

Production of low-level TOI's for Archeops

Éric Aubourg, September 1999

Introduction and rationale

Raw archeops data consist of telemetry blocs and flight recorder data. Both data are organized in the same way: blocks of different kind containing information for bolometers, stellar sensor, gyros, GPS or other devices, each block containing information for 72 transputer samples. Those two sets of data should contain the same information, but some blocks can be corrupted or missing.

An intermediate step in the analysis of the data is the production of Time-Ordered Information (TOI).

Since Archeops will fly several times, it is important to design a tool that can produce any set of low-level TOI's directly from the raw data, without any prior transformation of this data. This low level TOI's can then be fed into an analysis pipeline, whose first step can be the production of higher-level TOI's.

archtoi has been designed for such a purpose. It has been developed at CEA, in close cooperation with LAL, PCC-CDF, IAS and Grenoble.

Principles of operation

archtoi can be used in two different ways, either as a standalone program, or as a set of C++ classes. This note will first focus on the first mode of operation.

The first step is the creation of a request file, that contains the list of requested TOI, as well as various options, like the time range, time calibration constants, file path where to find the raw data, output format options, geometry information, etc.

archtoi can then be run with two command lines arguments, the name of this request file, and the name of the output file. The output file will contain as a header the same information as the request file, followed by a line by sample, containing the requested TOI's in successive columns, either in plain ASCII or in FITS bintable format.

Definitions

All times are given in MJD (Modified Julian Day), defined as JD-2450000. MJD=0 is thus 9 october 1995 noon UTC.

The request options

All request options should be prefixed with a '#'. The available options are

#ASCII	The output file will be plain ascii, one line per sample, one column per TOI and TOI flag, separated by tab characters.
#FITS	The output file will be in FITS format. The first extension of the file is a bintable, with one column per TOI and TOI flag. The names of the columns are the names of the TOI, with an optional index appended, e.g. BOLOTENS_6 for boloTens 6.
#TRANGE tmin tmax	Only TOI's with MJD between tmin and tmax will be output
#PATH directory	Raw data will be looked for in this directory. Files will be processed in alphabetical order, which is ok for TM and OBR naming conventions.
#FILE file	This raw data file will be processed. This option can be repeated. The files will be processed in alphabetical order.

#UNDEF string	Undefined TOI's will be output as "string" in the ascii output file. In FITS tables, the "NULL" FITS convention is used, usually read back as an IEEE NaN.
#RECORDER	The data is on-board recorder data, and not telemetry files
#MJD0 t0	Origin of samplenum-MJD calibration (see below).
#PERECH t	Sampling period, in seconds, for samplenum-MJD calibration (see below).
#ASIGPS file	Use the ASI GPS for balloon position. Raw ascii data is in file. For checking purposes, the GPS data used is dumped to a FITS file, <code>GPS-Dump.fits</code> . In that case, balloon position is always interpolated, and cannot be triggering.
#END	End of the requests, should appear after all options and all TOI's.

The TOI's

The following TOI's are defined:

sampleNum	Transputer sample number, defined as $\text{block_number} * \text{nb_samples_per_block} + \text{index_in_block}$.
internalTime	Transputer time in seconds, since last reboot. Computed from sample number and sampling period.
mjd	MJD of sample, computed from sample number, sampling period and time origin (MJD0)
boloTens i	Signal of bolometer i, in μV , filtered for differential measurement.
boloRaw i	Raw signal of bolometer i, in μV , without any filter.
boloTemp i	Temperature of bolometer i, in K. N/A ¹ .
sstChannel i	Raw signal of SST channel i. Range for i is 0-47.
sstDiode i	Raw signal of SST diode i. Range for i is 0-45. Diodes are reordered, diode 0 has lowest elevation.
sstStarCnt	Number of stars detected on SST in that sample.
sstStarZ i	Stars detected by SST. # of diode of star i. $i : 0 \rightarrow \text{sstStarCnt}-1$.
sstStarF i	Stars detected by SST. Flux of star, in picoAmps input current.
sstStarT i	Stars detected by SST. Time of peak, in mjd.
gyroRaw i	Raw gyro value. Range for i is 0-2.
gpsTime	UTC time, in integer seconds, given by the GPS
longitude	Balloon position. Can use ASI GPS data (cf #ASIGPS option)
latitude	Balloon position. Can use ASI GPS data (cf #ASIGPS option)
altitude	Balloon position. Can use ASI GPS data (cf #ASIGPS option)
tsid	Local sidereal time, in seconds.
azimut	SST azimut, as reconstructed from SST data, in degrees. N/A
alphaAxis	RA of gondola rotation axis. N/A.
deltaAxis	DEC of gondola rotation axis. N/A.
alphaSst	RA of center of SST. N/A.
deltaSst	DEC of center of SST. N/A.
alphaBolo i	RA of center of bolometer i. N/A.
deltaBolo i	DEC of center of bolometer i. N/A.

A TOI request should be prefixed by '@'. It can be followed by an integer index if the TOI is an indexed one. It can be followed by the keywords

1. N/A signals that this TOI is not yet available

- **notrig.** This TOI is not triggering. A triggering TOI will produce the output of a new line of data when its value changes. For instance, if you ask for @longitude, and @boloTens 2 notrig, you will get a line of data for each GPS new longitude information, alongside with the bolo signal at that time, although many more bolo samples are available inbetween.
- **repet.** When no new value is available, but another TOI triggered the output of a line of data, the previous data for this TOI will be repeated.
- **interp.** When no new value is available, but another TOI triggered the output of a line of data, the data for this TOI will be interpolated linearly with the previous and next sample.
- **flag.** Two columns will be output for this TOI, first a flag column, with “1” if a new value was obtained, and “0” if not, and then the value of the TOI, computed accordingly to “repet” or “interp” keywords.

An exemple of request file could then be

```
@sampleNum
@utc
@gpsTime
@longitude interp
@latitude interp
@altitude interp
#RECORDER
#PATH Sans titre
//#PATH Stock:Archeops:Trapani99
#MJD0 1376.8358819
#PERECH 0.005836818076
#ASIGPS Stock:Archeops:ASI_GPS_archeops1999.ascii
#UNDEF #
#END
```

Note about time calibration

GPS data contains UTC only with an accuracy of one second. What happens is that the GPS sends an ascii string containing the time (some models start sending the data on a second boundary, some others send it randomly within the one second interval). The transputer then receives the string after some delay due to the serial line speed. This GPS data will then be sent in the next GPS block.

It is possible to do a linear fit of UTC versus sample number. There are two caveats :

- For the Trapani 99 flight, the relation is not simple. It is linear before and after 19:00 UTC, with a step of more than 7 seconds inbetween. One might be concerned that, since the GPS had trouble seeing satellites during the flight, it might have miscalibrated its internal clock. Fortunately, we have Spain ground data, with more than 5 satellites in sight, which is continuous with the flight data. The flight data is then probably good, and the step is due to a step in transputer sample number (change of sampling frequency, or stop/start of transputer) rather than a step in GPS UTC.
- The residuals of the fit display a flat distribution between -0.5 and 0.5 seconds. This indicates that the GPS is sending its data anytime during the second interval it is valid. One should then add to the fit relation a 0.5 second constant, plus an estimate of the delay to get the string. Neglecting the latter transforms equation 1 of Gispert's archeops note #1 into

$$UT = 8.0611317 + 0.00011673636 * \text{block_number}$$

and taking into account the delay to transmit 57 characters at 4800 bauds yields

$$UT = 8.0611646 + 0.00011673636 * \text{block_number}$$

This formula should be accurate to better than 10^{-6} in relative values, and to about 0.1 second in absolute values.

The corresponding values for MJD0 and PERECH are then

```
#MJD0 1376.8358819  
#PERECH 0.005836818076
```

Software evolution

Processing of SST data, as done by D. Yvon and F. Couchot, will be integrated to yield accurate positioning of the SST center at any time. Overall geometry should be integrated to yield the position of the center of lobe of each bolometers.

The boundary between `archtoi`, i.e. “low-level” TOI’s, and the rest of the analysis pipeline should be defined. The set of TOI’s defined in the current version of this note yields a natural boundary, i.e. higher level TOI’s should be created in the rest of the pipeline. Note that this does not exclude iterations between the rest of the pipeline and `archtoi`, which is a better solution than producing a cascades of refined TOI’s. For instance, comparison of SST and bolometer data can help refine the geometry, which can be fed into `archtoi` through options, in order to produce a better set of TOI’s.

Using directly the C++ classes

A way to use this software is to drop the `archtoi` main program, and use directly the `toisvr` class, which returns the TOI’s one after the other. Data processing can be done directly in C or C++. For people who want to use Java as their data analysis language, a JNI or CORBA interface could be written, performance issues having to be dealt with for the CORBA interface.