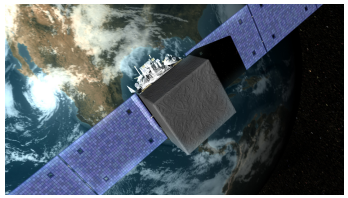
Ft2Util_2: code review

Giacomo Vianello
(CIFS/SLAC)



Goal 1: *fake*[1] FT2 file

Pointing and position
(Magic 7 file)

ft2Util_2

Fake FT2 file

Recon

Magic 7 file: a text file:

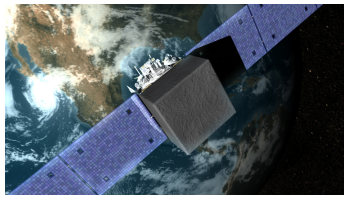
- Every line is a ATT (pointing) or ORB (position) message
- ATT @ 5 Hz, ORB @ 1 Hz
- Pointing and position are known exactly once per second (concurrent ATT and ORB message)

(see the main page of the Doxygen documentation for more details)

- In the ideal case, this is basically just re-**organizing** the information contained in the Magic 7 file

- If there are gaps in the M7 file, this could involve interpolating or extrapolating some quantities.

[1]The fake FT2 file contains only pointing and position information (LIVETIME=0 for every entry). It is requested by L1 at the *crumb* level, and it is used as input to *Recon*.



Goal 1: *fake* FT2 file

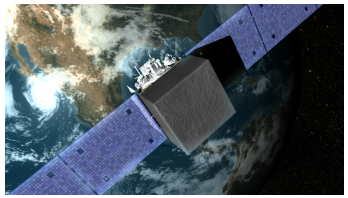
Steps:

•Parse the command line (Anyoption), get the configuration parameters (Configuration), instantiate the Ft2 class (makeFT2.cpp)

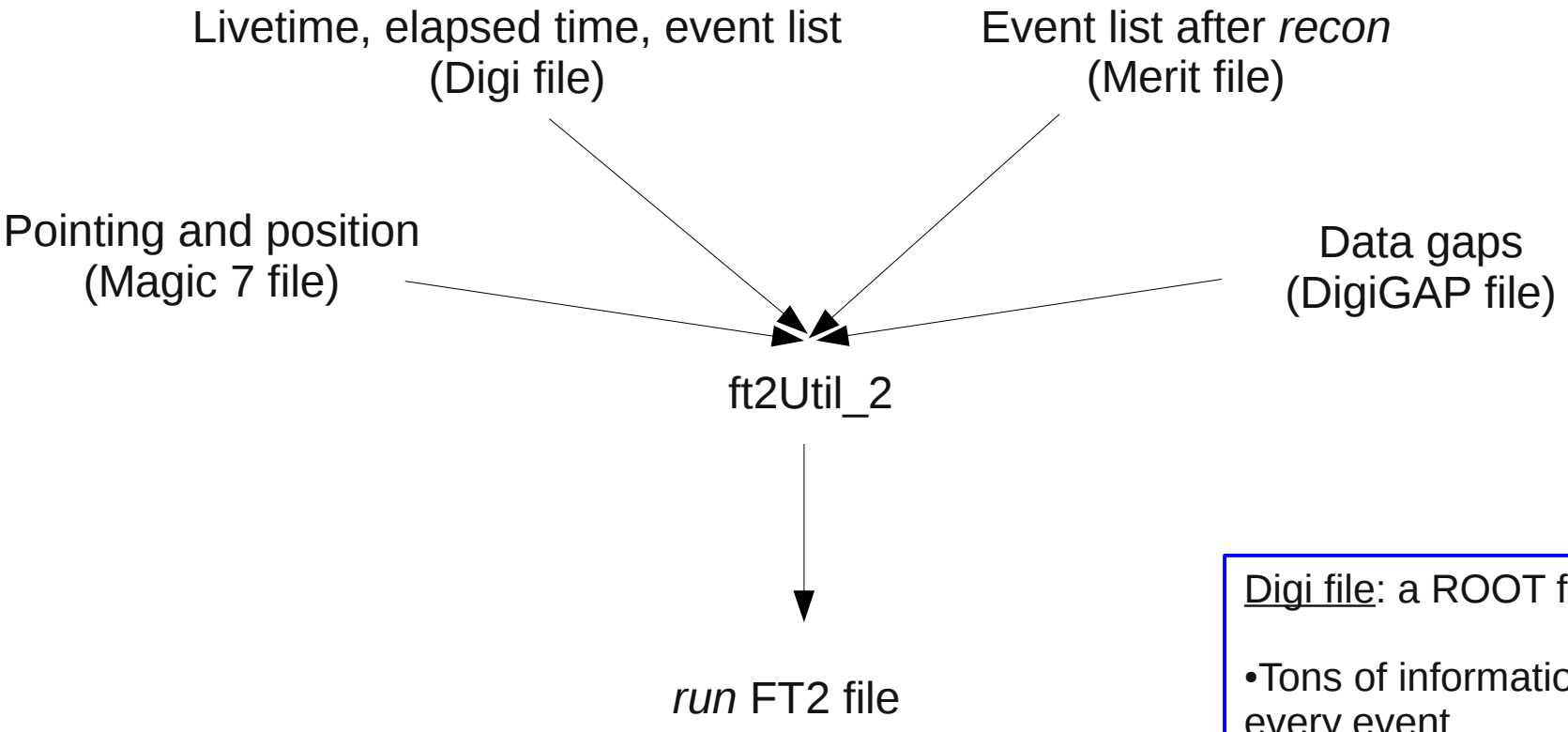
•Inside the constructor Ft2::Ft2(...):

- Parse the Magic 7 file (Magic7::Magic7(..)), determine the first time when ATT and ORB are concurrent and build a vector of times (in MET) starting there, with 1s increment, until the end of the file (Magic7::setupTimeIntervals())
- Starting from the vector of times reflecting the Magic 7 entries (magic7.timeIntervals), set up the time intervals to be written in the FT2 file, i.e., cut or pad to include the start and stop time provided by the user and nothing else
- For each time instant save (in std::maps):
  - a Status class: it provides 3 members to get all the quantities related to the spacecraft (ExtendedPointingInfo spacecraft) or the LAT (LatInfo lat), or the direction of the sun (astro::SkyDir sun)
  - a inSAA flag (true/false) based on M7 content
  - a livetime=0 (useless for *fake* FT2)

•Generate a FITS file with all the information obtained from the 3 maps filled above, organized in columns (ft2::writeFT2file())



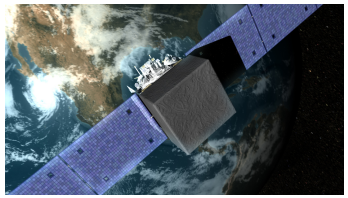
Goal 2: *run*[2] FT2 file



Digi file: a ROOT file:

- Tons of information about every event
- We are interested in the *livetime* and *elapsed* counters, as well as the event IDs

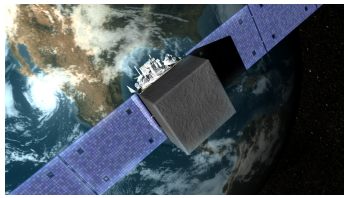
[2]The *run* FT2 file contains the pointing, position and livetime information for a whole run. It is called by L1 at the *run* level.



Livetime

- The livetime between two events e_1 and e_2 is the difference[3] in the value of *livetime* counter times the conversion factor between ticks and seconds
- The elapsed time is the difference[3] in the *elapsed* counter times the conversion factor
- The conversion factor can vary slightly due to clock drifts (as effect of temperature, for example)
- These effects are corrected for by comparing the number of ticks in one second with the elapsed time taken from the GPS. It is exactly the same procedure used to assign a time stamp to the events.

[3] In both cases the *value* of the counter has no meaning, only the *difference* is meaningful



Livetime

- Normally, the livetime between t_1 and t_2 is:

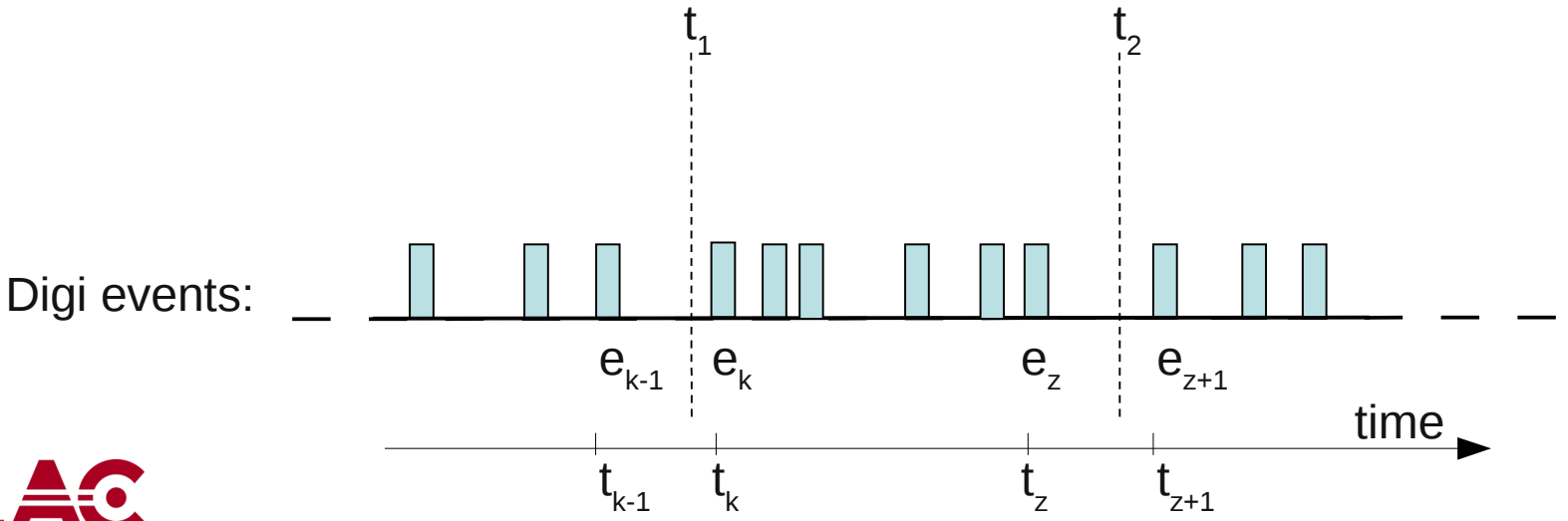
Livetime between the first and last event in the interval t_1-t_2

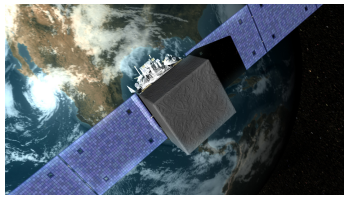
Livetime between the last event and t_2

$$\left(\sum_{i=k}^{z-1}(L_{i+1}-L_i)c_{i+1}\right)+\frac{t_k-t_1}{t_k-t_{k-1}}\times(L_k-L_{k-1})+\frac{t_2-t_z}{t_{z+1}-t_z}\times(L_{z+1}-L_z)$$

Livetime between t_1 and the first event

where L_i, t_i,c_i are respectively the livetime counter value, the time of the i-th event, and the conversion factor; e_k is the first event after t_1 and e_z is the last event before t_2.

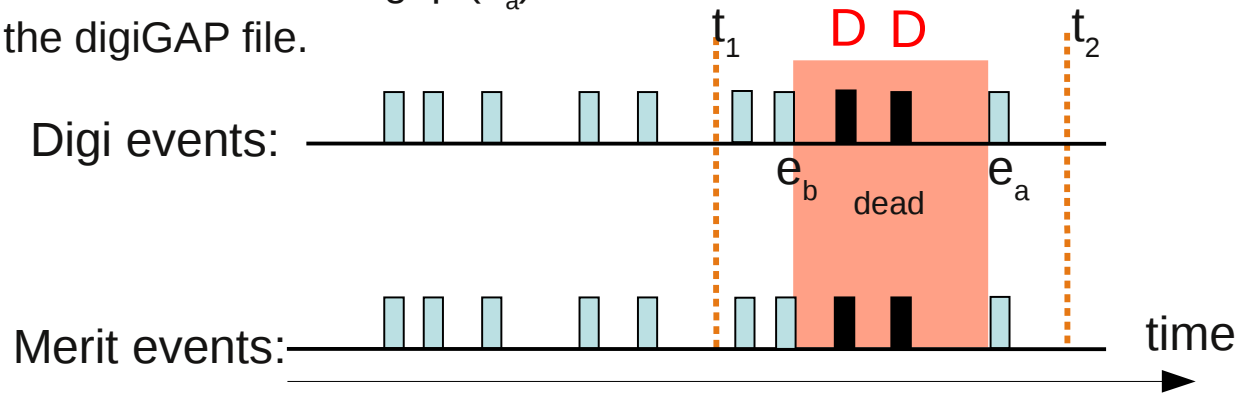




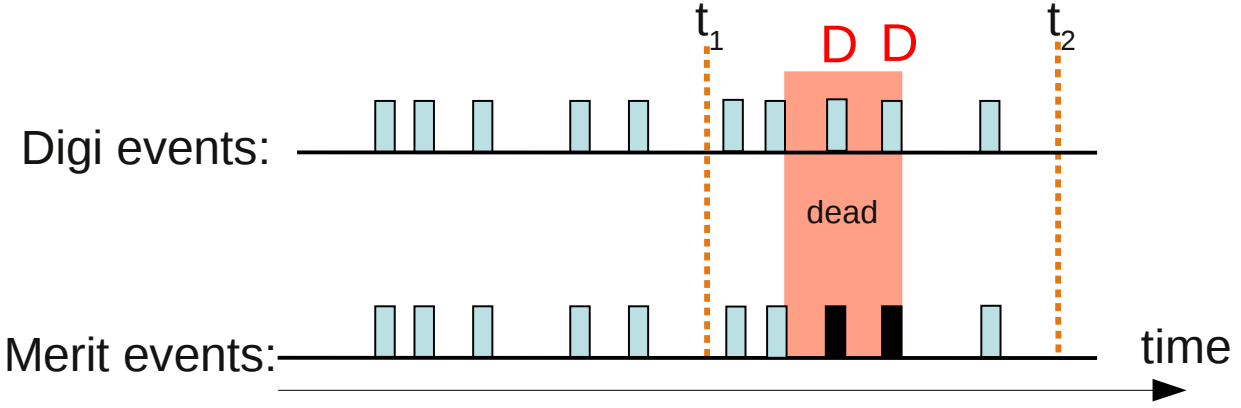
Livetime

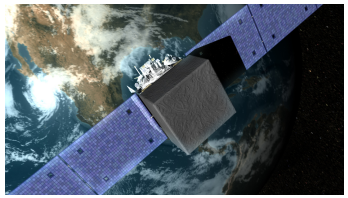
There can be 2 other sources of dead time, to be subtracted from the result of the previous formula:

- **Gaps in data**: data downloaded from the SC are incomplete. Events falling in the gaps are lost. They will lack in both the Digi both the Merit file. The time between the last event before (e_b) and the first event after the gap (e_a) is 100% dead time. The event ID of e_b and e_a are reported in the digiGAP file.



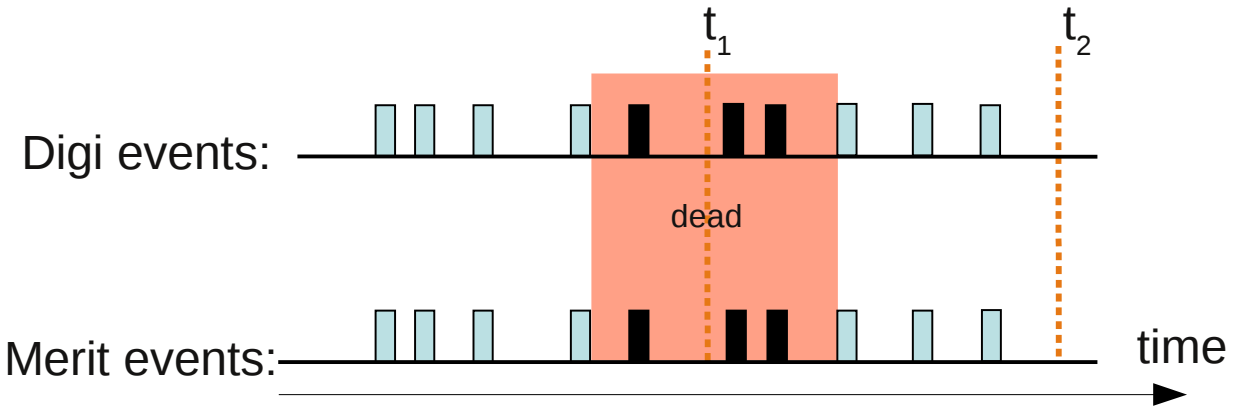
- **Recon crahes**: some events contained in the Digi file are not present in the Merit file, because for some reason they couldn't be reconstructed. The time between a “dead” event and the previous one is considered 100% dead time:

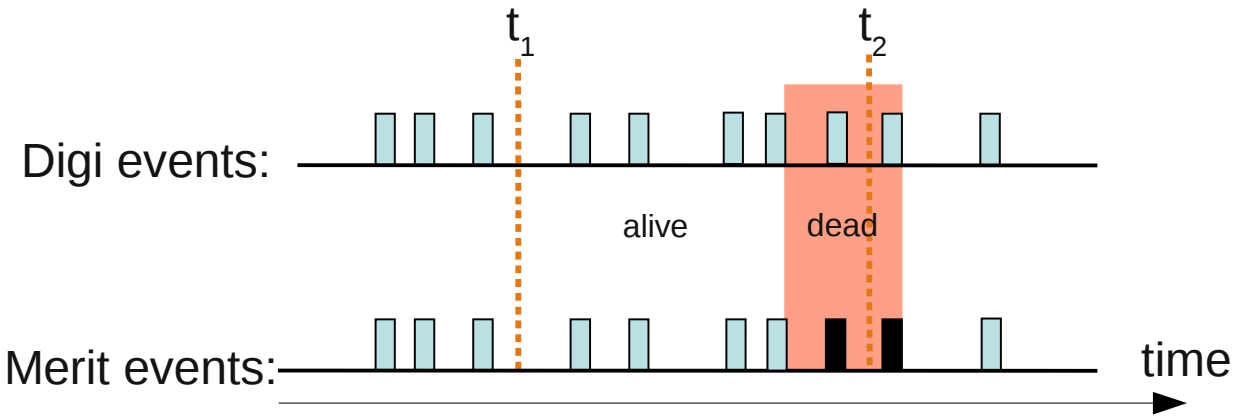




Livetime

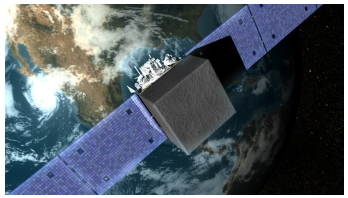
There are different possible combinations between t_1 and t_2 and the position of gaps:





•Compute the fraction of deadtime pertaining to the interval t_1-t_2, and subtract only that from the livetime

etc...



Goal 2: *run* FT2 file

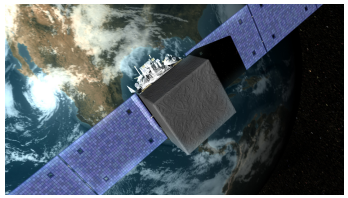
Steps:

•Parse the command line, get the configuration parameters, instantiate the Ft2 class (makeFT2.cpp)

•Inside the constructor Ft2::Ft2(...):

- Parse the Magic 7 file (Magic7::Magic7(..))
- Determine the time instants when ATT and ORB are concurrent (Magic7::setupTimeIntervals())
  (or they should be, but they aren't because of gaps in the M7 file)
- Cut the time instants vector at the start and stop time provided by the user
- For each time instant save (in std::maps):
  - a Status class: it provides 3 members to get all the quantities related to the spacecraft (ExtendedPointingInfo spacecraft) or the LAT (LatInfo lat), or the direction of the sun (astro::SkyDir sun)
  - a inSAA flag (true/false) based on M7 content
  - The livetime between this time instant and the following one (Livetime::operator())

•Generate a FITS file with all the information obtained from the 3 maps filled above, organized in columns (ft2::writeFT2file())



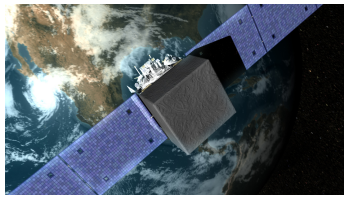
Goal 3: *30s* FT2 file

Steps:

•Parse the command line, get the configuration parameters (mergeFT2.cpp)

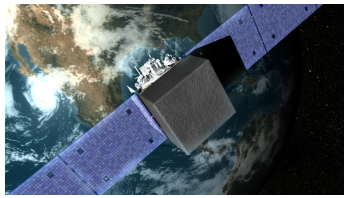
•“Rebin” the input 1s FT2 file to a new bin size (usually 30s). The “bin size” has to be integer:

- Output bins are long dt, BUT the algorithm stops the merging process before 30 s and begin a new bin if one of the following circumstances happen:
  - LAT_CONFIG changes
  - LAT_MODE changes
  - DATA_QUAL changes
  - LIVETIME becomes zero (saturation or digi crash)



Sweep events

- Sweep events are solicited triggers
- They always come in pairs (two events with the same event ID one after the other)
- There is always a sweep events at the beginning of a run, and there is usually a sweep events at the end of a run



Sweep events

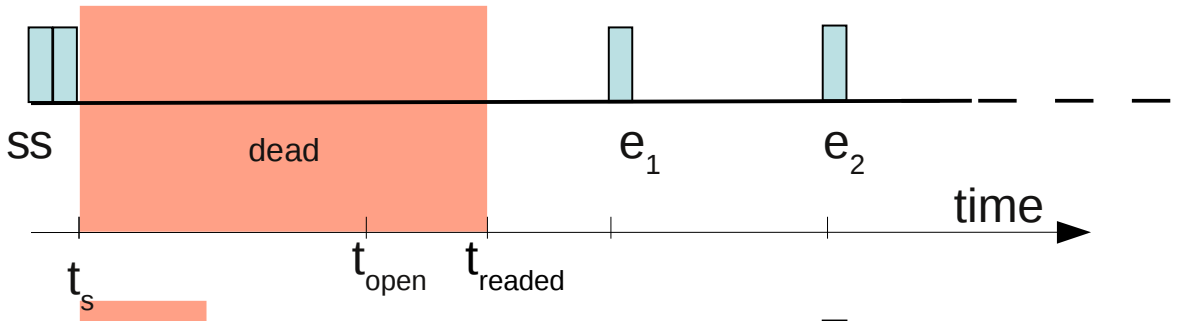
At the beginning of a run, there are 4 things happening:

A) Sweep event issued (solicited trigger) at t_s
B) Window open for physics at t_{open}
C) End of the read out of the sweep event at t_{readed}
D) First physics event of the run at t_1

- t_{open} is independent from the actual reading of the sweep event.
- If $t_{readed} > t_{open}$, the livetime is just $(L_1 - L_s)*c$ (as usual)
- If $t_{readed} < t_{open}$, there is an additional deadtime to take into account, which is $(t_{open}-t_{readed})$
- t_{open} = 10 x the number of ticks requested to complete a command (currently 10 x 196 = 1960 ticks).

Thus, to understand if C is before B, we can take the deadtime between the sweep event and the first physics event: if this deadtime is > 1960 ticks, then $t_{readed} > t_{open}$, otherwise $t_{readed} < t_{open}$.

Digi events:



Digi events: