SPIRE Data Frame Specification

Technical Note

0.2

6 Sep 2004

Pasquale Panuzzo
INAF – Osservatorio Astronomico di Padova
panuzzo@pd.astro.it

1 This Document

This note provides an overview on SPIRE Data Frames, describing their scope and the implementation. In this note it is also discussed the role of the SPIRE Telemetry packet processor. The storage and retrieval of data frames are described in another document ([4]).

2 Purpose

SPIRE Data Frames were developed to easily handle the scientific data coming from the telemetry (TM) of SPIRE.

During SPIRE operations, instrument data are downloaded from the satellite and stored into the ICC database. Instrument data are sent in the form of telemetry packets.

The data contained in each packet are stored in the database as a bit array. In order to look at the values of telemetry data, the packet must be extracted from the database and the data must be reconstructed from the bit array. Using this design, it is not possible to query the database for packets in which a given telemetry field has a selected value. The principal reason of developing SPIRE data frames is to solve the above limitation.

Moreover, the current implementation of SPIRE data frames is aimed to provide a structure for TM data that is easily accessible, and to have a easy conversion of data frames into SPIRE Data Products (see [7] for SPIRE data product definition).

3 SPIRE data frame definition

The On-Board Software (OBS) produces a number of different TM packet types. Information on the SPIRE TM packet structure can be found in [5]. Some TM packet types are related to software/transmission errors or reports. These TM packets will be not translated into data frames because they are not directly related with scientific data. TM packet types related to detectors, BSM, SCU, SMEC, calibration sources and housekeeping have corresponding data frame types, one for each TM packet type. For each data frame type a java interface in the `herschel.ccm.api.spire` package and a java class (that implements the corresponding interface) in the `herschel.versant.ccm.spire` package was written.

Each data frame contains one member for each quantity contained in the parent TM packet. As an example, a science telemetry packet created by the SPIRE photometer could contain the voltage readouts of all 288 detectors; this packet is converted into data frames in which each voltage readout is stored into a specific member of data frame java class. The detector arrays are not combined into single arrays, rather they are treated as independent parameters. This is consistent with the way they are defined in the telemetry.

Packet name	description	multi-frame
CHK	Critical HK Report	No
NHK	Nominal HK Report	No
PEAKUP	Peak Up Report	No
PHOTF	Photometer Full Array Frame	Yes
SPECF	Spectrometer Full Array Frame	Yes
SMECSCAN	SMEC Continuous Scan Frame	Yes
BSMNOMINAL	BSM Block	Yes
SCUNOMINAL	SCU Block	Yes
PHOTSW	Photometer SW Array Frame	Yes
PHOTMW	Photometer MW Array Frame	Yes
PHOTLW	Photometer LW Array Frame	Yes
SPECSW	Spectrometer SW Array Frame	Yes
SPECLW	Spectrometer LW Array Frame	Yes
SMECSELECT	SMEC Selected Data	Yes
PHOTOFF	Photometer Offsets	Yes
SPECOFF	Spectrometer Offsets	Yes
TFCSHK	TFCS HK Report	No
TFTSSCIENCE	Test FTS Science Report	Yes
TFTSHK	Test FTS HK Report	No

Table 1: TM packets that generate SPIRE data frames.

The detectors are also physically independent in the instrument, each having its own associated time readout and characteristics.

In order to reduce the number of TM packets, each packet can contain several science frames produced by the DRCU at different times. In this case, each data frame will correspond to a single science frame from the DRCU.

The Java classes are automatically generated from the information in the parameter and packet type tables, which are also used by the QLA. The names of java interfaces and classes are given by `SpireDataFrame` plus the packet name; e. g. the PHOTSW packet corresponds to `SpireDataFramePhotsw.java` interface and `SpireDataFramePhotswImpl.java` class. The list of TM packets with corresponding SPIRE data frames can be found in table 1. Packets that can have several frames inside are identified in the same table.

It is worth of noting that the above implementation of data frames is specific of the SPIRE instrument. HIFI and PACS instruments have different definitions of what a data frame is.

4 SPIRE Telemetry packet processor

The conversion of the TM packets in data frames is done during the ingestion into the ICC database. Information about the TM ingestion can be found in [1], [2], [3]. The extraction of data frames from TM packets is done by the SPIRE Telemetry packet processor. After the extraction, the data frames are stored in the ICC database.

During the extraction of TM data, the SPIRE Telemetry packet processor has also to translate the data format and to compute the appropriate time reference of the data frames, as described below.

It is also possible to create or regenerate the data frames after the TM packets have been ingested. This is important in order to recover possible errors during the ingestion.

The SPIRE TM packet processor should also control the validity of data frames rejecting those with an invalid checksum or FrameID. The current implementation of the processor does not make this control.

It is to be determined if the performances of the current implementation of the SPIRE TM packet processor are sufficient to create data frames during a real-time ingestion. If the SPIRE TM packet processor is not efficient enough, there could be a lost of data during the ingestion of data frames into the ICC database. Tests on the processor efficiency should be carried out.

5 Memory footprint of TM and data frame values

Data in TM packets are stored as unsigned values, represented with the appropriate number of bits. However, the java language only supports signed variables. For this reason the TM data have to be stored in data frames using signed variables with a larger number of bits. As an example, a 32-bits unsigned value has to be stored in a 64-bits signed (`long`) variable. The above solution poses a big problem because the storage space could be doubled.

In order to reduce the needed storage space, TM data representing detector readouts are stored in signed values with the same memory footprint. In TM packets a detector readout is stored in a 16-bits unsigned value; in data frames this value is stored in a 16-bits signed `short` variable. If the first bit of the 16-bits unsigned value is 1, the number is stored as a negative `short`. In order to restore the original value of the above detector readout, it was written an utility that provides the translation. This translation is transparent to the data frame user, they do not need to care about it.

6 The Time in SPIRE data frames

As said above, each SPIRE data frame correspond to a single telemetry frame. Each frame is generated at a given time. We define this time as the data frame time. The general HCSS definition of data frames have two different times associated to the dataframe: a "start" and an "end observation time". In SPIRE data frames these two times coincide. The time origin is fixed at 00:00:00 of 1st January 1958, and the time unit is the microsecond. Accordingly to this definition, the data frame time is stored as a `FineTime` variable.

For TM packets that can have only one frame (e.g. housekeeping TM packets), the data frame time is equal to the time of the packet creation by the DPU.

The time of the packet creation by the DPU is given in TM packets in CCSDS Unsegmented Time Code (CUC) format (see [8] and [9]), i.e. the time elapsed from 1st January 1958 in units of 1/65536 seconds. In data frames, this time is converted in a `FineTime`. However, the CUC value of time is also stored as a normal variable.

For TM packets that can have more than one frame, the time at which the frame was created is stored as the number of ticks of the DPU clock from the last reset of the DRCU. The DPU clock ticks have a duration of 3.2 microseconds. The last reset time of the DRCU is stored in the TRESET quantity into the Nominal HouseKeeping TM packets with the CUC format. Thus, in order to compute the data frame time form 1st January 1958 epoch of science frames, it is necessary to keep in memory the TRESET value of the last NHK packet. Thus the data frame time is given by:

$$\texttt{dataFrameTime} = (\texttt{long})(\texttt{treset}/65536. + \texttt{frameTime} * 3.2)$$

During a building block, the TRESET value should be constant. However, it could happen that the DRCU is resetted during the data-taking operation. The data taken between the two NHK packets indicating the changed TRESET should be marked as invalid. This operation cannot be done during the ingestion into the database, and should be done in the Engineering Data Process step of the pipeline (see [6]).

Finally, it should be noted that the readout time of each detector is different from the data frame time and the readout time of other detectors. Thus, the readout time and the data frame time should be not confused. The computation of the readout times will be carried out by the Engineering Data Process step of the pipeline.

7 The data frame factory

TBW

References

[1] Kevin Galloway, HCSS telemetry ingestion technical note, FSCDT/TN-015

[2] Kevin Galloway, HCSS telemetry ingestion software user manual, HERSCHEL-HSC-DOC-0231

[3] Kevin Galloway, HCSS telemetry packet processor technical note, HSCDT/TN-021

[4] Steve Guest, Telemetry and Data Frame Interface Control Document, SPIRE-RAL-DOC-000788

[5] Ken J. King, SPIRE Data Interface Control Document, SPIRE-RAL-PRJ-001078

[6] Ken J. King, Engineering Data Process Step Specification technical note

[7] Ken J. King, SPIRE Data Product Definition Document

[8] S. Thürey, Herschel/Planck Packet Structure - Interface Control Document, SCI-PT-ICD-7527

[9] CCSDS 301.0-B-3 Blue Book