

YODAPROFILER (v2r00) MANUAL

Emiliano Mocchiutti
(Emiliano.Mocchiutti@ts.infn.it)
2006/10/31

1) Installing YodaProfiler

YodaProfiler requires YODA, ROOT and MySQL (version greater equal to 4.1.20) to be already installed in the system. The package yodaUtility is not needed anymore.

YodaProfiler provides:

- executable YodaProfiler
- executable R2-D2
- library libGLTables.so
- library libsgp4.so
- header file GLTables.h
- header file sgp4.h
- header GLTablesStruct.h
- bash script retrieve_TLE.sh
- bash scrip install_DB.sh
- sql script PAMELAProductionDB.sql
- this manual

To install the YodaProfiler program follow these step:

- set up the PAMELA environment, as example you can use my setting (it is done for the BASH shell):

```
export PAM_DBHOST=mysql://srv-g2-01.ts.infn.it/pamelaflightnew
export PAM_DBUSER=root
export PAM_DBPSW=CaloTs
export PAM_YODA=/wizard3/pamela/sw/slc4
export PAM_BIN=/wizard3/pamela/sw/slc4/bin
export PAM_LIB=/wizard3/pamela/sw/slc4/lib
export PAM_SRC=/wizard3/pamela/sw/slc4/src
export PAM_INC=/wizard3/pamela/sw/slc4/inc
export PAM_MACROS=/wizard3/pamela/sw/slc4/macros
export PAM_DOC=/wizard3/pamela/sw/slc4/docs
export PAM_CALIB=/wizard3/pamela/sw/calib
export ARCH=`uname`
```

the PAM_DB* variables can be omitted, their use is explained below.

- download the YodaProfiler source code
- enter the directory YodaProfiler
- give the command

```
make distclean all upgrade
```

- create the DB, to do so enter the directory “docs” and give the command:

```
./install_DB.sh --user=root --host=mysql://localhost/pamelaprod --psw=CaloTs
```

where as input you must give username and password of a MySQL user who have write permissions on the DB. The host must be the computer running the MySQL server the name of the DB (“pamelaprod” in this example) can be any.

Wait 30 seconds and the DB will be created.

If you have set up the PAM_DB* variables you can just give the command:

```
./install_DB.sh
```

if you give the input variables the environmental variables will be overridden.

WARNINGS:

*1) if a DB already exists with that name it will be dropped and recreated, **all DB data will be lost!** You will have 30 seconds to interrupt the procedure before deleting the DB.*

*2) **NO ERROR OR WARNINGS MUST be issued by this procedure.** If you experience any problem first check that you are using a MySQL version newer or equal to 4.1.20, than check that you have the InnoDB engine active (look in the file /etc/my.cnf , the “skip-innodb” must be commented or absent). Notice that if you need to include “CURRENT_TIMESTAMP” between apices in the PAMELAProductionDB.sql file you are probably using a wrong MySQL version, the YodaProfiler will NOT work properly in that case.*

The installation is completed.

2) YodaProfiler outlook

YodaProfiler has been written assuming that input files does not contain repeated events. Some patches have been implemented to be able to process YODA files which, at present, have repetitions of events. The main patch try to find the repetition of events at the end of, almost, any file and discard the repeated event from the processing.

The task of this program is to fill the PAMELA DB with RUN and calibration informations in order to allow the data/calibration association using the absolute time. The DB scheme is shown in the figure attached to this document. The DB structure has been developed around the main table GL_RUN which contains the informations about the PAMELA runs. The GL_NAME_CALIB tables allows the association between data and calibration for calorimeter, tracker and S4 (where NAME=CALO,TRK,S4).

YodaProfiler can be run on the files in any order but some small differences between two DB created processing files with a different order could be noticed due to the patches implemented to handle repeated events.

The program uniquely identifies the runs using the three variables OBT, PSCU packet number and the boot number. To find the runs input files are scanned looking for runtrailers. Once a runtrailer has been found the program search for the corresponding runheader and check if its PSCU packet counter is correct, this is done subtracting to the runtrailer packet number the number of events recorded in that run as stated in the

word "PKT_COUNTER" saved in the runtrailer. If the packet number correspond the run is saved in the GL_RUN table. If the check is wrong and in case of any problem (like missing runheader or missing runtrailer or full mass memory, etc.) the program check for a certain group of physics events if any packet of different type has been saved in between. This way YodaProfiler can distinguish also run with missing runheader and runtrailer.

A dedicated routine handle runs which have been transmitted in two different downloads. That runs are temporary stored in the GL_RUN_FRAGMENTS table (a clone of the GL_RUN table) till the corresponding piece has been found. In that moment the two pieces of run are moved in the GL_RUN table and stored as two different entries. The link between the two pieces is the variable ID_RUN_FRAG which is cross set to the ID of the other piece while is zero for "normal" runs.

Pieces of runs for which it is not possible to find the corresponding part remains in the GL_RUN_FRAGMENTS table. YodaProfiler contains a routine which can be used to clean the fragment table moving pieces of run in the GL_RUN table. These runs can be identified in the GL_RUN table looking at the ID_RUN_FRAG flag which is, in this case, equal to the ID of the run.

Another routine is called by the program to validate runs, that is to set to 1 the flag VALIDATION in the GL_RUN table in case the run is associated to the correct tracker calibration.

These two last routine are disabled by default and can be enabled giving a running input. The validation and the cleaning of the fragment table are performed on DB entries older than a certain time set by the user. **Notice that this feature must be used very carefully since if a file is processed and creates a run inserted in a already validated region the run WILL NOT be validated and if a piece of run has been moved from the fragment table to the GL_RUN table it WILL NOT recognized as a run fragment (a piece of run will be lost). Hence the validation must be performed only when sure that there are no more files to be processed for a certain period.** If there is the need to process an old file or to reprocess a file uses the "-force" flag which will remove the file from DB, process it, move run fragments to the run table and validate runs belonging to the processed file.

Other flags allow to remove a file and all what follows from the DB, to validate a single file, to clean fragments for a single file and to avoid storing runs in the fragments table.

When a file is removed from the DB entries in GL_RAW, GL_RUN, GL_TIMESYNC, GL_RUN_FRAGMENTS and GL_RUN_TRASH are deleted, runs in the GL_RUN table are moved to the GL_RUN_TRASH table where also the name of the root file is saved, this way is possible to track already level2 processed runs which have been deleted for any reason.

The RUN ID is univoquely assigned to each run, reprocessing a file will generate new run ids. R2-D2 can be used to associated old to new runs.

It is possible to run YodaProfiler twice on the same file, no change will be made to the DB. A temporary patch has been written to handle the processing of two different files containing the same download. The program will try to choose the less corrupted runs by looking at the number of events contained in the run compared to the number of event saved by the CPU in the runtrailer. Notice that physics events are not scanned to check their integrity. Hence some difference can arise depending on the order on

which the YodaProfiler is run on two files with the same download. The best would be to run first on the good file and in case on the bad one. Doing the opposite in theory could create the following situation: a run contains a lot of physics packets with errors but all the events of the run are present. In this case when running on the good file the program will find an already inserted run and since the number of events is equal it will skip it even if all data inside that run are good!

As already said in this section, the program requires the boot number and needs to calculate the absolute time. Boot number is by default determined looking in the VarDump data. If VarDump data are missing or corrupted the program exit with error. To process the file it will be necessary determine by hand the boot number and give it as input on the command line to the program. To determine the absolute time three informations are needed: the Resurs time zero, which is stored in the table GL_RESURS_OFFSET, the Resurs time sync and the CPU OBT at the time sync. The program try to find these two last informations in the processed file looking for: the timesync macrocommand, the timesync saved in a runheader, the timesync saved in a runtrailer, the timesync that can be found in the inclination macrocommands. NOTICE that at the first occurrence the program will stop and use what it has found, hence the absolute time determination in the DB is not the best it can be obtained by data but it is good enough to allow a univoque association between data and calibration. Timesync informations are stored for each file in the table GL_TIMESYNC. The column "TYPE" indicates from where the information was extracted, 55 = mcmd timesync, 20 runheader, 21 runtrailer, 666 mcmd inclination, 999 given as input to the program. If the program is not able to determine the timesync it will exit with error. It is possible in that case to determine by hand the timesync and give it as input on the command line to the program in order to process the file.

GL_RAW	
+ID:	unsigned int (32 bit)
+PATH:	varchar (255)
+NAME:	varchar (255)
+BOOT_JNUMBER:	unsigned int (16 bit)
+INSERT_TIME:	timestamp

GL_ROOT	
+ID:	unsigned int (32 bit)
+ID_RAW:	unsigned int (32 bit)
+ID_ROOT_L0:	unsigned int (32 bit)
+PATH:	varchar (255)
+NAME:	varchar (255)
+BOOT_JNUMBER:	unsigned int (16 bit)
+INSERT_TIME:	timestamp

GL_TIMESYNC	
+ID:	unsigned int (32 bit)
+ID_RAW:	unsigned int (32 bit)
+OBJID:	unsigned int (32 bit)
+TIMESYNC:	unsigned int (32 bit)
+TYPE:	unsigned int (16 bit)

GL_RESURS_OFFSET	
+ID:	unsigned int (32 bit)
+FROM_ORBIT:	unsigned int (32 bit)
+SPECIAL_FILE:	varchar (255)
+OFFSET_DATE:	timestamp
+INSERT_TIME:	timestamp

__RUNID_GEN	
+ID:	unsigned int (32 bit)

GL_RUN / GL_RUN_FRAGMENTS	
+ID:	unsigned int (32 bit)
+ID_RUN_FRAG:	unsigned int (32 bit)
+ID_ROOT_L0:	unsigned int (32 bit)
+ID_ROOT_L2:	unsigned int (32 bit)
+RUIHEADER_TIME:	unsigned int (32 bit)
+RUIHEADER_TABLE:	unsigned int (32 bit)
+RUIHEADER_OBJT:	unsigned int (32 bit)
+RUIHEADER_OBJT:	unsigned int (32 bit)
+RUIHEADER_PNT:	unsigned int (32 bit)
+BOOT_JNUMBER:	unsigned int (16 bit)
+EV_FROM:	unsigned int (32 bit)
+EV_TO:	unsigned int (32 bit)
+EVENTS:	unsigned int (32 bit)
+PNT_COUNTER:	unsigned int (32 bit)
+PNT_READY_COUNTER:	unsigned int (32 bit)
+COMPLETIONTIMESLAMP:	unsigned int (32 bit)
+AV_WRN_SCHEDULE:	unsigned int (8 bit)
+EFF_WRN_SCHEDULE:	unsigned int (8 bit)
+PRH_VAR_TRK_JUDGE_A:	unsigned int (32 bit)
+PRH_VAR_TRK_JUDGE_B:	unsigned int (32 bit)
+ACQ_BUILD_DURO:	unsigned int (32 bit)
+ACQ_VAR_DURO:	unsigned int (16 bit)
+ROLL_ACQ_AFTER_CALIB:	unsigned int (8 bit)
+ROLL_ACQ_SETTING_MODE:	unsigned int (8 bit)
+TRK_CALIB_USED:	unsigned int (32 bit)
+CALL_DS_TASK:	unsigned int (8 bit)
+LAST_TIMESYNC:	unsigned int (32 bit)
+OBT_TIMESYNC:	unsigned int (32 bit)
+VALIDATION:	unsigned int (8 bit)
+INSERT_TIME:	timestamp
+FILEFIELD:	varchar (255)
+FIELD1:	varchar (255)
+FIELD2:	varchar (255)

GL_TLE	
+ID:	unsigned int (32 bit)
+TLE1:	varchar (255)
+TLE2:	varchar (255)
+TLE3:	varchar (255)
+FROM_TIME:	timestamp
+INSERT_TIME:	timestamp
+NOTE:	varchar (255)

GL_PARAM	
+ID:	unsigned int (32 bit)
+PATH:	varchar (255)
+NAME:	varchar (255)
+FROM_TIME:	unsigned int (32 bit)
+TO_TIME:	unsigned int (32 bit)
+TYPE:	unsigned int (16 bit)
+DESCR:	varchar (255)
+INSERT_TIME:	timestamp

GL_CALO_CALIB	
+ID:	unsigned int (32 bit)
+ID_ROOT_L0:	unsigned int (32 bit)
+EV_ROOT:	unsigned int (32 bit)
+FROM_TIME:	unsigned int (32 bit)
+TO_TIME:	unsigned int (32 bit)
+SECTION1:	unsigned int (8 bit)
+OBJT:	unsigned int (32 bit)
+PNT:	unsigned int (32 bit)
+BOOT_JNUMBER:	unsigned int (16 bit)
+VALIDATION:	unsigned int (8 bit)

GL_TRK_CALIB	
+ID:	unsigned int (32 bit)
+ID_ROOT_L0:	unsigned int (32 bit)
+EV_ROOT_CALIBTRK1:	unsigned int (32 bit)
+EV_ROOT_CALIBTRK1:	unsigned int (32 bit)
+FROM_TIME:	unsigned int (32 bit)
+TO_TIME:	unsigned int (32 bit)
+OBJT:	unsigned int (32 bit)
+PNT1:	unsigned int (32 bit)
+OBJT:	unsigned int (32 bit)
+PNT2:	unsigned int (32 bit)
+BOOT_JNUMBER:	unsigned int (16 bit)
+VALIDATION:	unsigned int (8 bit)

GL_SI_CALIB	
+ID:	unsigned int (32 bit)
+ID_ROOT_L0:	unsigned int (32 bit)
+EV_ROOT:	unsigned int (32 bit)
+FROM_TIME:	unsigned int (32 bit)
+TO_TIME:	unsigned int (32 bit)
+OBJT:	unsigned int (32 bit)
+PNT:	unsigned int (32 bit)
+BOOT_JNUMBER:	unsigned int (16 bit)

PAMELA DB Structure
E. Mocchiutti 2006/10/31

DB time is expressed in seconds starting from 1970-01-01 00:00:00 UTC. YodaProfiler must be used to fill the GL_TLE table needed to analyze orbital informations with DarthVader. To do so use the -tle tlefile option. To retrieve the TLEs from spacetrack.com a "pamela" account has been created, username "pam2006", password "Resurs05". A script is provided to retrieve the TLEs automatically, you can find it in the "docs" directory, it is called "retrieve_TLE.sh". Can be used this way:

```
./retrieve_TLE.sh
```

it will create a file called "tle.txt" that can be used directly with YodaProfiler. Use the "--help" option to know more options of this script.

A file called "tle_061017.txt" can be used as example for the TLE filling, it contains TLE till October 31st 2006.

3) Running YodaProfiler

The program build an executable which is called YodaProfiler. For online help give the --help option which will print:

Usage:

```
YodaProfiler [options] -rawFile raw_filename -yodaFile yoda_filename
```

```
-rawFile      full path to the raw file
-yodaFile     full path to the YODA file
```

Options can be:

```
--version     print informations about compilation and exit

-h | --help   print this help and exit
-v | --verbose be verbose [default]
-s | --silent print nothing on STDOUT
-g | --debug  be very verbose [default: no]

-boot number   CPU boot number [default = taken from VarDump]
-tsync number  timesync (s) [default = taken from data]
-obt0 number   obt at timesync (ms) [default = taken from data]
-clean number  number in seconds after which the fragment table
               can be cleaned and runs validated [default = -1 do not clean],
               if 0 force cleaning immediatly, if negative do not clean
-remove file   remove file and all related runs and calibrations from DB
               file must be the YODA filename (full path is not needed)
               'same' can be used if in conjunction with -yodaFile
-validate file validates runs between the two closest calibration to file
               not belonging to file itself. File must be the YODA filename
               'same' can be used if in conjunction with -yodaFile
-cleanfrag file clean run fragmenst for file only
               File must be the YODA filename
               'same' can be used if in conjunction with -yodaFile
-nofrag       do not leave runs in the fragment table and look for fragments
               in the GL_RUN table.
-force       to be used to reprocess a file or to process a file
               when already validated the surroundings, it is equivalent to:
               -remove same -validate same -cleanfrag same -nofrag
-host       name for the host
               [default = $PAM_DBHOST or mysql://localhost/pamelaprod]
-user       username for the DB [default = $PAM_DBUSER or "anonymous"]
-psw       password for the DB [default = $PAM_DBPSW or ""]
-tle <file> ascii file containing TLE obtained from celestrak.org or
               space-track.org [default = no]
```

The order of input files and options does not matter.

Example:

```
YodaProfiler -yodaFile /path/to/rawreader/files/00000_000_000_cln2.pam -rawFile /
```

```
path/to/filesfromyoda/00000_000_000_cln2.root -v
```

The standard call will be like the one in the example. Notice that if you have set the PAM_DB* environmental variables they will be used for the DB connection. If they are missing and no input is given, the default will be host=mysql://localhost/pamelaprod, user=anonymous, psw="". The DB connection can be controlled using the input flags -host, -user and -psw. Notice that these flags will override the environmental variables if set.

WARNING: a user with write permissions on the DB must be used!

The standard output of the program will be:

```
Welcome to the PAMELA YodaProfiler, version v2r00

1 => Initialize and open SQL connection
2 => Insert a RAW file in GL_RAW
3 => Update a single GL_RAW record with its BOOT_NUMBER
4 => Insert an entry in GL_TIMESYNC
5 => Insert unpack ROOT file in GL_ROOT
6 => Scan physics and store runs in the GL_RUN table
7 => Insert calorimeter calibrations in the GL_CALO_CALIB table
8 => Insert tracker calibrations in the GL_TRK_CALIB table
9 => Insert S4 calibrations in the GL_S4_CALIB table
13 => Free objects and close SQL connection

Finished, exiting...
```

If any error is issued it will be printed before exiting.

5) Running R2-D2

This is a utility program. Can be used to:

a) Given the run number determine in which YODA file it is contained:

```
|Emi@srv-g2-01 ~>R2-D2 -idRun 1

Run 1 belongs to file /gpfs/wizard/flight/filesfromyoda/001_001_01177_cln2.root
```

b) Given a YODA file determine which runs are contained in it:

```
|Emi@srv-g2-01 ~>R2-D2 -filename 001_001_01177_cln2.root

File 001_001_01177_cln2.root contains the following runs:

=> ID = 1 _-_-_ the run started at 2006-08-30 10:16:07 UTC ended at 2006-08-30
10:46:07 UTC

=> ID = 2 _-_-_ the run started at 2006-08-30 10:46:08 UTC ended at 2006-08-30
10:59:39 UTC

=> ID = 3 _-_-_ the run started at 2006-08-30 11:00:43 UTC ended at 2006-08-30
11:02:23 UTC

=> ID = 4 _-_-_ the run started at 2006-08-30 11:02:23 UTC ended at 2006-08-30
11:32:23 UTC

=> ID = 5 _-_-_ the run started at 2006-08-30 11:32:24 UTC ended at 2006-08-30
11:48:12 UTC

=> ID = 6 _-_-_ the run started at 2006-08-30 11:48:15 UTC ended at 2006-08-30
12:18:15 UTC

=> ID = 7 _-_-_ the run started at 2006-08-30 12:18:15 UTC ended at 2006-08-30
12:24:58 UTC
```

```

=> ID = 8 _-_- the run started at 2006-08-30 12:24:59 UTC ended at 2006-08-30
12:26:19 UTC

=> ID = 9 _-_- the run started at 2006-08-30 12:26:20 UTC ended at 2006-08-30
12:33:40 UTC

=> ID = 10 _-_- the run started at 2006-08-30 12:34:43 UTC ended at 2006-08-30
12:36:23 UTC

=> ID = 11 _-_- the run started at 2006-08-30 12:36:24 UTC ended at 2006-08-30
13:06:24 UTC

=> ID = 12 _-_- the run started at 2006-08-30 13:06:24 UTC ended at 2006-08-30
13:36:24 UTC

=> ID = 3199 _-_- the run started at 2006-08-30 13:36:25 UTC ended at 2006-08-30
13:48:06 UTC

```

```

File 001_001_01177_cln2.root belongs to raw data file /
gpfs/wizard/flight/data//001_001_01177_cln2.pam

```

c) Convert a DB time (in seconds) into a date using a certain time zone (default MSK):

```

|Emi@srv-g2-01 ~>R2-D2 -convert 1156936567 -tzone UTC

DB time 1156936567 is 2006-08-30 10:16:07 UTC

```

d) Search for a run containing a given date in a certain time zone (default MSK):

```

|Emi@srv-g2-01 ~>R2-D2 -runat "2006-08-30 10:16:07" -tzone UTC

Date 2006-08-30 10:16:07 UTC (DB time 1156936567 ) is contained in run 1

```

e) Search for a run contained in a given DB time:

```

|Emi@srv-g2-01 ~>R2-D2 -runatDB 1158091366

DB time 1158091366 is contained in run 37

```

f) Given a root file and an OBT to convert it to DB time and a date:

```

|Emi@srv-g2-01 ~>R2-D2 -tsfile 01400_004_001_cln2.root -obt 12000

OBT 12000 in the file 01400_004_001_cln2.root corresponds to DBtime 1157036668
and date 2006-08-31 17:04:28 UTC

```

g) To print on the screen and save as tle.txt file the TLE for a given date:

```

|Emi@srv-g2-01 ~>R2-D2 -dumpTLEfor "2006-08-15 12:00:00"

Date 2006-08-15 12:00:00 UTC (DB time 1155636000 ) is contained in TLE 237

RESURS DK-1
1 29228U 06021A 06227.37047658 .00002168 00000-0 48562-4 0 2077
2 29228 069.9354 262.4321 0167536 034.2679 326.9166 15.32012453 9367

TLE has been dumped in file tle.txt

```

These time zones are (at the moment) recognized:

- 1) MSK – Moscow Winter Time
- 2) MSD – Moscow Summer Time

- 3) UTC (or GMT) – Coordinated Universal Time
- 4) CET – Central European Time
- 5) CEST – Central European Summer Time