDir* = “”, TString *saveas* = “eps”) (Version 2.13)

Input variables:

- TString *filename*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run.
- TString *viewxy*
  String, calorimeter view to analyze. Default value is “both”, can be “x” or “y”.
- TString *parity*
  String, calorimeter planes to analyze. Default value is “both”, can be “odd” or “even”.
- Int_t *plane* = 0
  Integer, plane number to analyze. Default value is zero, all planes. The range goes from 1 to 11.
- Int_t *fromevent* = 0
  Integer, first event to be analyzed. Default value is 0 (= all the events). If *toevent* is zero the program will process only the event number *fromevent* when different from zero.
- Int_t *toevent* = 0
  Integer, last event to be analyzed. Default value is 0 (= all the events). It can differ from zero only if “fromevent” is not zero.
- TString *outDir* = “”
  String, path to a directory where to save the output figures. Default is empty, it will save in the YODA root directory for the data file that will be analyzed.

- TString *saveas* = “eps”
  Type of figures to save. Deafult are “eps” figures, can be any format recognized by ROOT.

Example of a standard call:

```
bash> root
root [1] .L CaloPLANES.c
root [2] CaloPLANES("/home/data/yodafiles/dw_041126_00400/");
```

More examples (the first two steps are mandatory):

```
root [2] CaloPLANES("/home/data/yodafiles/dw_041126_00400/",
"both","odd",0);
```

Analyze x and y odd views, all planes, all events.

```
root [2] CaloPLANES("/home/data/yodafiles/dw_041126_00400/",
"both","both",7,146,1000);
```

Analyze x and y, odd and even plane number 7 from event 146 to event 1000.

```
root [2] CaloPLANES("/home/data/yodafiles/dw_041126_00400/",
"both","both",0,0,0,"/home/myhome/myyodafigures/");
```

Analyze the whole calorimeter, all events and put figures in the directory “/home/myhome/myyodafigures/”.

```
root [2] CaloPLANES("/home/data/yodafiles/dw_041126_00400/",
"both","both",0,143);
```

Analyze the whole calorimeter, only event number 143.

The output is reported in one or two canvas depending on the input parameters. The whole calorimeter, standard execution of the program, fills two canvas. The first contains x even and y odd planes, the second one x odd and y even planes, see figure 6. For each event the energy distributed on each plane has been normalized to one in order to enhance noisy or bad working strips.

Notice, last example and figure 7, that this program can be used to see tracks and to check the trasversal distribution of energy for each plane in the case of the single event. However, since the energy is normalized in each plane there could be some distorsions in the visualization.

In standard conditions each plane should show a uniform distribution without any hole (dead strips) or peak (noisy strips). The shape of the distribution will depend on the number of plane and on the PAMELA acceptance as function of rigidity. Two neighbour planes must have similar distributions. In the case of a noisy strip it could be difficult to distinguish the signal of the other strips, in that case it will be necessary to change the y–scale from linear to logarithmic (click with the right button of the mouse on the figure and click on SetLogy). The linear scale is essential to notice any distorsion in the distribution (a bad calibration of one preamplifier, for example).

4.4 CaloMIP(TString *filename*, Int_t *view* = 0, Int_t *plane* = 0, Int_t *strip* = 0, Int_t *fromevent* = 0, Int_t *toevent* = 0, TString *outDir* = "", TString *saveas* = "eps") (Version 2.12)

Input variables:

- TString *filename*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run.

- Int_t *view*
  String, calorimeter view to analyze. Default value is 0 (both), can be 1 (x) or 2 (y).

- Int_t *plane* = 0
  Integer, plane number to analyze. Default value is zero, all planes. The range goes from 1 to 22.

- Int_t *strip* = 0
  Integer, strip to be analyzed. Default value is 0 (all the strips). The range goes from 1 to 96.

- Int_t *fromevent* = 0
  Integer, first event to be analyzed. Default value is 0 (= all the events). If *toevent* is zero the program will process only the event number *fromevent* when different from zero.

- Int_t *toevent* = 0
  Integer, last event to be analyzed. Default value is 0 (= all the events). It can differ from zero only if "fromevent" is not zero.

- TString *outDir* = ""
  String, path to a directory where to save the output figures. Default is empty, it will save in the YODA root directory for the data file that will be analyzed.

- TString *saveas* = "eps"
  Type of figures to save. Deafult are "eps" figures, can be any format recognized by ROOT.

Example of a standard call:

```
bash> root
root [1] .L CaloMIP.c
root [2] CaloMIP("/home/data/yodafiles/dw_041126_00400/");
```

More examples (the first two steps are mandatory):

```
root [2] CaloMIP("/home/data/yodafiles/dw_041126_00400/",
1,17,0);
```

Analyze x view, plane 17, all strips and all events.

```
root [2] CaloMIP("/home/data/yodafiles/dw_041126_00400/",
2,13,45);
```
Analyze x view, plane 13, strip 45, all events.

The output are two canvas. The first one, figure 8, shows the MIP distribution. The second one, figure 9, shows on the left the number of strip hit discarding bad strips and on the right the corresponding total energy deposit in the calorimeter. In the case of protons and muons the two distribution should peak at about 44 strips and 50 mips.

4.5 CaloMATRA(TString *filename*, Int_t *fromevent* = 1, Int_t *toevent* = 0, TString *tyhist* = "box", TString *outFile* = "", TString *saveas* = "eps") (Version 3.00)

Input variables:

- TString *filename*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run.
- Int_t *fromevent* = 1
  Integer, first event to be analyzed. Default value is one, the first event of the file. It can be zero, in that case it will show all the events one by one all the calibration in the input file waiting for the user to press enter in between. If *toevent* is zero the program will process only the event number *fromevent* when different from zero.
- Int_t *toevent* = 0
  Integer, last event to be analyzed. Default value is 0 (= all the events). It can differ from zero only if "fromevent" is not zero.
- TString *tyhist* = "box"
  String, type of visualitation. Default is "box", it can be "lego" to see three dimensional, black and white figures.
- TString *outDir* = ""
  String, path to a directory where to save the output figures. Default is empty, it will save in the YODA root directory for the data file that will be analyzed.
- TString *saveas* = "eps"
  Type of figures to save. Deafult are "eps" figures, can be any format recognized by ROOT.

Example of a standard call (it will show event number one):
```
bash> root
root [1] .L CaloMATRA.c
root [2] CaloMATRA("/home/data/yodafiles/dw_041126_00400/");
```

More examples (the first two steps are mandatory):

```
root [2] CaloMATRA("/home/data/yodafiles/dw_041126_00400/",
1,10);
```
Analyze events from 1 to 10.

Two panels with the strip hit in the x and y views of the calorimeter are drawn, see figure 10. Below the lower panel the number of strip hit (nstrip), the total energy released (QTOT) and the number of bad strips are printed. The bad strips are not used in the calculation of NSTRIP and QTOT but are shown in the figures. Only strips with energy above 0.7 MIP are plotted and used in the calculation.

4.6 CaloFINDCALIBS(TString *filename*) (Version 2.06)

Input variables:

- TString *filename*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run.

This program searches in the file if there is any good calibration. If so it will divide the file in time bins and it will associate at each bin a calibration. This is done for each section without any connection to the other sections.

This is an example of the output is printed on the screen:

```
bash> root
root [1] .L CaloFINDCALIBS.c
root [2] CaloFINDCALIBS("/home/filesfromyoda/dw_050301_00200/");

 Obtjump = 0 - FIRST OBT 26622578 - LAST OBT 36637127

 ------ /home/filesfromyoda/dw_050301_00200/ -------

 ** SECTION 0 **
 - from time 26622578 to time 27028067 use calibration at
 time 23709196, file: /home/filesfromyoda/dw_050301_00100
 - from time 27028067 to time 29311525 use calibration at
 time 27028067, file: /home/filesfromyoda/dw_050301_00200/
 - from time 29311525 to time 32980945 use calibration at
 time 29311525, file: /home/filesfromyoda/dw_050301_00200/
 - from time 32980945 to time 36637127 use calibration at
 time 32980945, file: /home/filesfromyoda/dw_050301_00200/

 ** SECTION 1 **
 - from time 26622578 to time 27028121 use calibration at
 time 23709250, file: /home/filesfromyoda/dw_050301_00100
 - from time 27028121 to time 29311579 use calibration at
 time 27028121, file: /home/filesfromyoda/dw_050301_00200/
 - from time 29311579 to time 32980999 use calibration at
 time 29311579, file: /home/filesfromyoda/dw_050301_00200/
 - from time 32980999 to time 36637127 use calibration at
 time 32980999, file: /home/filesfromyoda/dw_050301_00200/

 ** SECTION 2 **
 - from time 26622578 to time 27028174 use calibration at
 time 23709303, file: /home/filesfromyoda/dw_050301_00100
 - from time 27028174 to time 29311639 use calibration at
 time 27028174, file: /home/filesfromyoda/dw_050301_00200/
 - from time 29311639 to time 32981053 use calibration at
 time 29311639, file: /home/filesfromyoda/dw_050301_00200/
 - from time 32981053 to time 36637127 use calibration at
 time 32981053, file: /home/filesfromyoda/dw_050301_00200/

 ** SECTION 3 **
 - from time 26622578 to time 27028228 use calibration at
 time 23709357, file: /home/filesfromyoda/dw_050301_00100
 - from time 27028228 to time 29311693 use calibration at
 time 27028228, file: /home/filesfromyoda/dw_050301_00200/
 - from time 29311693 to time 32981106 use calibration at
 time 29311693, file: /home/filesfromyoda/dw_050301_00200/
 - from time 32981106 to time 36637127 use calibration at
 time 32981106, file: /home/filesfromyoda/dw_050301_00200/

 ----------------------------------------------------------
```

The program CaloMySQLFINDCALIBS.c will query the database and print on the screen the same kind informations.

The old program will divide the time intervals using only calibrations inside the processed file and associating at each event the closer calibration (in time). This is an example of the output:

```
bash> root
root [1] .L CaloFINDCALIBS.c
root [2] CaloOLDFINDCALIBS("/home/filesfromyoda/dw_050301_00200/");
 ----------------------------------------------------------

 Section 0 from time 0 to time 28169796 use calibration
                                             at time 27028067
 Section 0 from time 28169796 to time 31146235 use calibration
                                             at time 29311525
 Section 0 from time 31146235 use calibration at time 32980945
Section 1 from time 0 to time 28169850 use calibration
                                             at time 27028121
Section 1 from time 28169850 to time 31146289 use calibration
                                             at time 29311579
Section 1 from time 31146289 use calibration at time 32980999

Section 2 from time 0 to time 28169906 use calibration
                                             at time 27028174
Section 2 from time 28169906 to time 31146346 use calibration
                                             at time 29311639
Section 2 from time 31146346 use calibration at time 32981053

Section 3 from time 0 to time 28169960 use calibration
                                             at time 27028228
Section 3 from time 28169960 to time 31146399 use calibration
                                             at time 29311693
Section 3 from time 31146399 use calibration at time 32981106

-----------------------------------------------------------
/home/filesfromyoda/dw_050301_00200/
```

4.7 CaloADC2MIP.c (Version 4.01)

Only the main program can be called with the executables without using ROOT. For other routines start ROOT and load the compiled library.

4.7.1 CaloADC2MIP(TString *filename*, TString *calcalibfile* = “”, TString *flist* = “”)

Input variables:

- TString *filename*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run. If used in conjuction with “flist” this variable must contain the path to the directory containing the YODA directories from unpacked files.

- TString *calcalibfile* = “”
  File containing a valid calibration for calorimeter. Default is empty, it will search for calibrations. In certain cases it could be necessary to give an input file here.

- TString *flist* = “”
  Path to a file containing a list of files to process. The list must contain YODA filenames separated by spaces or in columns. For example the file could contain a list of four files like:
  dw_050302_00100 dw_050302_00200 dw_050302_00300 dw_050224_00100

Example of a standard call:
```
bash> root
root [1] .L CaloADC2MIP.c
root [2] CaloADC2MIP("/home/data/yodafiles/",
"/home/data/yodafiles/dw_041126_00400/","mylist.txt");
```
Run over files listed in mylist.txt file using file dw_041126_00400/ as calibration file if it fails in searching the best calibration.

4.7.2 Calo4224BAK(TString *filename*)

Input variables:

- TString *filename*
  Name of the backup file to look into.

Example of a standard call:
```
bash> root
root [1] .L CaloADC2MIP.c
root [2] Calo4224BAK("CaloADC2MIPf10.bak");
```
Look at figures contained in the file "CaloADC2MIPf10.bak".

4.7.3 Calo4224FIT(TString *filename* = "CaloADC2MIPf.root", TString *filevalue* = "CaloADC2MIP.root", TString *type*="")

Input variables:

- TString *filename*
  Name of the file containing figures.
- TString *filevalue*
  Name of the file containing calorimeter conversion values.
- TString *type*
  Description of the data contained in the two previous files (backup data or final data). Default is "", final data, can be "bak" instead.

Example of a standard call:
```
bash> root
root [1] .L CaloADC2MIP.c
root [2] Calo4224FIT();
```
Look at figures and data contained in the files "CaloADC2MIPf.root" and "CaloADC2MIP.root". It allows to fit again figures and to modify values by hand changing the "CaloADC2MIP.root" file used by CaloLEVEL1. See figure 11.

4.7.4 Calo4224STATUS(TString *filename* = "CaloADC2MIP.root", TString *type*="")

Input variables:

- TString *filename*
  Name of the file containing calorimeter conversion values.

- TString *type*
  Not used variable (yet).

Example of a standard call:
```
bash> root
root [1] .L CaloADC2MIP.c
root [2] Calo4224STATUS();
```
See figure 12.

4.7.5 Calo4224MIPVALUES(TString *filename* = "CaloADC2MIP.root", TString *type*="")

Input variables:

- TString *filename*
  Name of the file containing calorimeter conversion values.
- TString *type*
  Not used variable (yet).

Example of a standard call:
```
bash> root
root [1] .L CaloADC2MIP.c
root [2] Calo4224MIPVALUES();
```
Returns eleven figures with ADC to MIP conversion values as function of the strip number. 100 is summed to y–views strip number. It requires to press enter to switch from a figure to the next one. See figure 13.

4.7.6 CaloBAKFIT(TString *filename*)

Input variables:

- TString *filename*
  Name of the file containing the backup figures from CaloADC2MIP function.

Example of a standard call:
```
bash> root
root [1] .L CaloADC2MIP.c
root [2] CaloBAKFIT("CaloADC2MIPf120.bak");
```

4.7.7 CaloRAWADC2MIPDATA(TString *filename*, TString *calcalibfile* = "", TString *flist* = "")

Input variables:

- TString *filename*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run. If used in conjuction with "flist" this variable must contain the path to the directory containing the YODA directories from unpacked files.

- TString *calcalibfile* = “”
  File containing a valid calibration for calorimeter. Default is empty, it will search for calibrations. In certain cases it could be necessary to give an input file here.

- TString *flist* = “”
  Path to a file containing a list of files to process. The list must contain YODA filenames separated by spaces or in columns. For example the file could contain a list of four files like:
  dw_050302_00100 dw_050302_00200 dw_050302_00300 dw_050224_00100

Example of a standard call:
```
bash> root
root [1] .L CaloADC2MIP.c
root [2] CaloRAWADC2MIPDATA("/home/data/yodafiles/",
"/home/data/yodafiles/dw_041126_00400/","mylist.txt");
```
Create the file CaloADC2MIPdata.raw and fill it with ADC values of the files contained in mylist.txt. If necessary use file dw_041126_00400/ to calibrate data.

4.7.8 CaloRAWADC2MIPPLOT(TString *filename*, TString *calcalibfile* = “”, TString *flist* = “”)

Input variables:

- TString *filename*
  String, name of the root directory created by YODA for the data file is going to be analyzed. There is no default value, without this input the program will not run. If used in conjuction with “flist” this variable must contain the path to the directory containing the YODA directories from unpacked files.

- TString *calcalibfile* = “”
  File containing a valid calibration for calorimeter. Default is empty, it will search for calibrations. In certain cases it could be necessary to give an input file here.

- TString *flist* = “”
  Path to a file containing a list of files to process. The list must contain YODA filenames separated by spaces or in columns. For example the file could contain a list of four files like:
  dw_050302_00100 dw_050302_00200 dw_050302_00300 dw_050224_00100

Example of a standard call:
```
bash> root
root [1] .L CaloADC2MIP.c
root [2] CaloRAWADC2MIPDATA("/home/data/yodafiles/",
"/home/data/yodafiles/dw_041126_00400/","mylist.txt");
```
Create the file CaloADC2MIP.raw and fill it with 4224 histograms of RAW ADC values of the files contained in mylist.txt. If necessary use file dw_041126_00400/ to calibrate data.

4.7.9 CaloLOOKATSTRIP(Int_t *view*, Int_t *plane*, Int_t *strip*, Int_t *fromevno*, Int_t *toevno*, Int_t *fromtime* = 0, Int_t *totime* = 1000000000, Int_t *fit* = 0)

Input variables:

- Int_t *view*
  Calorimeter view ("view" = 0 means x-view, "view" = 1 means y-view);
- Int_t *plane*
  Calorimeter plane (from 0 to 21);
- Int_t *strip*
  Calorimeter strip (from 0 to 95);
- Int_t *fromevno*
  Starting event number (YODA counter);
- Int_t *toevno*
  Last event number (YODA counter);
- Int_t *fromtime*
  Starting OBT time number. If given the ADC values will be shown as function of time, elsewhere they will be shown as function of event number;
- Int_t *totime*
  Last OBT time number;
- Int_t *fit*
  Flag to tell the program to (1) or not to (0, default) perform an exponential fit.

Example of a standard call:

```
bash> root
root [1] .L CaloADC2MIP.c
root [2] CaloLOOKATSTRIP(0,21,31,780000,830000,19200000,22000000,1);
```

Reads the file CaloADC2MIPdata.raw and draw as function of time strip 32 of x–plane 22. Then fit the result.

```
   bash> root
root [2] CaloLOOKATSTRIP(0,21,31,780000,830000);
```

Reads the file CaloADC2MIPdata.raw and draw as function of event number strip 32 of x–plane 22. Do not fit.

5 Figures

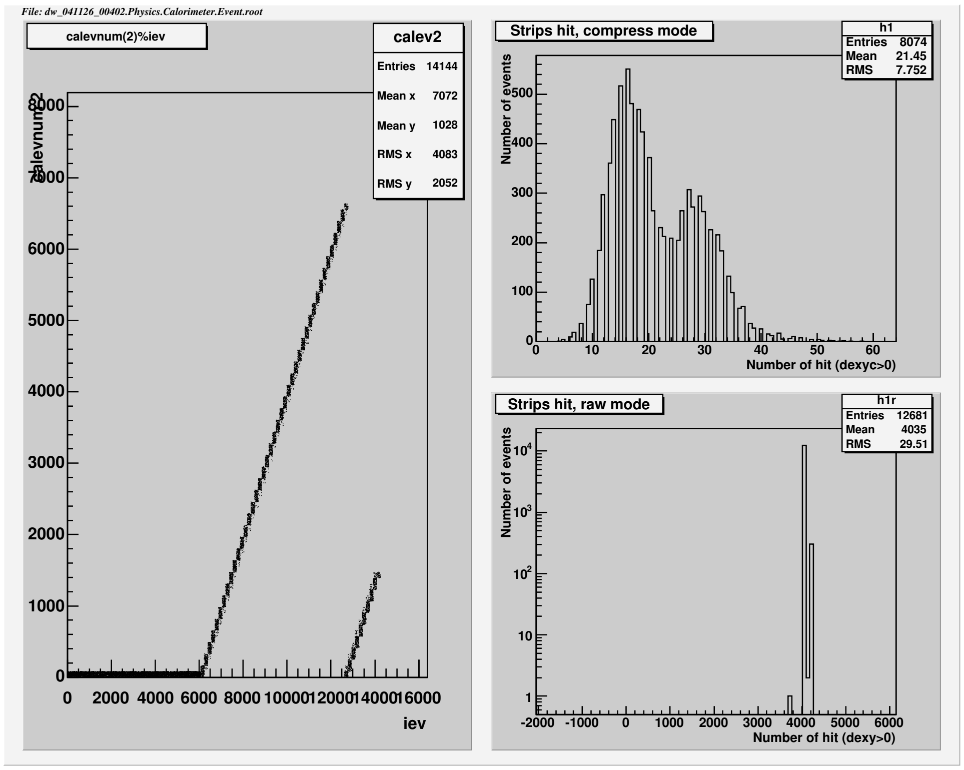


Figure 1: Calorimeter_Detector_Report_1/3. On the left the calorimeter counter is shown as function of the YODA counter. The calorimeter counter remain at zero in the RAW mode acquisition and start to increase when the acquisition is changed at iev of about 6000. At iev of about 13000 there was a DSP reset and the counting started again from one. On the right two panels are shown. In the top panel the strip hit in the COMPRESS mode is reported. The strange "two peaks" behaviour is due to noisy strips due to problems in the power supply during the acquisition of this data file. The same problem caused some latch–up alarm, which means that some views of the calorimeter switched off during the acquisition. This can be noticed in the lower figure were four lines can be noticed instead of the one expected. This means that there were four calorimeter "status" in which there were four different number of strips working (during the acquisition in RAW mode only).

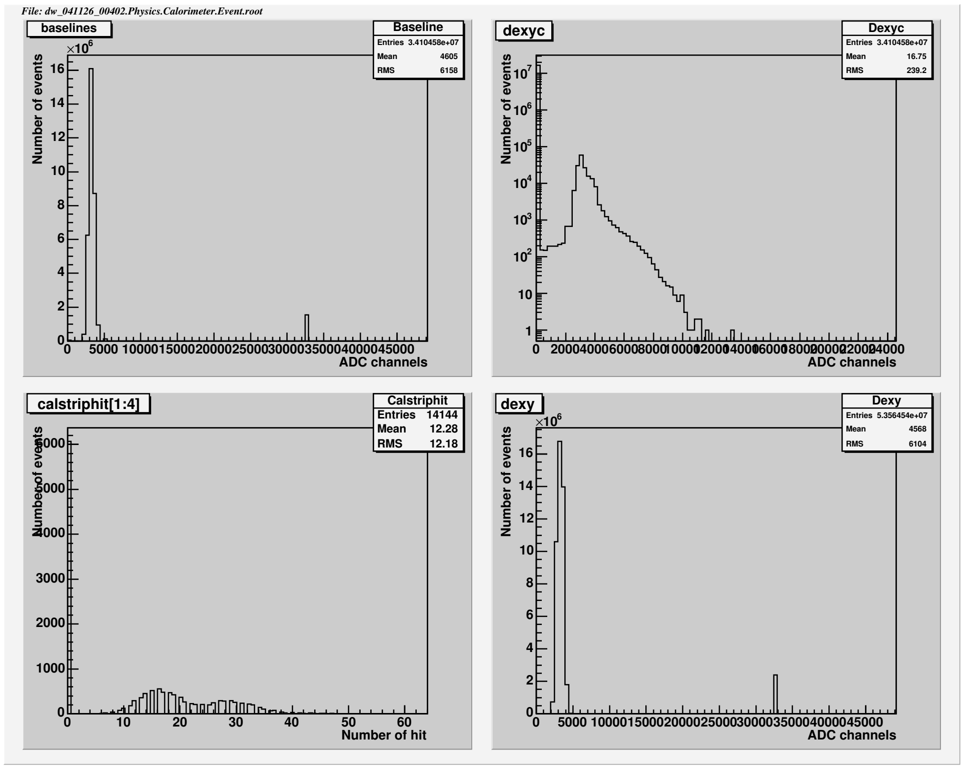
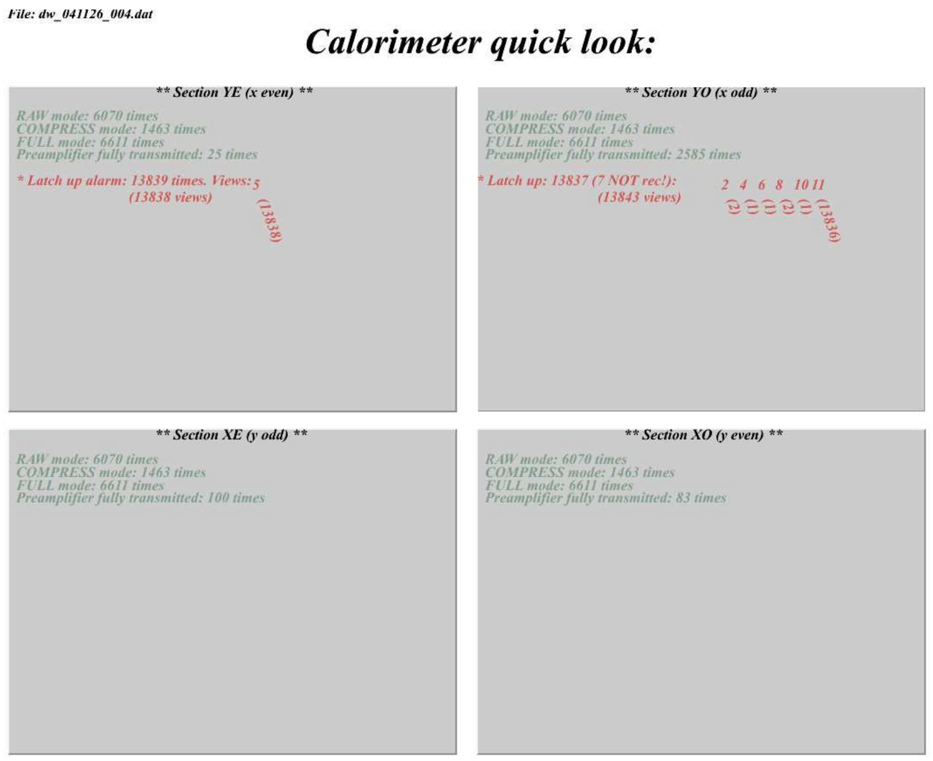


Figure 2: Calorimeter_Detector_Report_2/3. Clockwise from the top right figure: the dexyc dsitribution.Some noisy strips spread the detected signal over a large ADC channel range. Ususally there should be only a peak at zero and a distribution of values between 2000 and 4000 ADC channels. Below there is the dexy distribution. Notice the presence of a peak at about 32000, meaning a latch–up alarm in one or more planes. On the left the strip hit distribution as determined by the DSP; the same behaviour of figure 1 top right can be noticed. On the top left there is the baselines distribution, again the value at about 32000 indicates a latch–up alarm.



File: dw_041126_004.dat

Calorimeter quick look:

** Section YE (x even) **

RAW mode: 6070 times
COMPRESS mode: 1463 times
FULL mode: 6611 times
Preamplifier fully transmitted: 25 times

* Latch up alarm: 13839 times. Views: 5
(13838 views)
(13838)

** Section YO (x odd) **

RAW mode: 6070 times
COMPRESS mode: 1463 times
FULL mode: 6611 times
Preamplifier fully transmitted: 2585 times

* Latch up: 13837 (7 NOT rec!): 2 4 6 8 10 11
(13843 views)
(2) (1) (1) (2) (1) (13836)

** Section XE (y odd) **

RAW mode: 6070 times
COMPRESS mode: 1463 times
FULL mode: 6611 times
Preamplifier fully transmitted: 100 times

** Section XO (y even) **

RAW mode: 6070 times
COMPRESS mode: 1463 times
FULL mode: 6611 times
Preamplifier fully transmitted: 83 times

Figure 3: Calorimeter_Detector_Report_3/3. This is the report sheet of the quick–look. Each section has recorded 6070 events in RAW mode, 1463 events in COMPRESS mode and 6611 events in FULL mode. The section YO was the one which transmitted the higher number of times the preamplifier in raw mode (2585) which is about the 30% of the FULL and COMPRESS mode acquisition. In this file there were some latch–up alarms in sections YE and YO. Notice that in the case of section YE the DSP recorded a latch–up alarm 13839 times while CaloQLOOK only 13838 times, one time less since there are some capacitors which take time to discharge and during the first latch–up event that section recorded some data even if the plane was not powered on. The opposite can be noticed in section YO, where the latch up was recorded 13837 times by the DSP but 13843 times by CaloQLOOK. As discussed in section 2.1 the exceeding latch–up alarm are not real latch–up but an effect of the warming up of that section.

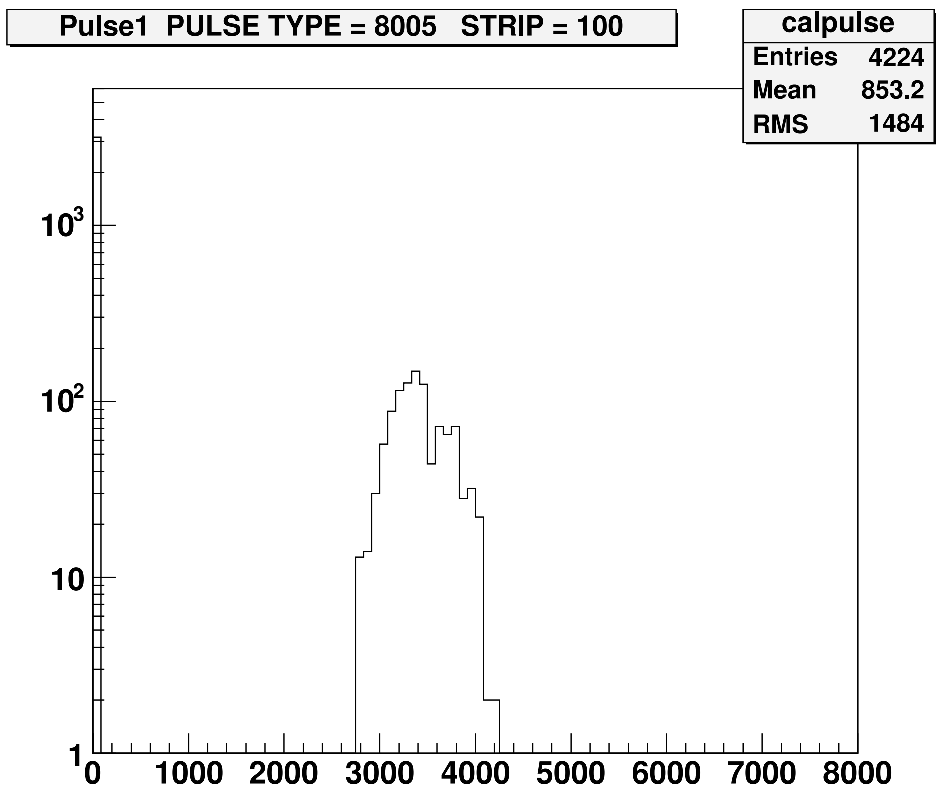


Figure 4: Calorimeter pulse 1 for one of the four sections (the other sections have ADC values at zero). Notice that the pulse type and strip is not correct if more than one calibration exists on a file (known bug).

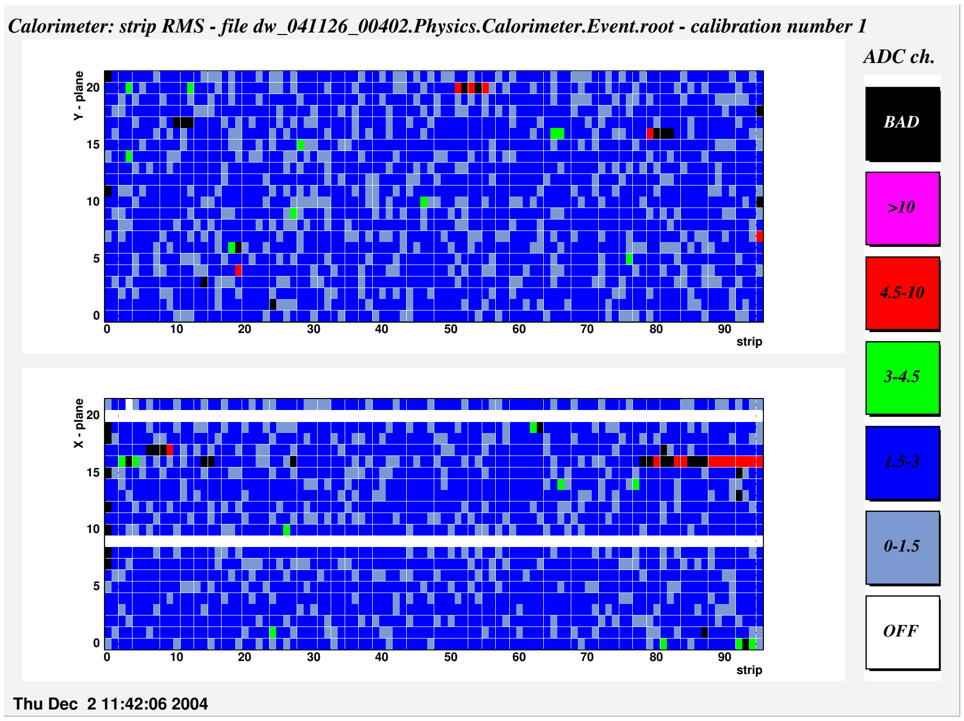


Figure 5: The status at calibration one of file dw_041126_004 (YODA processing number 02). There can be easily notice the two planes of latch–up in the X–view. Moreover the fourth strip of X–plane 21 is off. A part this the calorimeter is working fine and only one noisy preamplifier can be seen, strips from 80 to 96 X–plane number 16.

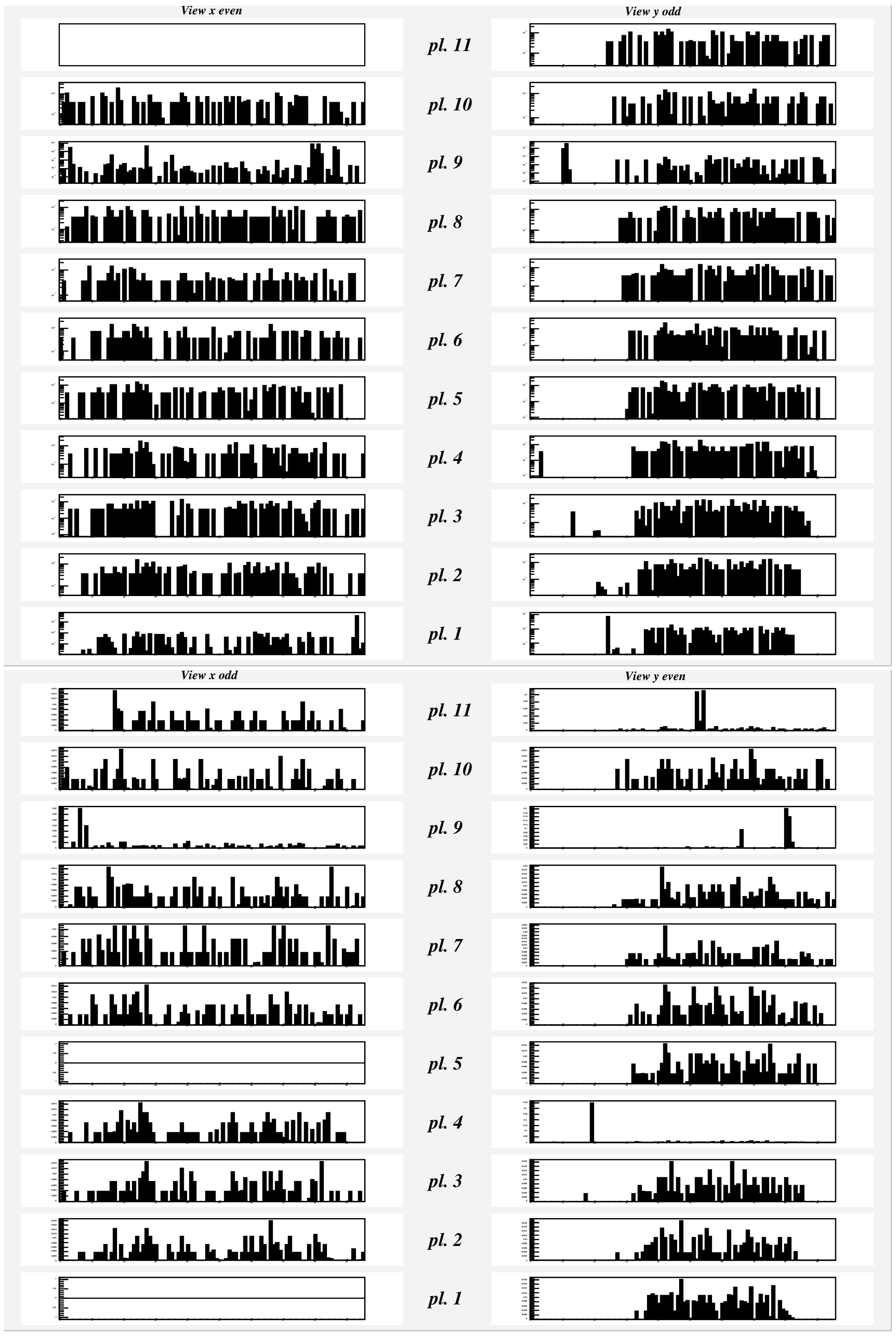


Figure 6: The strip hit for each plane in the case of about 100 events. Notice that in the two upper figures the scale is logarithmic in each panel, while it is linear in the two lower figures. When the number of events is high there should not be any hole in the distributions. Notice that in this case there are three planes with latch–up, x even 11, x odd 5 and x odd 1. Planes with one or more noisy strips like, for example, y even 4 should be seen also in logarithmic scale to see better the signal on the other strips.

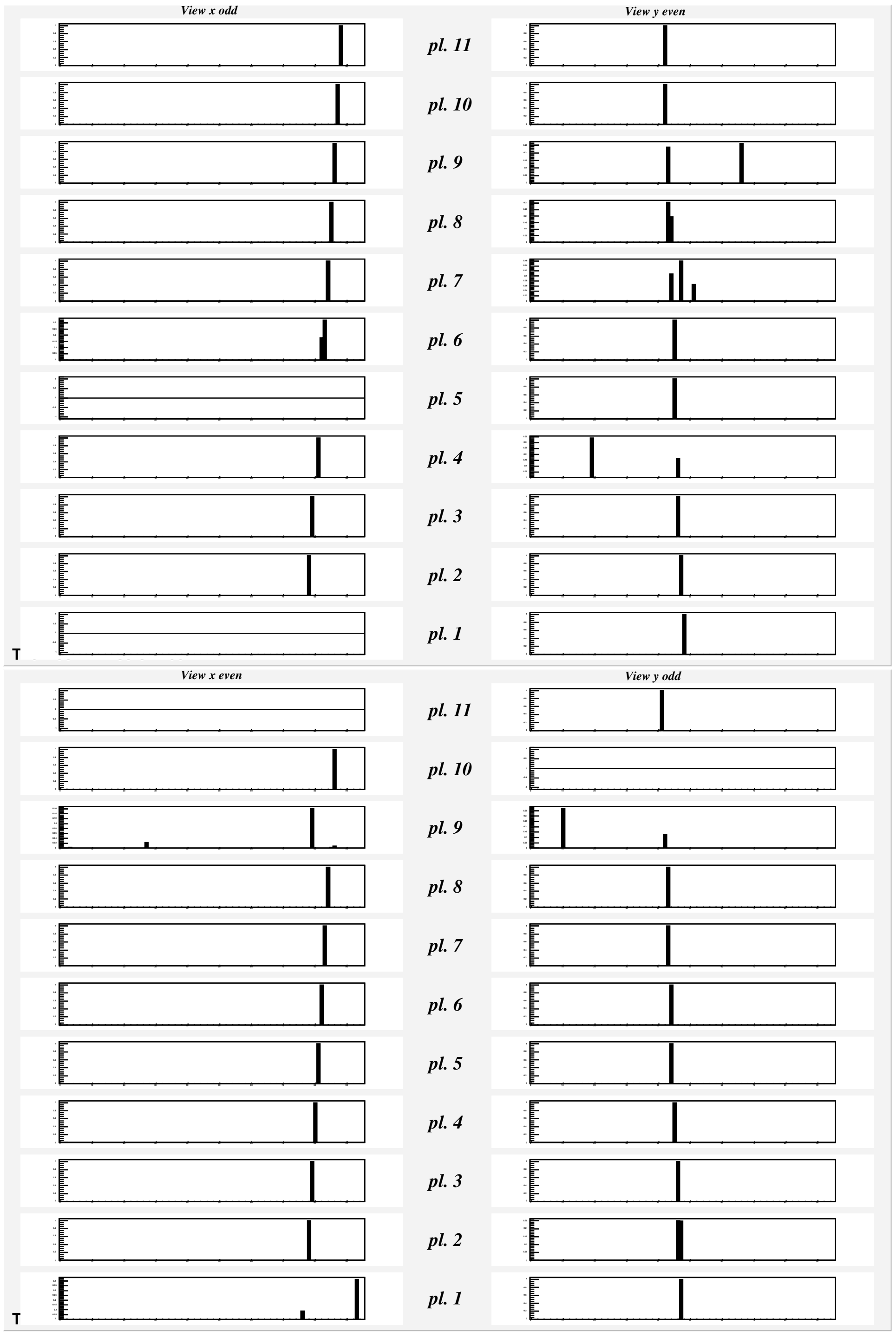


Figure 7: A muon track in the calorimeter as seen by CaloPLANES.

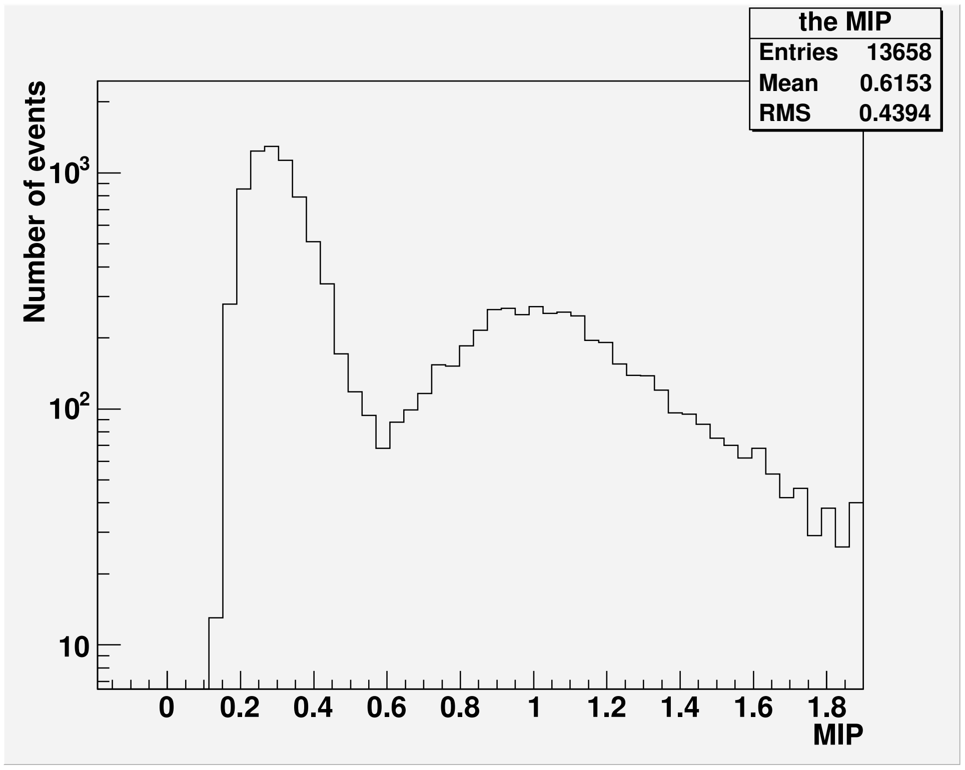


Figure 8: The MIP in the case of about 400 events. The peak on the left is what remain of the pedestal after the compression of data. On the right the average Landau distribution of minimum ionizing particles for all the strips of the calorimeter.

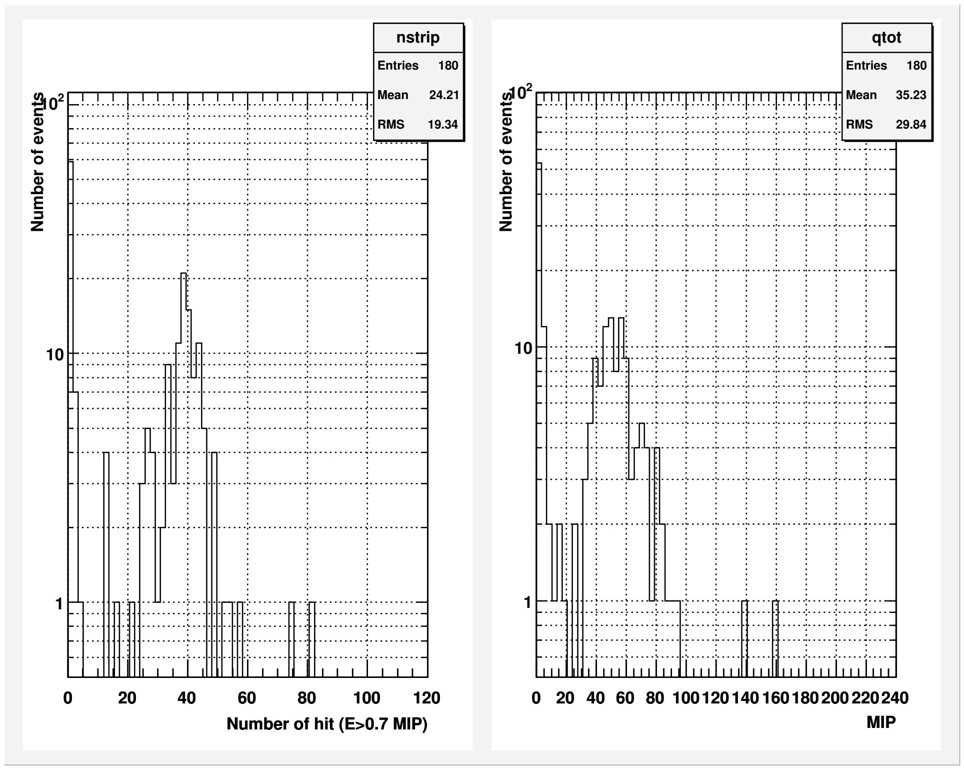


Figure 9: Number of strip hit (nstrip) and energy released (qtot) in the calorimeter for ground muon events. The “nstrip” distribution is peaked at about 41, number of working planes while the “qtot” distribution is peaked at about 50 MIP.

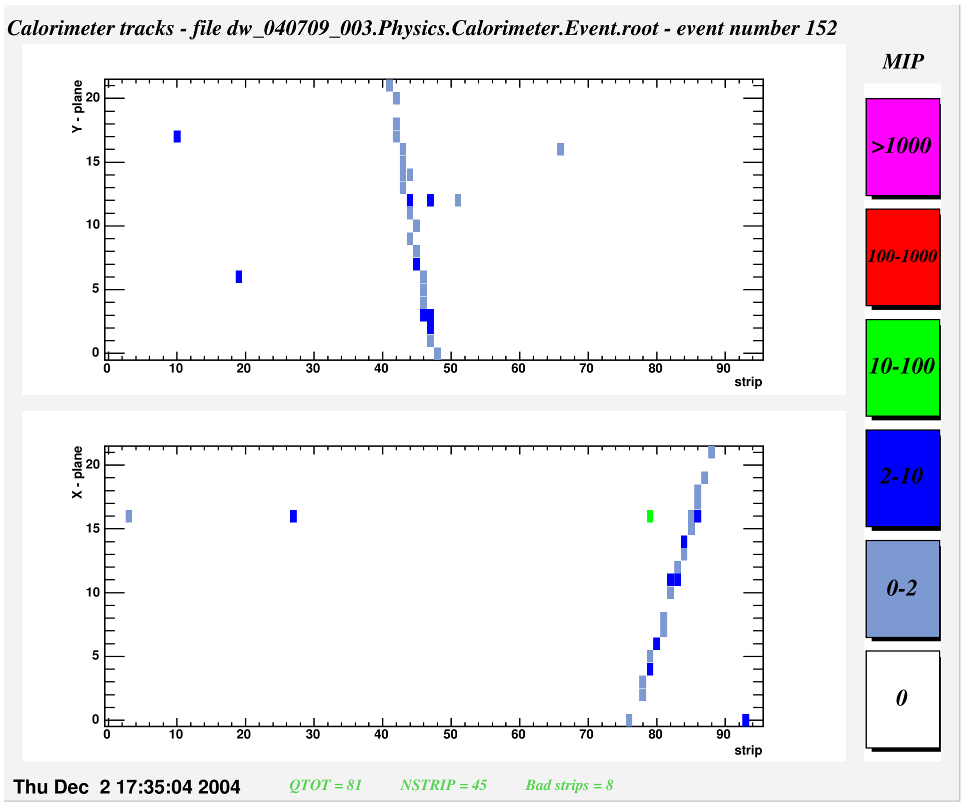


Figure 10: A muon track in the calorimeter as seen by CaloMATRA (the same track of figure 7).

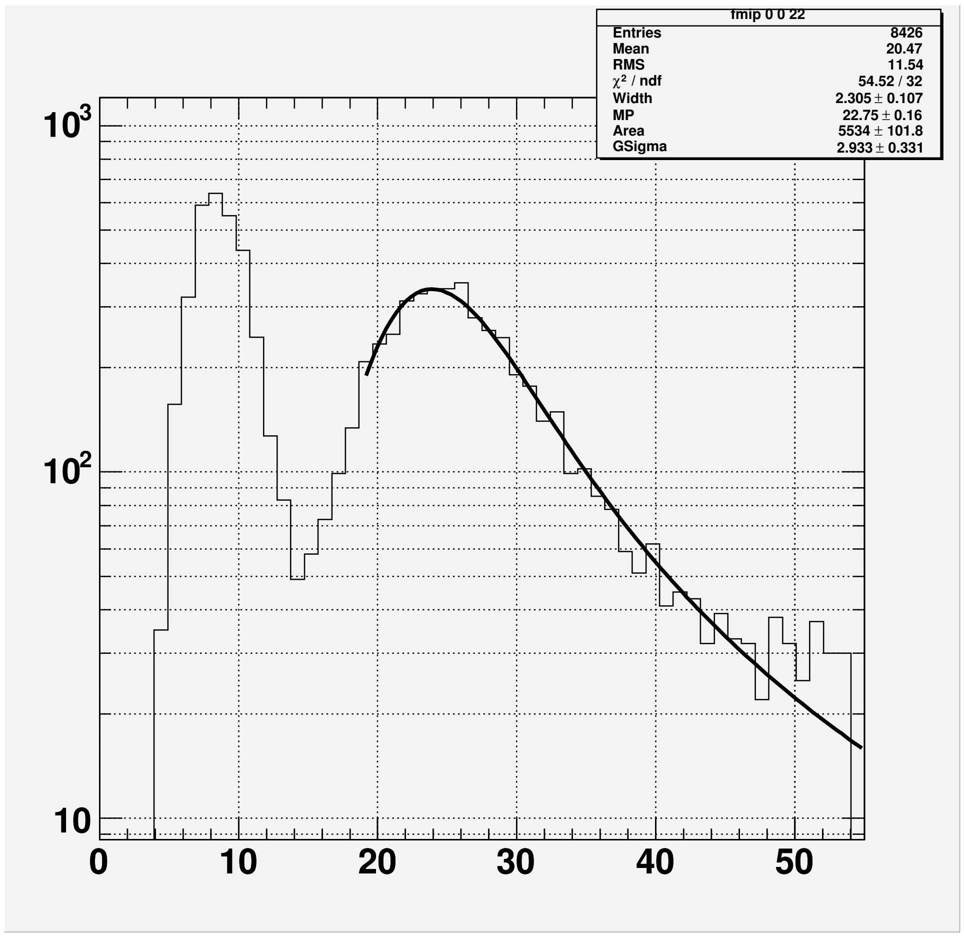
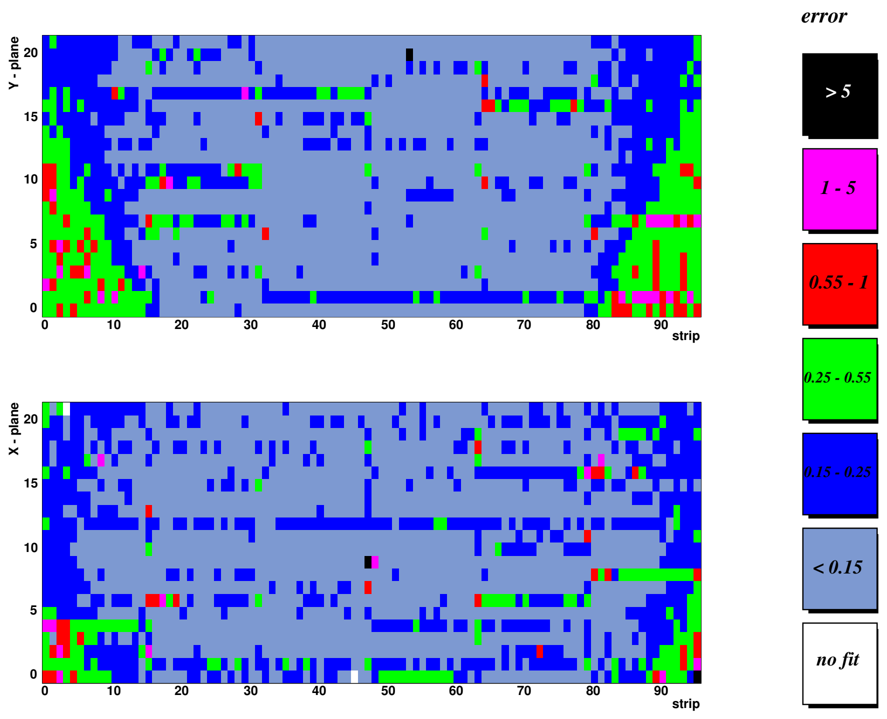


Figure 11: Fit of the data for strip 23 plane 1 x–view.



Tue May 3 15:27:50 2005

Figure 12: Figure showing with colours the error in determining the position of the MIP peak for x (lower panel) and y (upper panel) views.

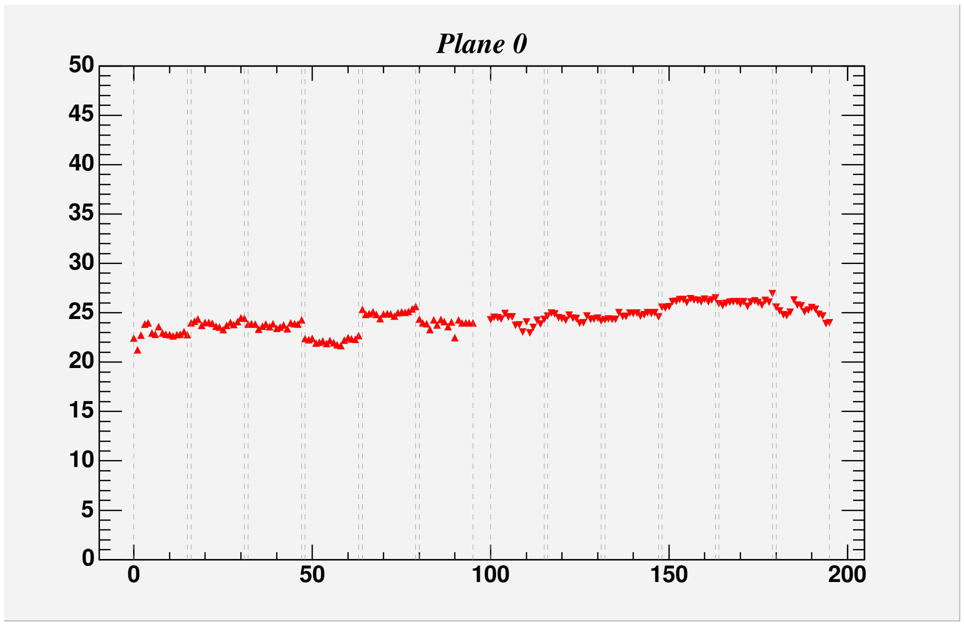


Figure 13: ADC to MIP conversion values for plane 1, x–view from 0 to 95 and y–view from 100 to 195.