MAGIC-TDAS 02-01
020101/OBlanch

Description of the ROOT data format for the MAGIC Telescope

O Blanch

<blanch@ifae.es>

January, 2002

Abstract

Here we intend to describe and fix the format of the data that the MERPP will write to a file as well as the output format after the full Monte Carlo chain. In doing so we will make a brief description of the ROOT file.

Contents

1 Data storage in a single root file

The MAGIC Telescope will have several subsystems like the control active mirror system, the weather station, ... that will produce data as well as the DAQ will do. The later will write to disk in a binary format [1] and the other will use several formats (not all fixed yet).

It has been decided to use ROOT for the analysi, so that it is really worth to translate all this data to root language. This is a work of the socalled MERPP[1], which will read data from all subsystems and write all the data to a single structured root file. On the other hand, it is foreseen to use the same analysis tools for Monte Carlo. Therefore the last modular program in the Monte Carlo simulation writes also a root file that has the same structure than the actual data. To be precise, root files coming from Monte Carlo and Raw data have not exactly the same format since the former has also Monte Carlo information. Inside one root file, there are Trees. Each Tree is an absolutely independent set of data. The information inside one Tree is organized in one or more Branches, which are no more than a certain number of entries of one c++ class. The Branches of one Tree are correlated in a sense that all of them have the same number of entries. While different trees can have different number of entries.

Using that tool of ROOT, the raw data file is going to be written structured in trees. There will be one Tree called RunHeaders, which will contain the general information of one run. This Tree will have as many entries as runs are written in the file. So far, it has been decided to put one run per file and write one file per run. There will also be a Tree for each subsystem data. The exact formats for all of them are not decided yet but it is for the data coming from FADC. Although all raw data from FADC is coming from the same subsystem, there will be distributed in more than one tree. This is due to the fact that they will have really different origin and propose: pedestal runs, real events, calibration ... But inside each tree all of them will have the same structure. It is not defined which raw FADC trees will be in the root file. For sure, we will have the Events tree with the actual events.

In the following chapters we are going to show the internal format for the RunHeaders and for trees with raw FADC data that, for simplicity, we will call Events.

2 ROOT variables translation

Binary	**ROOT**
character	*Char_t*
unsigned integer of 8 bits	*Byte_t*
unsigned integer of 16 bits	*UShort_t*
signed integer of 16 bits	*Short_t*
unsigned integer of 32 bits	*UInt_t*
unsigned integer of 64 bits	*UInt_t*[2]
float of 32 bits	*Float_t*

Table 1: Relation between binary and root variables. Root does not support integers of 64 bits, so that and array of two 32 bits long unsigned integer (*UInt_t*) is used.

Since the binary format for the data [1] that the DAQ writes to disk is already fixed, it has been decided to keep almost the same for the root format. But the variable types had to be translated to root variable type. The following table correlates root and binary types.

[1] MERging an PreProcessing program.

3 Summary of variable names

Some of the described classes are derived from a more general root class that could have its own member variables. These variables are not listed here except for the ones that are relevant for the raw data format.

3.1 RunHeaders Tree

The RunHeaders tree has one Branch with only one entry for each data taking run. The information of the run can be found in this Branch, which is filled with the MRawRunHeader class.

If the root file has raw data, there is only this branch. But for root files that are coming from the Monte Carlo simulation there could be three more branches: MMcRunHeader,MMcFadcHeader, and MMcTrigHeader.

MRawRunHeader class[2]

fMagicNumber	$UShort\_t$
fFormatVersion	$UShort\_t$
fSoftVersion	$UShort\_t$
fRunType	$UShort\_t$
fRunNumber	$UInt\_t$
fProjectName	$Char\_t * 20$
fSourceName	$Char\_t * 10$
fSourceEpochChar	$Char\_t$
fSourceEpochDate	$UShort\_t$
fMJD	$Float\_t$
fDateYear	$UShort\_t$
fDateMonth	$Byte\_t$
fDateDay	$Byte\_t$
fNumCrates	$UShort\_t$
fNumPixInCrate	$UShort\_t$
fNumSamplesLoGain	$UShort\_t$
fNumSamplesHiGain	$UShort\_t$
fNumEvents	$UInt\_t$
fPixAssignment	$*TArrayS$

MMcRunHeader class

fNumTrigCond	$UInt\_t$
fAllEvtsTriggered	$Byte\_t$
fMcEvt	$Byte\_t$
fMcTrig	$Byte\_t$

[2] TArrayS is a root structure. It is an array of short integers (16 bits per element) that has a variable length. Since Root does not support arrays of unknown length and the length of fPixAssignment could change, we need this root class. See Root documentation in root.cern.ch.

fMcFadc	*Byte_t*
fNumAnalisedPixels	*Int_t*
fNumSimulatedShowers	*UInt_t*
fNumStoredShowers	*UInt_t*
fStarFieldRaH	*Byte_t*
fStarFieldRaM	*Byte_t*
fStarFieldRaS	*Byte_t*
fStarFieldDeD	*Byte_t*
fStarFieldDeM	*Byte_t*
fStarFieldDeS	*Byte_t*
fNumPheFromDNSB	*Float_t*
fTelesTheta	*Float_t*
fTelesPhi	*Float_t*
fSourceOffsetTheta	*Float_t*
fSourceOffsetPhi	*Float_t*
fShowerThetaMax	*Float_t*
fShowerThetaMin	*Float_t*
fShowerPhiMax	*Float_t*
fShowerPhiMin	*Float_t*
fCorsikaVersion	*UShort_t*
fReflVersion	*UShort_t*
fCamVersion	*UShort_t*

MMcFadcHeader class

fFadcShape	*Float_t*
fAmplFadc	*Float_t*
fFwhmFadc	*Float_t*
fPedesMean[]	*Float_t*
fElecNoise[]	*Float_t*

MMcTrigHeader class

fTopology	*UShort_t*
fMultiplicity	*UShort_t*
fThreshold[]	*Float_t*
fTrigPattern[2]	*UInt_t*
fTrigShape	*Float_t*
fAmplTrig	*Float_t*
fFwhmTrig	*Float_t*
fOverlapingTime	*Float_t*
fGateLeng	*Float_t*

3.2 Events Tree

The Events tree will have one entry for each event that will have triggered the system. For each of these events, all needed information will be stored in three different branches: MRawEvtHeader, MRawEvtData, and MTime.

For Monte Carlo files, it is worth to save information about the Monte Carlo simulation itself. Therefore, there could be some other branches: MMcEvt and MMcTrig.

MRawEvtHeader class [3]

fDAQEvtNumber	$UInt\_t$
fNumTrigLvl1	$UInt\_t$
fNumTrigLvl2	$UInt\_t$
fTrigPattern[]	$UInt\_t * 2$
fNumLowGainOn	$UShort\_t$
fTrigType	$UShort\_t$
fPixLoGainOn	$*MArrayB$

MRawEvtData class [4]

fHiGainPixId	$*MArrayS$
fHiGainFadcSamples	$*MArrayB$
fLoGainPixId	$*MArrayS$
fLoGainFadcSamples	$*MArrayB$

MTime class

fTimeStamp[2]	$UInt\_t$
fDuration	$UInt\_t$

MMcEvt class

fPartId	$UShort\_t$
fEnergy	$Float\_t$
fTheta	$Float\_t$
fPhi	$Float\_t$
fCoreD	$Float\_t$
fCoreX	$Float\_t$
fCoreY	$Float\_t$
fImpact	$Float\_t$

[3] MArrayB is a MARS structure. See MARS information in magic.uni-sw.gwdg.de/mars/

[4] MArrayS is a MARS structure. See MARS information in magic.uni-sw.gwdg.de/mars/

fPhotIni	*UInt_t*
fPassPhotAtm	*UInt_t*
fPassPhotRef	*UInt_t*
fPassPhotCone	*UInt_t*
fPhotElfromShower	*UInt_t*
fPhotElinCamera	*UInt_t*

MMcTrig class

fNumFirstLevel	*Short_t*
fNumSecondLevel	*Short_t*
fTimeFirst[]	*Float_t*
fPixelsFirst[][]	*Byte_t*
fFirstToSecond	*Short_t*

3.3 Crate Data

The crate information should be consistent within itself and with stored data in MRawEvtHeader and MRawEvtData. A part from that, it is just some hardware information, which we do not want to put with the data. Therefore, it would only be written if some disagreement have been found.

MRawCrateData class

fDAQCrateNumber	*UShort_t*
fFADCEvtNumber	*UInt_t*
fFADCClockTick	*UInt_t*

4 Full description of variable names

Each root file corresponds to a data taking run. Among others, it has one Tree with the general run information (RunHeaders tree) and one Tree for the raw event data (Events tree).

4.1 RunHeaders Data

4.1.1 MRawRunHeader

The MRawRunHeader has the general information about the run conditions.

fMagicNumber *Ushort_t*

It will be constant for all future data files. They allow to identify the file as a MAGIC raw data file. Although this number should have been already checked by MERPP while writing the root file, we will keep it in the root format for further fast identification.

fFormatVersion *UShort_t*

Version of root data format. We introduce this number foreseeing any future change in the root format. The current number for the format version is 2.

fSoftVersion *UShort_t*

Version of the chain of programs that produce the root file. For the current Monte Carlo programs it is 50 [5].

fRunType *UShort_t*

Type of run: normal data run (0), pedestal (1), calibration (2), ... And it would be 256 if it is a Monte Carlo run.

fRunNumber *UInt_t*

Run number. This number starts with 1 at the beginning of a datataking night.

fProjectName $Char\_t * 20$

Code name of the project.

fSourceName $Char\_t * 10$

Source name using 10 characters.

fSourceEpochChar and fSourceEpochDate $Char\_t * 1$, *UShort_t*

Epoch for the source coordinates. SourceEpochChar can be “J” for julian coordinates, “B” for besselian coordinates, “D” for “epoch of date”.

fMJD *Float_t*

Modified julian date up to second accuracy.

fDateYear, fDateMonth, fDateDay *UShort_t*, *Byte_t*, *Byte_t*

Again (UTC) date but this time in more familiar units: year YYYY, month MM and day of the month DD.

fNumCrates *UShort_t*

Number of crates that are stored in this run.

fNumPixInCrate *UShort_t*

Number of pixels in one crate. This variable will have been already used by MERPP to determine the number of pixels, whose information will have been saved by the DAQ system. It is kept still here to allow the identification of corrupted pixel if some crate shows a strange behaviour .

fNumSamplesLoGain, fNumSamplesHiGain *UShort_t*

Number of values saved for each event and used pixel in the high gain and low gain branches. For the time being, it is 15 time slices. In the future, it could be a different or even variable (then we can use 0 as a flag) number of time slices, or (once the behaviour of the detector is well understood) few numbers that describe the event.

[5] Files generated with version 40 or larger can be analysed. From then on, only some fine tunnig of the program has been done and still will be done but for general studies there is not difference.

fNumEvents *UInt_t*

Total number of events in the run.

fPixAssignment **TArrayS*

Pixel assignments. For each channel in the FADC system, we provide here the corresponding spiral pixel number on the camera. This will have been already used to relate the DAQ hardware and the camera hardware and it is kept here for completeness.

4.1.2 MMcRunHeader

This branch will only appear in Monte Carlo files. It has information about the conditions used to run the Reflector and Camera programs.

fNumTrigCond *UInt_t*

It indicates the number of trigger conditions that were analysed and stored in the root file. If its value is 0, it means that a single trigger condition option [2] has been used while running the Camera simulation program. Otherwise, this is the number of trigger conditions that have been checked in the trigger loop mode[2].

fAllEvtsTriggered *Byte_t*

This variable indicates if all the events have triggered the camera or not. It is 1 if all the events in the file have fulfilled at least one of the checked trigger conditions. In the case that events that have not triggered the camera are stored, the value of fAllEvtsTriggered is 0.

fMcEvt *Byte_t*

It is a flag that indicates whether the Monte Carlo information,which would be in the MMcEvt branch, of each event has been stored (1) or not (0).

fMcTrig *Byte_t*

It is a flag that indicates whether the trigger information, which would be in the MMcTrig branch in the Events tree and in the MMcTrigHeader in the RunHeaders tree, has been stored (1) or not (0).

fMcFadc *Byte_t*

It is a flag that indicates whether the trigger information, which would be in the MMcFadcHeader, has been stored (1) or not (0).

fNumAnalisedPixels *Int_t*

The number of the camera pixels that have been analysed.

fNumSimulatedShowers *UInt_t*

fNumStoredShowers *UInt_t*

The number of simulated showers that have been read by the camera program and the number of them that have been stored in the root file [6]

[6] In some very special cases the number of stored showers can be larger than the analysed ones since the meaning of a shower after and before the camera is slightly different. Before the camera, they are just showers which have been generated but after they are events which are read by the FADC, and some of them could be just NSB.

fStarFieldRaH, fStarFieldRaM, fStarFieldRaS *Byte_t*

fStarFieldDeD, fStarFieldDeM, fStarFieldDeS *Byte_t*

The center of the field of view that is used to calculate the light coming from the Stars.

fNumPheFromDNSB *Float_t*

Mean number of photo-electron per ns per inner pixel that are produced due to Diffuse Night Sky Background.

fTelesTheta, fTelesPhi *Float_t*

They are the coordinates where the Telescope is pointing. There exist the option of pointing the telescope always to the shower direction, then fTelesTheta is larger than 180 (200) and fTelesPhi is larger than 360 (400).

fSourceOffsetTheta, fSourceOffsetPhi *Float_t*

Angular distance between the source position in the camera and the camera center. It is worth in the case of wobble mode Monte Carlo simulations[3]

fShowerThetaMax and fShowerThetaMin *Float_t*

fShowerPhiMax and fShowerPhiMin *Float_t*

They are the angular range of the simulated showers. The showers that are generated with a version of Corsika smaller than 6 (see description of fCorsikaVersion below) are distributed in a square region limited by these values. Phi follows a flat distribution while Theta follows $sin\Theta$ distribution.

fCorsikaVersion, fReflVersion and fCamVersion *UShort_t*

Number version for the three main programs in the Magic Monte Carlo. They are the number version of Corsika in which MMCS is based, the number version of the Reflector program, and the number version of the Camera.

4.1.3 MMcFadcHeader

This branch will only appear in Monte Carlo files. It has information about the simulation of the FADC.

fFadcShape *Float_t*

Flag to indicate the shape of the analog signal for a single phe response that goes to the FADC simulation. Currently, 0 is the only valid value and it means that the shape is gaussian.

fAmplFadc, fFwhmFadc *Float_t*

Mean amplitude (mV) and fwhm (ns) of the single photo-electron response.

fPedesMean[] *Float_t*

Mean value of the FADC pedestal for each pixel.

fElecNoise[] *Float_t*

Amplitude in mV of the electronic noise in the FADC chain for each pixel.

4.1.4 MMcTrigHeader

This branch will only appear in Monte Carlo files. It has information about the simulation of the trigger system.

fTopology *UInt_t*

Topology that the simulated event has to fulfill to trigger the camera. Currently there are three different topologies implemented: 0 means central pixel[7], 1 means Closed-pack Next Neighbours[8] and 2 means simple Next Neighbour[9].

fMultiplicity *UInt_t*

Minimum number of fired pixels that must be on at the same time in a given topology (fTopology) to trigger the camera.

fThreshold[] *Floatt_t*

Discriminator threshold in mV for each pixel. The pixel is fired for a fix time gate each time that the analog signal crosses this level.

fTrigPattern[2] *UInt_t*

Used to identify with MRawEvtHeader::Trigger Pattern identification (See section 4.2).

fTrigShape *Float_t*

Flag to indicate the shape of the analog signal for a single phe response that goes to the trigger simulation. Currently, 0 is the only valid value and it means that the shape is gaussian.

fAmplTrig, fFwhmTrig *Float_t*

Mean amplitude (mV) and fwhm (ns) of the single photo-electron response.

fOverlapingTime *Float_t*

Minimum time in ns that two pixels must be fired simultaneously to assume that trigger system will see them fired at the same time. Due to limitations in the simulation programs it has to be a multiple of 0.25 ns, which is the minimum value.

fGateLength *Float_t*

Size in ns of the gate, which indicates that the pixel is fired, that the discriminator opens each time the analog signal of that pixel crosses the discriminator threshold.

4.2 Raw Event Data

4.2.1 MRawEvtHeader

The MRawEvtHeader has the general information about the event.

fDAQEvtNumber *UInt_t*

Event number. This number comes from a counter in the DAQ program.

[7] One fired pixel with N-1 neighbour pixels which are also fired
[8] N neighbour fired pixels that if any of them were not fired the other N-1 would be still neighbours
[9] N neighbour fired pixels

fNumTrigLvl1 and fNumTrigLvl2 $UInt\_t$

First and second level trigger number from a counter in the trigger system. These numbers should always be greater than DAQEvtNumber since the FADC system may lose triggers during its dead time.

fTrigPattern[] $UInt\_t * 2$

Trigger pattern for this particular event. Reports if the trigger is 4fold, 5fold, 4 NN, 4 NN+1 or whatever.

fNumLowGainOn $Byte\_t$

One flag indicating how many of the pixels have low gain signal.

fTrigType $Byte\_t$

Trigger type (normal trigger (0), pedestal (1), calibration (2), ...).

fPixLoGainOn $*MArrayB$

Array which tell you which pixels have data for the low gain channel.

4.2.2 MRawEvtData

Here we find the actual data of the event, which is the FADC samples.

fHiPixId $*MArrayS$

List of the corresponding spiral pixel number on the camera for pixels that have data from the high gain channel.

fHiGainFadcSamples $*MArrayB$

The ACTUAL data. Samples of the pixel signal through the high gain channel in 1/300 MHz time slices. Each sample has 8 bits size. The number of samples is given by dividing the length of fHiFadcSamples over the length of fHiPixId. That data follows the same ordering than pixel IDs in fHiPixId.

fLoPixId $*MArrayS$

This is a list of pixel IDs, in the spiral order on the camera, that have data from the low gain channel.

fLoGainFadcSamples $*MArrayB$

Here we have a list with data of low gain samples of pixels in fLoPixId. That data follows the same ordering than pixel IDs in fLoPixId.

4.2.3 MTime

The MTime branch has the time information of the event.

fTimeStamp[] $UInt\_t * 2$

Absolute time as determined using a GPS clock.

fDuration $UInt\_t$

Validity time of the current event.

4.2.4 MMcEvt

This branch will only appear in the Monte Carlo root files and it has the information about the shower generation and the number of photons that are over after each of the simulation steps.

fPartId *UShort_t*

Type of the incident particle that generates the shower in the atmosphere. The numbering convention is the one that is used by CORSIKA [10]

fEnergy *Float_t*

Energy of the incident particle in GeV.

fTheta and fPhi *Float_t*

Direction of the incident particle.

fCoreD, fCoreX and fCoreY *Float_t*

Distance from the telescope to shower core position and the (x,y) coordinates of this core.

fImpact *Float_t*

Impact parameter for this event. It is define as the shortest distance between the original trajectory of the primary (assumed shower-axis) and the direction where the telescope points to.

fPhotIni *UInt_t*

Number of Cherenkov photons that are produced in the atmosphere due to the shower development.

fPhotAtm *UInt_t*

Number of Cherenkov photons that reach the reflector. Some of the initial photons will be absorbed in the atmosphere before reaching the reflector of our detector.

fPhotRef *UInt_t*

Number of Cherenkov photons that the reflector reflects and reach the camera plane.

fPhotCone *UInt_t*

Number of Cherenkov photons that enter in Winston Cones. Actually, there is not Winston Cone simulation. So that, every photon, which arrives to this step, hits the photo-cathode of the PMT.

fPhotElfromShower *UInt_t*

Number of final photo-electrons from the simulated shower.

fPhotElinCamera *UInt_t*

Number of photo-electrons in the camera. They can come either from the shower itself or from the night sky background.

[10] See the CORSIKA documentation in http://www-ik3.fzk.de/ heck/corsika/

4.2.5 MMcTrig

This branch will only appear in the Monte Carlo root files and has information about the trigger simulation for this event.

fNumFirstLevel *Short_t*

Counter of First Level Trigger for the actual simulated shower. Since there can be more than one trigger event from the same shower plus NSB analysed period. This number will go increasing for events that come from the same shower[11].

fNumSecondLevel *Short_t*

Counter of Second Level Trigger like before but for the second level.

fTimeFirst[] *Float_t*

Array with times, when the first trigger happens.

fPixelsFirst[][] *Byte_t*

Bidimensional array that has a map of the pixels that are on for each of the first level triggers of this event. The first dimension indicates which of the trigger you are looking at and the information about which pixel is in the second dimension. To be precise, each byte has the information about 8 pixels, one per bit, following the spiral ordering. Zero means that the pixel was not fired and one that it was.

fFirstToSecond *Short_t*

First level trigger that also fulfills the second level trigger conditions.

4.3 Crate Data (MRawCrate class)

The crate information should be consistent within itself and with stored data in MRawEvt. A part from that, it is just some hardware information, which we do not want to put with the data. Therefore, it would only be written if some disagreement have been found.

fDAQCrateNumber *UShort_t*

Crate number as determined from a counter in the DAQ program.

fFADCEvtNumber *UInt_t*

Event numbers as determined by an internal counter in this particular FADC crate. This number should be equal to fDAQEvtNumber and all crates should have the same number.

fFADCClockTick *UInt_t*

Tick of the 20 MHz crate clock as determined using a counter in the FADC crate. This tick will reset automatically to 0 every 50ns$*2^{32}$ = 214s.

[11] This variable had a slightly different meaning in camera files generated with version smaller than 5. There, it was just the number of First Triggers for the current shower. This change does not make substantial difference in the analisys of the output files.

References

[1] J. Cortina and T. Coarassa, *Description of the DAQ raw data format*, MAGIC-TDAS 00-09/2.

[2] O. Blanch , *How to use the camera simulation program*, MAGIC-TDAS 01-04/2.

[3] W. Wittek, *A proposed mode of operation for MAGIC*, MAGIC-TDAS 01-05.