PacketLib 3.4.0 Interface Control Document

R.I. CIWS-IASFBO-TN-011

Custodian:	Name:	Andrea Bulgarelli	Signature:		Date:	
Prepared by:	Name:	Andrea Bulgarelli	Signature:		Date:	
Reviewed by:	Name:		Signature:		Date:	
Approved by:	Name:		Signature:		Date:	

AUTHOR LIST

Andrea Bulgarelli	INAF/IASF Bologna, Italy

DISTRIBUTION LIST

CIWS e-mail list	ciws@iasfbo.inaf.it

DOCUMENT HISTORY

Version	**Date**	**Modification**
d0.1	31 March 2014	First draft

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Scope and Purpose of the Document

The PacketLib C++ library has been developed in order to facilitate the development of applications which handle data streams containing Source Packets compliant to the ESA Telemetry and Telecommand standard as defined by AD[1] and AD[2], respectively.
The library has been developed in the framework of the DISCoS system (RD[1], RD[2]) adopted for the development of some of the ground software for the AGILE mission (RD[3], RD[4], RD[5], RD[6], RD[7]).
The library is devoted to applications which are required either to generate or to process data stream containing ESA standard Source Packets.
In general, the library is used for applications which require to decode/encode the various data fields of the packets which are read/written in raw format from/to an input/output stream.
The various cases are dealt by the I/O Abstraction Layer of the library, which allows the application to easily select the I/O device among those supported by the library (file, socket, shared memory).
The Telemetry Management Layer of the PacketLib library implements the data stream structure by means of configuration files, allowing the application to handle different layouts without using any hard-coded information.
The purpose of the present document is to identify and describe the various interfaces of the PacketLib.
Chapter 3 introduces the PacketLib design concept, which is based on the configuration files presented in Chapter 4. Chapter 5 identifies the various type of Source Packet layouts which can be easily modelled by the PacketLib configuration files.
Further documentation on the PacketLib is provided by AD[3].

Definitions, Acronyms, Abbreviations

The following is a list of definitions used throughout this document.

1.1.1. Acronyms

AC	Anti-coincidence auxiliary subsystem
ASCOS	AGILE Science Console System
Calibration MGSE	Calibration Mechanical Ground Support Equipment
CCOE	Central Checkout Equipment
EGSE	Electrical Ground Support Equipment
GSE	Ground Support Equipment
Instrument SC	Instrument Science Console
IP	Integrated Payload
MCAL	Mini-calorimeter detector

PD	Photo Diode
P/L	Payload
PDHU	Payload Data handling Unit
SA	X-Ray detector named Super-AGILE
S/C	Space Craft
ST	Silicon Tracker gamma-ray detector
TBC	To Be Confirmed
TBD	To Be Defined
TC	Telecommand
TE	Test Equipment
TM	Telemetry

2. APPLICABLE AND REFERENCE DOCUMENTS

1.1. Applicable Documents

AD[1] Packet Telecommand Standard, ESA-PSS-04-107, Issue 2
AD[2] Packet Telemetry Standard, ESA-PSS-04-106, Issue 1

1.2. Reference Documents

The following is a list of documents that are referenced within the text of this document:
RD [1] Gianotti F., Trifoglio M., DISCoS – a detector-independent software for the on-ground testing and calibration of scientific payloads using the ESA Packet Telemetry and Telecommands Standards, Istituto TeSRE/CNR -Via P. Gobetti 101, 40129 Bologna, Italy Astronomical Data Analysis Software and Systems X, Boston 12-15 November 2000, ASP Conference Series, Vol. 238, 2001
RD [2] A. Bulgarelli, M. Gianotti, F. Trifoglio, *PacketLib: a C++ library for scientific satellite telemetry applications*, Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003 H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds., p.473.

3. Packetlib overview

1.1. Architecture

As sketched in Figure 3.1, the PacketLib is structured into two main layers: the Telemetry Management layer, which interfaces the application, and the I/O Abstraction layer, which interfaces the Operating System.
The former allows the application to address the various elements forming the packet data stream, without having to deal with the specific structure. The latter abstracts from the specific input and output mechanisms (e.g. sockets, files, and shared memories).
This approach allows a more rapid development of the user applications without having to deal with the kind of input and output sources, that could be changed at run time. Indeed the library could be used either to process or to generate a telemetry stream.

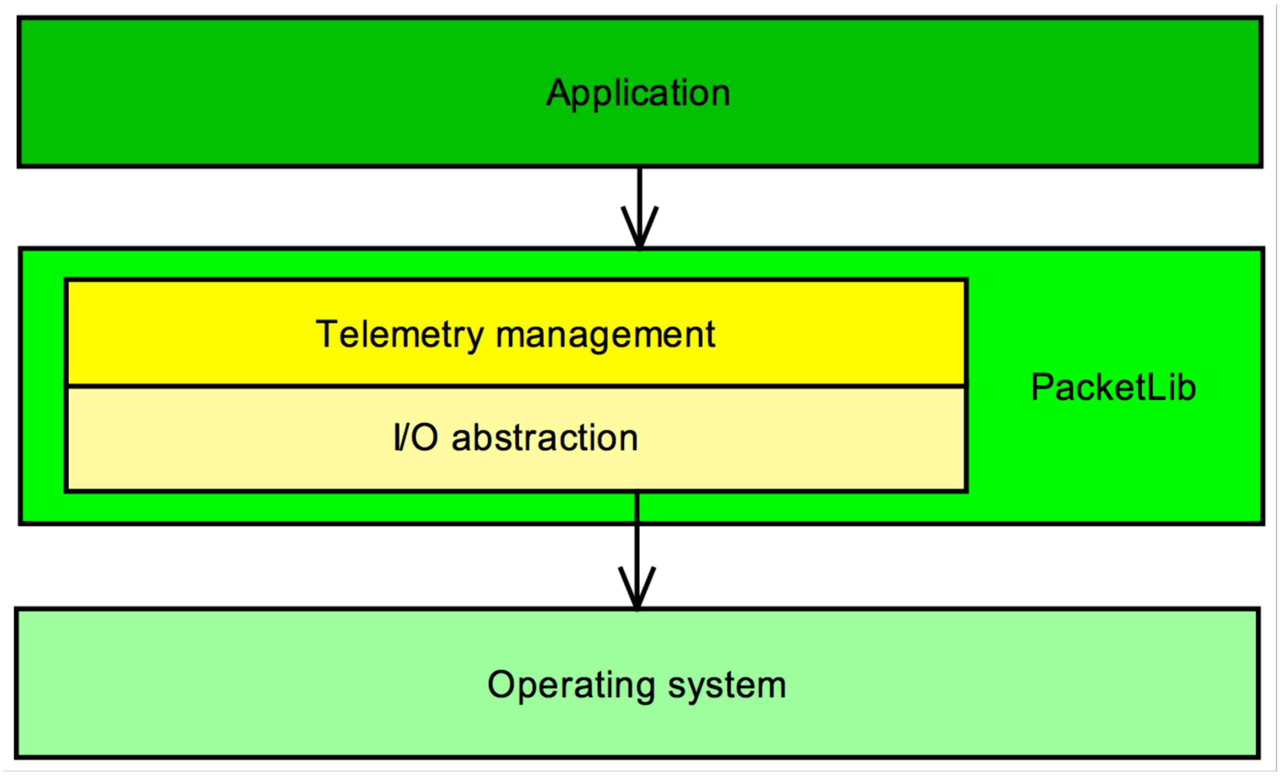

Figure 3.1: PacketLib layers

1.2. How does the library work

The real working of the library can be easily understood with a little coding example that processes the input as detailed herafter.

```
1. InputPacketStream ps;
```

```
2. char* parameters; //set the parameters of input

3.Input *in = new InputSocket(parameters);

4. in.open();

5.ps.createStreamStructure("conf.stream");

6. ps.setInput(in);

7.Packet* p = ps.getPacket();

8. cout << "The packet reads from input is" << p->name << endl;

9.cout << "The value of field 4 of header is " << p->header-
>getFieldValue(4) << endl;

10. cout << "The value of field 2 of source data field is " << p-
>dataField->sourceDataField->getFieldValue(2) << endl;
```

It is noted that, the use of configuration files containing all the parameters which specify the characteristic of the data stream (prefix, endianity, …) and the structure of the Source Packets (header length, fields, …) allows the application to handle the data stream without using any hard-coded information.
Line 1 instantiates the object that represents the input byte stream, and line 5 loads the configuration files and creates into memory the byte stream structure and the packets structure (header, data field and fields). Lines 2, 3, and 4 create an input source and open it. To change the input source without modifying the rest of the code, it is only necessary to modify the line 3 by instantiating another type of input (e.g. InputFile instead of InputSocket), and opening it with the correct parameters.
The line 6 links the input byte stream with the input source, and the remaining code extracts the required packet information from the input stream. In particular, line 7 reads a complete packet, whereas lines 8-10 print the value of the various fields.
Summarizing: lines 2-4,6 provide the I/O Abstraction layer, while lines 1,5,7-10 are related to the Telemetry Management layer.
From this example it is evident that with a few lines of code it is possible to connect the application with an input source and obtain an object representing a packet with all the field values decoded starting from a byte flow. The library is able to manage either big-endian and little-endian formats.
For a full comprehension of how the library works, it is necessary to understand how the input is processed. After having read the packet Header (which is of fixed length), the library knows the length of the Data Field which follows. The actual structure of the Data Filed is derived by its identifier which consists of one or more fields containing some predefined values. Typically, the APID field of the TM/TC standard is used, but the library has not limitation, and allows the use of identifiers which include others fields (e.g.: the Type/Subtype fields of the Data Field Header).

The identification of the various packets is performed by looking in the packets for one of the possible identifiers defined in the specific configuration file.
Identified packets (i.e. Packets which structure is compliant with those specified by the configuration file) are unpacked and the various fields are read into the packet structure defined in the configuration files, where can be directly addressed as shown in lines 9,10. In the negative case, an object representing a generic not recognized packet is created, where the Data Field byte sequence is copied without any structure.

4. Packetlib configuration files

1.1. The telemetry stream

The PacketLib is able to handle **binary data streams** containing a sequence of Source Packets, with additional prefix/trailer, if any.
The structure of a given data stream is provided to the PacketLib by the ***.stream** configuration file, which specifies:

- the presence of a fixed length prefix (if any), and its length
- the Endianity (little-endian or big-endian)
- the name of the PacketLib configuration files containing the definition of the Telemetry and Telecommand structure of the Source Packets (see 4.2).

Details on the *.stream configuration file format that describe the stream are given in Chapter 6.

1.2. Source Packet structure

According to AD[1] and AD[2], the Source Packets contained in the data stream is usually divided into two sections:
Packet Header: it has fixed length. It is mandatory and provides the fundamental information which describe the remaining part of the Source Packet, namely:

1. identification of the source and its application process (APID)
2. sequential numbering of the packet having the same APID (SSC)
3. packet data field length

Packet Data Field: it contains either the TM or the TC data.
In turn, the Packet Data Field has its own internal structure, which, in the more general case, consists of:

- **Data Field Header**: which contains information related to the Source Data Field.
- **Source Data Field:** which contains the data grouped according to a logical sequence of bits which represent well defined information.
- **Tail**, containing additional fields (for example, CRC value).

In addition a pair number of byte should be present in each telemetry packet before the header. This is not part of the standards, but it is often used in the context of the test equipments to add to the packet some particular information that depends by the current test. This group of bytes is called **Prefix**.
All the parameters, which define the fixed characteristics of the above structures, are provided to the PacketLib by the Source Packet configuration files, namely:

- the ***.header** file, containing the definition of the packet header format (see Paragraph 4.3), described in the Chapter Error: Reference source not found
- the ***.packet** files, containing the definition of the different packets which are expected in the data stream (see Paragraph Error: Reference source not found), described in the Chapter Error: Reference source not found.

The *.packet file provides to the PacketLib the parameters which describes the Packet Data Field structure.
The PacketLib requires one (or more) file for each different Source Packet structure.

A given Source Packet structure is univocally identified by a given set of fields contained in predefined position of the Source Packet and having specific predefined value. i.e.:

- the APID field
- the Type field
- the Subtype field.

These fields are listed in a section of the .*packet file. The identifiers should be freely chosen among the fields of the header (value 0), of the data field header section (value 1) or of the fixed part of the source data field (value 2).
The **.packet* configuration file format foresees various layouts of the packet data field. Details on the **.packet* file format are given in the Paragraph 6.5,

1.1.1. Basic structure of a PacketLib TM/TC packet

The supported telemetry layouts are specified in terms of *blocks* and *elements*, which are defined by the PacketLib as follows:

- **Block**: this is the general term which identifies a logical unit of information which is contained in the Source Data Field. E.g.: the burst event data block reported in the example of Paragraph 4.6 is a Block (of variable dimension). Another example is the grid event data block reported in the example of Paragraph 4.4 (with a block of fixed dimension).
- **Element**: this is the logical unit of information which is repeated within the Block of variable dimensions. E.g.: the 2x16-bit words information provided for each CsI Bar in the block reported in the example of Paragraph 4.6 is an Element.

PacketLib can manage a *recursive* structure of blocks (blocks defined within other blocks) with different layout and variable dimensions. This is the most general telemetry layout and it is described in the Paragraph 5.4. An example is reported in the Paragraph 4.7.

1.3. The Packet Header

This file specifies the structure of the Source Packet Header, which is unique for all the Source Packets, providing to the PacketLib the description of the fields contained in the Header in terms of:

- Text Description
- Length (in bits)
- Predefined value (if any)
- the index of the field of the header that contains the dimension (in byte) of the packet.

Details on the *.header configuration file formats are given in see Paragraph 6.4.

1.1.1. Packet Header example

The following is an example of a packet header which is compliant with the specifications given in AD[1] and which is modelled by the .packet configuration file reported in Paragraph 6.2.

Packet Header (3x16-bit word)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
MSB															**LSB**
Version Number			Type	DHFH	APID										
SF		Source Sequence Counter													
Packet Length															

Where:

First word:

Version number:	must be set to '100' binary
Type:	must be set to '0' binary
DFHF:	Data Field Header Flag, must be set to '1' binary
APID:	Application Process Identifier

Second word:

SF:	Sequence Flag, must be set to '11' bin
Source Sequence Counter:	counts the packets of the above APID; to be reset at the begin of each Test Session (note: each APID has its own counter).

Third word:

Packet Length:	[number of octects in Packet Data Field] – 1;

The description of the *.header file that describe this structure is reported in the Paragraph 6.4.

1.4. Packet Data Field – example 1

The packet Data Field Header has the following format:

Data Field Header (1 x 16-bit word)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
MSB															**LSB**
Spare		Checksum type		ACK				Packet Type				Packet subtype			

Where:

Spare:	must be set to '00' binary.
Checksum type:	must be set to '00' binary
ACK:	must be set to '00' binary.
Packet type:	must be set to 0x5.
Packet subtype:	must be set to 0x5.

Hence: this word must be set to the fixed value: 0x0055

The Source Data Field has the following format:

Source Data Field (1 x 16-bit word record)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
MSB															**LSB**
TC Application Data															

Where:

TC Application Data:	**must be set to the fixed values:**
	– **0x0200,** for the "Start Observation" command;
	– **0x0000,** for the "Stop Observation" command.

1.5. Packet Data Field – example 2

In this example the Source Data Field contains variable number of events having fixed length.

As shown below, the Data Field Header contains the field "*Nevents*" which gives the actual number of events contained in Source Data Field.

Data Field Header - first 4 (16-bit) words

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
MSB															**LSB**
Spare						Checksum flag		Packet Type				Packet subtype			
Time Tag second (32 bits integer signed MSB)															
Time Tag second (32 bits integer signed LSB)															
Time Tag millisecond (16 bit unsigned)															
Spare												Nevents			

Where:

Spare:	must be set to '00' binary.
Checksum type:	must be set to '00' binary
Packet Type:	identification of the packet type
Packet Subtype:	identification of the packet subtype
Time Tag:	CPU System Time expressed as second and millisecond since 00:00 UTC January 1,1970 (i.e. as given by the C routine *ftime*) corresponding to the creation of the packet.
Nevents:	Number of events contained in the packet.

In this example, derived from the AGILE Minicalorimeter Test Equipment, each event has a fixed length, and consists of:

- a time tag (2x16-bit words)
- 30 CsI Bars information (2x16-bit words for each Bar)

The event structure is shown hereafter.

GRID event data block format (62 x 16 bit word)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
MSB															**LSB**
TE Internal Counter (MSB)															
TE Internal Counter (LSB)															
'0'	spare			Bar 01 PDA PHA (12 bits)											
'0'	spare			Bar 01 PDB PHA (12 bits)											
…	…			…											
…	…			…											
'0'	spare			Bar 30 PDA PHA (12 bits)											
'0'	spare			Bar 30 PDB PHA (12 bits)											

Where:
PDA PHA:Pulse Height Amplitude of PD A
PDB PHA:Pulse Height Amplitude of PD B

1.6. Packed Data Field – example 3

In this example the Source Data Field contains a variable number of events having variable length.

In addition to the specific field in the Data Field Header in case a), in this case the PacketLib requires that each event contains in a fixed position either its actual length, or a value from which the actual length can be derived.

In this example, derived from the AGILE Minicalorimeter Test Equipment, each event consists of:

- a time tag (2x16-bit words)
- a variable number of CsI Bars information (2x16-bit words for each Bar)

As shown below, the first field of the event gives the “Number of bars -1” parameter from which the total event length can be derived.

BURST event data block format (variable number 3+2*n of 16 bit word)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
MSB															**LSB**
'1'	spare										Number of bars -1				
TE Internal Counter (MSB)															
TE Internal Counter (LSB)															

'0'	'0'	1st Bar ID (2 MSB)	PDA PHA (12 bits)
'0'	1st Bar ID (3 LSB)		PDB PHA (12 bits)
…	…		…
…	…		…
'0'	'0'	n-th Bar ID (2 MSB)	PDA PHA (12 bits)
'0'	n-th Bar ID (3 LSB)		PDB PHA (12 bits)

Where:
Bar ID:Bar identification: range [1,30]
PDA PHA:Pulse Height Amplitude of PD A
PDB PHA:Pulse Height Amplitude of PD B
A PacketLib element in the above layout is the following:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
MSB															**LSB**
'0'	'0'	n-th Bar ID (2 MSB)		PDA PHA (12 bits)											
'0'	n-th Bar ID (3 LSB)			PDB PHA (12 bits)											

1.7. Packet Data Field – example 4

This is the example that uses blocks defined within other blocks. This packet is compound of block called Events. Each event is compound of 4 blocks of variable dimension.

Data Field Header

MSB

LSB

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0 2	*32 bits Synchronization Marker*															
4	*CRC Flag*		*Packet Type*						*Packet Subtype ID*							

6	*MM Page ID*		
8	*MM Packet Counter*		
10	*Century (0x20)*	*APID Sequence Counter*	
12	*On-board Time-tag (64 bits)*		
14			
16			
18			
20	*Mode*	*Pkt. Struct. ID*	*Redund. ID*

Table 1

The detailed content of the Source Data field follows:

Byte offset	Field description	Type	Size (bit)	PTC/ PFC
28	Star Sensor Attitude quaternion (1st element, ECI j2000 format)	ANSI/IEEE 754 standard	32	5-1
32	Star Sensor Attitude quaternion (2nd element, ECI j2000 format)	ANSI/IEEE 754 standard	32	5-1
36	Star Sensor Attitude quaternion (3rd element, ECI j2000 format)	ANSI/IEEE 754 standard	32	5-1
40	Star Sensor Attitude quaternion (4th element, ECI j2000 format)	ANSI/IEEE 754 standard	32	5-1
44	GPS position (x coord. in ECI j2000, unit=cm)	32-bit signed number in 2's complement format, most significant byte first	32	4-14
48	GPS position (y coord. in ECI j2000, unit=cm)	32-bit signed number in 2's complement format, most significant byte first	32	4-14
52	GPS position (z coord. in ECI j2000, unit=cm)	32-bit signed number in 2's complement format, most significant byte first	32	4-14

56	configuration + orbital phase	Structured, table not reported	32	
60	Number n (1 <= n <= 73) of single **Events** contained in the packet	Unsigned integer	32	3-14
64	Event 1	Structured, see Table 3	Variable	
……	……	……	……	……
……	……	……	……	……
Variable	Event n	Structured, see Table 3	Variable	

Table 2

Each Event contained in the packet has the following structure:

row	Field description	Type	Size (bit)	PTC/PFC
1	Number of blocks of block type 0, 1, 2 and 3	Structured, see Table 4	32	
2	Time tag of the event (start time)	3 unsigned integers, table not reported	3 * 16 = 48	3-12
3	OBT correction	Unsigned integer	16	3-12
4	Configuration of the triggered top and lateral acquisition chains	Structured, see Table 5	16	
5	MGO and upper thresholds signals (chain 1)	Structured, see Table 6	16	
6	MGO and upper thresholds signals (chain 2)	Structured, see Table 6	16	
7	F_{AC}, F_{MCAL}, F_{BKG}, F_{HT}, F_{BURST}, F_{ST} flags	Structured, table not reported	16	
8	FVC $\underline{X}$ vector	Structured, see Table 7	16	
9	FVC $\underline{Z}$ vector	Structured, see Table 7	16	

10	BLOCK of type 0 (0 <= n. of blocks <= 48)	For the structure of each block element see Table 8	Single element size = 96	
11	BLOCK of type 1 (0 <= n. of blocks <= 48) – same structure of the block of type 0, but contain different data of a different detector	For the structure of each block element see Table 8	Single element size = 96	
12	BLOCK of type 2 (0 <= n. of blocks <= 30)	For the structure of each block element see Table 9	Single element size = 32	
13	BLOCK of type 3 (0 <= n. of blocks <= 192)	For the structure of each block element see Table 10	Single element size = 16	

Table 3: Event

MSB

LSB

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Byte 0	*spare*			*MCAL valid*	*Number of blocks of bock type 0*						*Number of blocks of bock type 1*					
2	*spare*			*Number of blocks of bock type 2*					*Number of blocks of bock type 3*							

Table 4

MSB

LSB

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LAT 11	LAT 10	LAT 9	LAT 8	LAT 7	LAT 6	LAT 5	LAT 4	LAT 3	LAT 2	LAT 1	LAT 0	TOP 2	TOP 1	TOP 0	Spare

Table 5

MSB

LSB

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MGO								*Upper threshold*							

Table 6

MSB

LSB

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
spare				*FVC X or Z vector*											

Table 7

MSB

LSB

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Byte 0 Center Cluster	*FEB (0..11)*				*CHIP (0..23)*					*Strip address (0..127)*						
Byte 2	*Total Charge compressed value (16 bits)*															
Byte 4	*Total Width*								*Central strip charge*							
Byte 6	*Left strip charge (CentralStrip-2)*								*Left strip charge(CentralStrip-1)*							
Byte 8	*Right strip charge(CentralStrip+1)*								*Right strip charge(CentralStrip+2)*							

Table 8: block of type 0 and 1

MSB

LSB

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Byte 0	*spare*			*Bar ID (0..29)*					*Energy side B*							

2	*Energy side B*	*Energy side A*

Table 9: block of type 2

MSB

LSB

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Spare							*FEB Addr.*				*Chip Address*				

Table 10: block of type 3

5. packetlib Supported Packets

Concerning the packet header, the PacketLib is able to handle all the Source Packets which are compliant with the packet header format specified in AD[1] and AD[2].
For what concerns the ability of decoding the various fields contained in the Packet Data Field, the PacketLib is able to handle all the Source Packets having a Packet Data Field structure which can be described by the *.packet configuration file presented in Paragraph 6.5.
The following cases of Packet Data Field structures are envisaged:

- Layout 1: fixed number of fields (structure without blocks);
- Layout 2: fixed/variable number of blocks of fixed dimension;
- Layout 3: fixed/variable number of blocks of variable dimension;
- Layout 4: blocks defined recursively.

As specified in the following subsections, in some cases the Data Field header is required to include some specific information.

1.1. Layout 1 Packets

In this case the Source Data Field is able to contain a fixed number of fields. See example in Paragraph 4.4.
The Layout 1 general structure is shown in Figure 5.1.

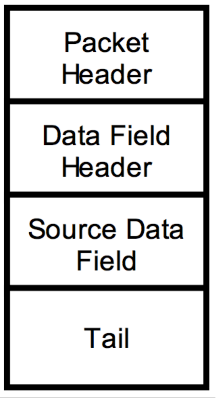

Figure 1: layout 1

No particular field is required in the Packet Data Field.

An example of .packet is reported in the Paragraph 6.5.1

1.2. Layout 2 Packets

In this case, fixed/variable number of blocks of fixed dimension, (see the example reported in Paragraph 4.5) the number of fixed length blocks should be fixed of variable.
As depicted in Figure 5.2, in case of variable number of block, the actual number of Blocks contained in the packet is provided to the PacketLib by a specific field to be foreseen in the Data Field Header.

An example is reported in the Paragraph 4.5. The .packet file structure is reported in the Paragraph 6.5.2.

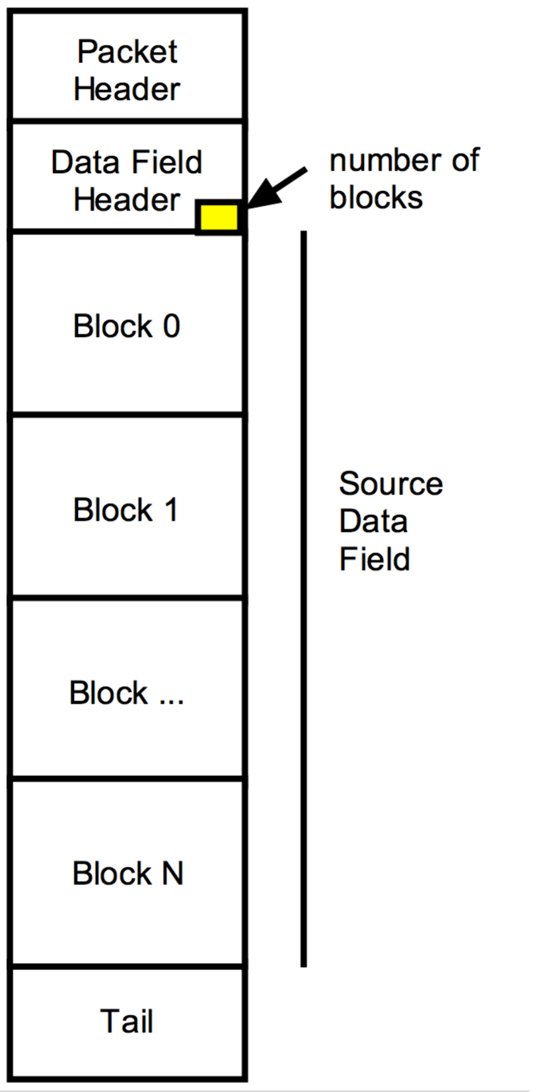

Figure 2: layout 2

1.3. Layout 3 Packets

This is a generalization of Layout 2. In this case (see the example reported in Paragraph 4.6) the Source Data Field is able to contain a fixed or variable number of Blocks with Blocks having variable dimensions.
The resulting structure is depicted in Figure 5.3.

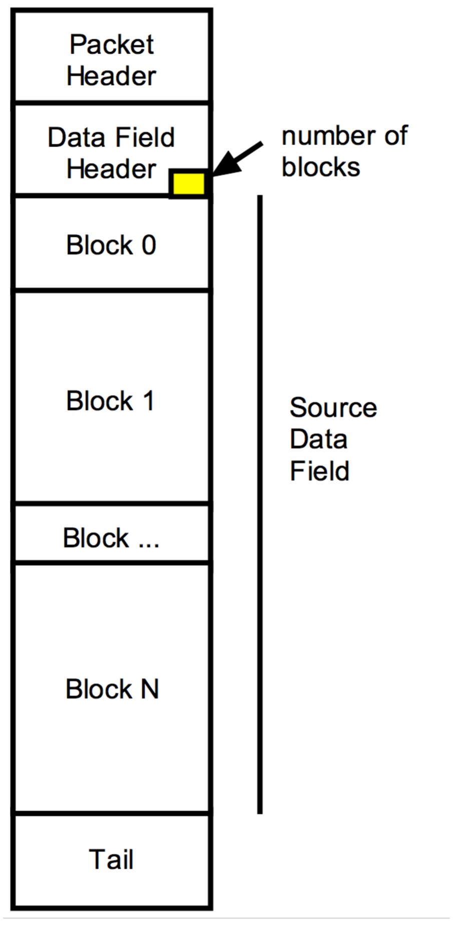

Figure 3: layout 3

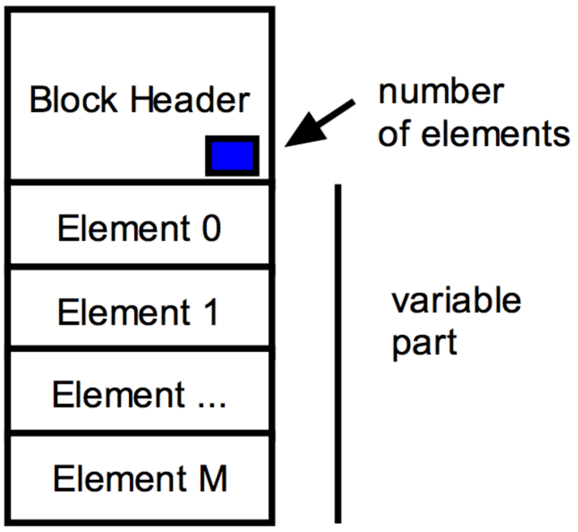

Figure 4: variable block structure

As in the Layout 2 case, the actual number of Blocks contained in the packet must be provided to the PacketLib by a specific field to be foreseen in the Data Field Header (in the case of variable number of blocks).
The Blocks with variable dimensions shall consist of:

- a fixed part, which is the header of the Block which contains all the information that are common for all the elements of the block;
- a sequence of Elements.

The variable dimensions of the blocks leads to require that the fixed part of the block shall contain at least one field which provides the actual number of the elements contained in the block
The remaining characteristics of this type of packets shall be provided to the PacketLib by the *.packet file, as presented in Paragraph 6.5.2.
An example of this type of packet is reported in the Paragraph 4.6.

1.4. Layout 4 Packets

This is the most complex layout, which is the generalization of layouts 1, 2 and 3 that have been introduced for simplicity. This layout is compound by blocks defined recursively. This type of block is called **recursive block** (**rblock**).
The main characteristics of this layout are the following:

A. from the **point of view of the telemetry layout** (used for the definition of the layout of the telemetry):
   1) the packet is compound by header, data field header, source data field and tail
   2) the source data field is compound by rblocks
   3) each rblock is compound by
      a. a fixed part (an header)
      b. a variable part, compound by others rblock of different type; two rblocks are of different types if the structure (number and type of fields) are different.

   The rblock that contains other rblocks is called **rblock container**,
   4) the source data field should be seen as a single rblock. The Source Data Field is compound by a fixed part and by a variable part and the variable part is compound by rblocks.

B. from the **point of view of the instance** of a particular telemetry packet (used to work with a real telemetry packet):
   5) each rblock can contain one or more group of blocks. Each **group** is a set of blocks of a particular type of rblock. Each of these blocks is called **element**. The number of blocks for each group can be different in number: in the current header (fixed part of the rblock container) or in the header of higher levels is specified a field that contain the number of blocks of each type (in the case of rblock type of variable format).
   6) the type of the rblocks cannot be mixed. This mean that the first group of N blocks are of type 0, the second groups of rblock are of type 1 and so on.

This mean that an rblock is compound of a fixed part and a list of blocks (rblocks) divided in group. Each group is a set of blocks of a particular type of rblock.

An example of a structure that can be defined in terms of rblock is reported in the Paragraph 4.7.

1.1.1. Telemetry layout point of view

In the Figure 5.5 is reported an example of a packet compound of N rblocks.
Each rblock is compound of a fixed and a variable part (see Figure 5.6). In the variable part could be present other rblocks.

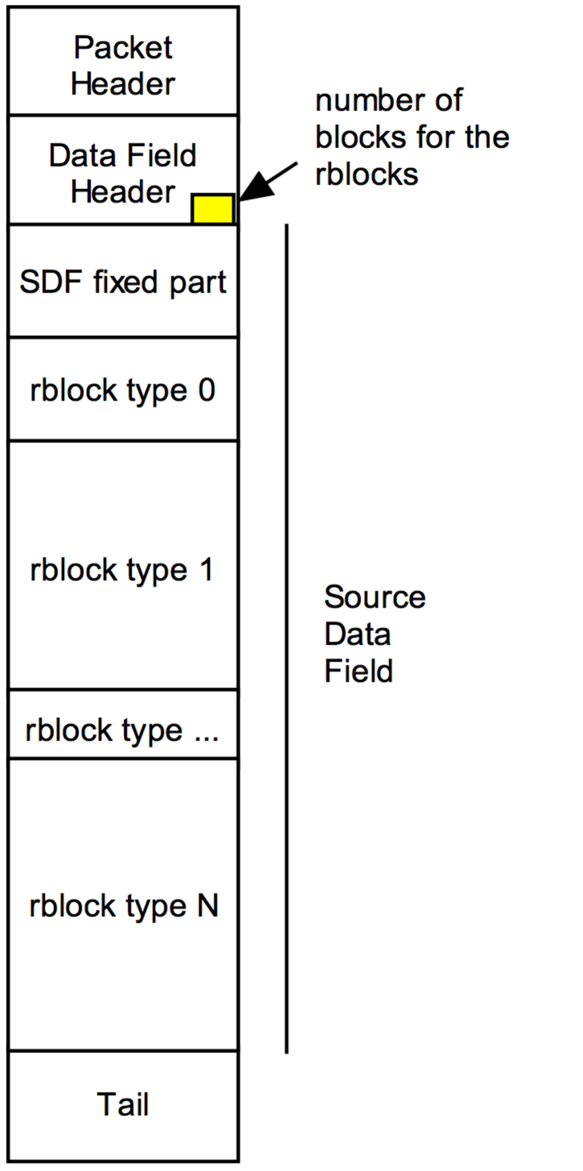

Figure 5: packet with rblock

This from the point of view of the definition of the layout.

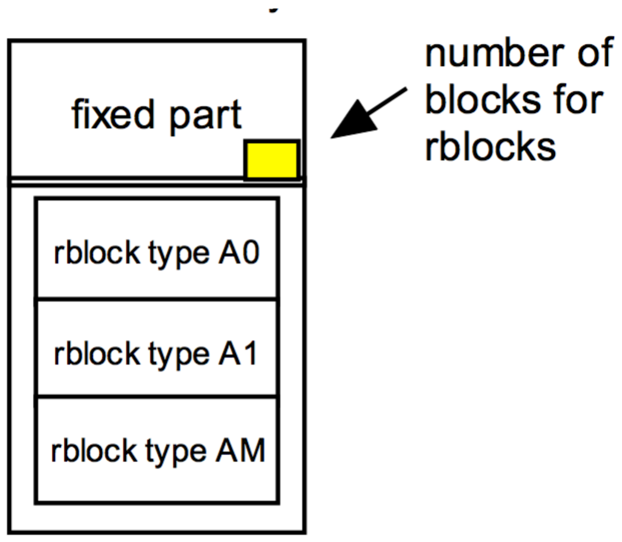

Figure 6: RBLOCK

1.1.2. Telemetry instance point of view

This point of view is used to work with a real telemetry packet defined with the telemetry layout point of view.
The Figure 5.7 shows an example. It should be noted that each element should be an rblock (recursively).

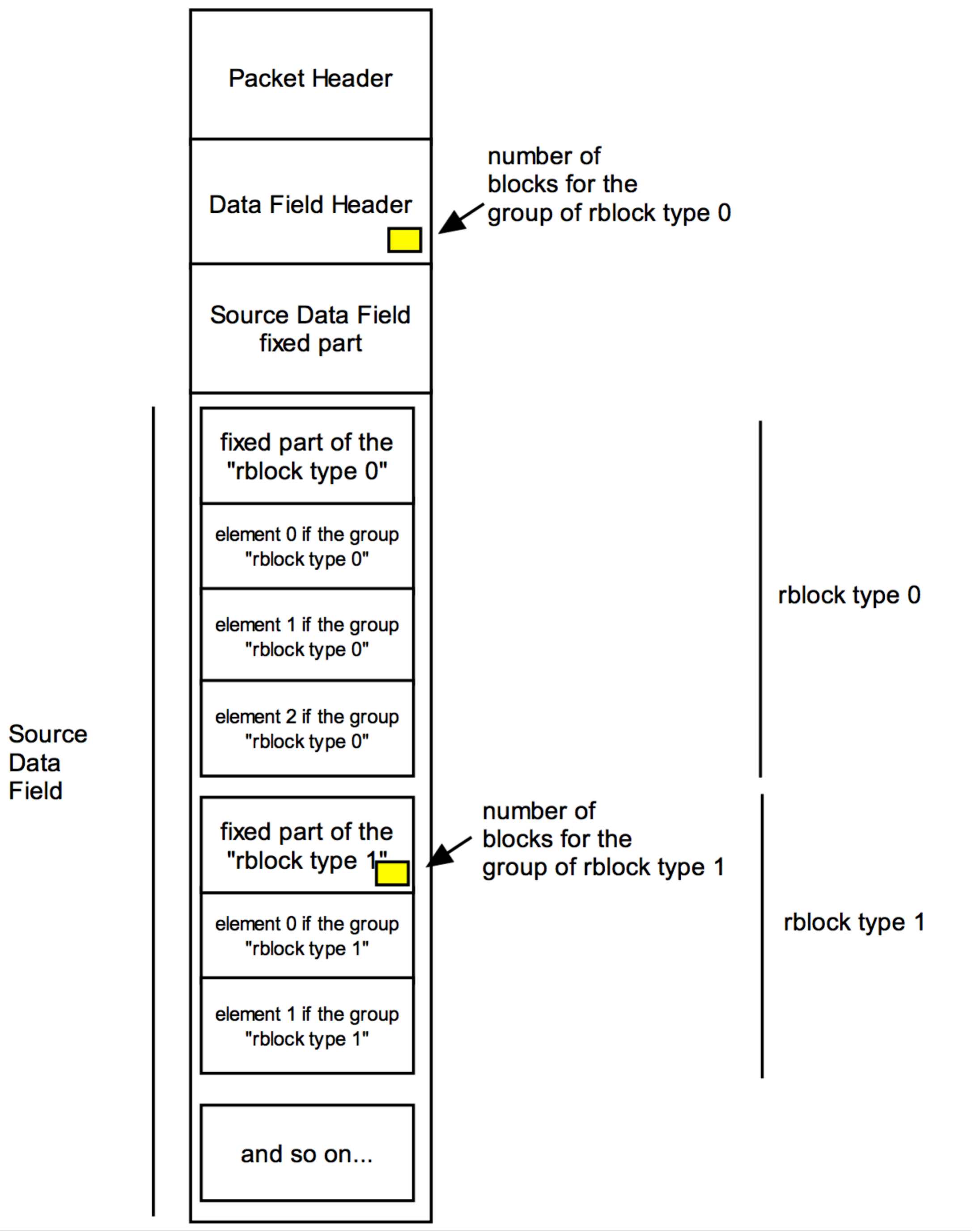
Packet Header
Data Field Header
number of blocks for the group of rblock type 0
Source Data Field fixed part
fixed part of the "rblock type 0"
element 0 if the group "rblock type 0"
element 1 if the group "rblock type 0"
element 2 if the group "rblock type 0"
rblock type 0
Source Data Field
fixed part of the "rblock type 1"
number of blocks for the group of rblock type 1
element 0 if the group "rblock type 1"
element 1 if the group "rblock type 1"
rblock type 1
and so on...

Figure 7: telemetry instance point of view

1.1.3. An example

Let us consider the two point of view for the telemetry layout reported in the Paragraph 4.7.

This telemetry packet is compound by:

1. an header
2. a data field header (Table 1)
3. a source data field (Table 2) compound by
   1.1. a fixed part (the first part of the Table 2)
   1.2. a variable part, compound by a variable number of **events** (Table 3).

The event is the first rblock (that we call rblock of type E in this example). The high view of the packet (telemetry layout point of view) is the following:

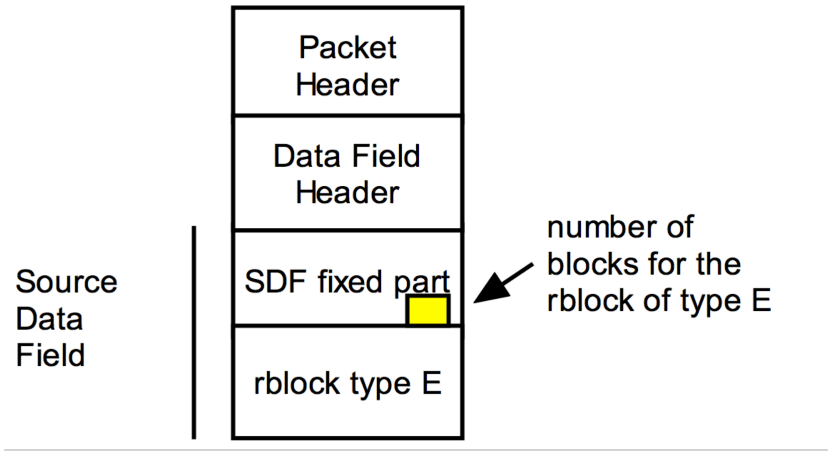

Figure 8

The number of blocks for the rblock of type E is reported in byte offset 60 of Table 2.
As reported in the Table 3: Event, each event is compound by

- a fixed part (first part of the table, from row 1 to row 9)
- a variable part, compound of 4 type of blocks (row 10-13)

In the fixed part are presents some fields (row 1) that contains the number of blocks of each type.
The 4 types of block could be seen as 4 rblocks called rblock type 0 (Table 8), 1 (Table 8), 2 (Table 9) and 3 (Table 10).

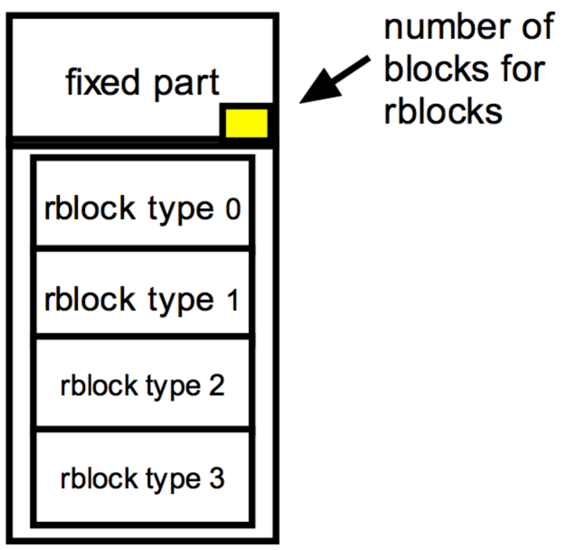

Figure 9: rblock of type E

Finally, each rblock of type 0, 1, 2 and 3 is compound only by a fixed part, and this stops the definition of other embedded rblocks because the variable part is not present.

6. Configuration file specifications

This Chapter specifies the file formats of:

- the *.stream configuration file
- the *.header configuration file which describe the header of the Source Packet contained in the data stream
- the *.packet configuration files which describes the format of the various Source Packet layouts supported by the PacketLib. Related with the .packet are the .rblock, for the layout 4 packet data format.

Both the *.stream, *.header, *.packet and *.rblock files are ASCII files.

1.1. General characteristic

Each element of a configuration file shall have its own line. The comment starts with new line. It is not possible to have elements and comment in the same line.

1.1.1. Comment

Everywhere, these files can contain comments lines, which are lines that starts with two consecutive hyphen characters (--).

1.2. *.stream file format

This in an ASCII file. The file name has extension *.stream.*
Everywhere, the file can contain comments lines, which are lines that starts with two consecutive hyphen characters (-).
The file has three sections:

- Configuration
- Header format: this section contains the file name of *.header configuration file.
- Packet format: in this section lists the file name of *.packet configuration file of the each different Source Packet layout which is expected in the data stream.

The example reported below specifies:

- the keywords which identify each section
- the order and the meaning of the parameters to be provided in each section.

```
[Configuration]
--prefix is present?
true
--stream format bigendian
true
--dimension of prefix
2
[Header Format]
headerESA.header
[Packet Format]
GRID.01.packet
TC_CAL_start.packet
TC_CAL_stop.packet
```

If prefix is false, the dimension of prefix is ignored.

1.3. Packet Fields

Each field of the packet is described by three lines:

- the first line contains the name of field
- the second line contains the dimension, in bit, of the field
- the third line contains either a predefined value (used for writing functionality of PacketLib) or the keyword *none* (this indicates that there is no predefined value).

The predefined values is specified in one of the following formats:

- binary (with 0b prefix)
- decimal (without prefix).

See following examples.

One field with a predefined value:

```
Version number
3
0b100
```

One field without a predefined value:

```
APID
11
none
```

1.4. *.header file format

This file contain the description of the header of the packet. The extension of this file is *.header*.
The file contains:

1) the name of the header
2) the *index* (see the definition below) of the field in the [Field] section which contains the total dimension of the packet in bytes.
3) the [Field] keyword which identifies the Field section
4) the list of the Fields describing the packet header layout

Each field is identified by the *index*, which is defined as its ordinal position within the section (starting from 0).
These specifications are further explained by the *.header file example reported below (see Paragraph 4.3).

```
-- name of header
Header ESA standard
-- number of field with the dimension of packet
6
[Field]
-- field 0
Version number
3
0b100
-- field 1
Type
```

```
1
0b0
-- field 2
DHFH
1
0b1
-- field 3
APID
11
none
-- field 4
SF
2
0b11
-- field 5
Source Sequence Counter
14
none
-- field 6
Packet Length
16
none
```

1.5. *.packet file format

This file contains a description of a single packet. The conventional extension of this file is .packet.

In each .packet the following sections are present

1) name of the packet (the first line of the file)
2) the **[PacketHeader]** section, which refers to the *.header file (see Paragraph 6.4) in order to describe the Packet Header, which is common to all the packets
3) the **[DataFieldHeader]** section, that contains a list of fields that compounds the Data Field Header
4) the **[SourceDataField]** section, different for each Layout, that describes the structure of the Source Data Field
5) the **[Identifiers]**: this section specifies the Fields of the Source Packet which shall be combined by the PacketLib in order to define the identifier which univocally identify the Source Packet itself. Each Field is specified by the following parameters:
    1) the index of the field within either the Packet Header or the Data Field Header or the Source Data Field as specified by the 2nd parameters
    2) the section in which is present the field, 0 for the Packet Header, 1 for the Data Field Header, 2 for the Source Data Field (fixed part).
6) the **[Tail]** section, that contains a list of fields that compound the tail of the packet.

1.1.1. Layout 1

The **[SourceDataField]** section contains the keyword noblock and fixed. This identifies this kind of Layout. The **[SourceDataFieldList]** contains the list of fields.

1.1.1.1. Example

See the Paragraph 4.4 for the definition of this packet.

```
-- name of packet
TC Packet: start
```

```
[PacketHeader]
-- file containing the header description
headerESATC.header
[DataFieldHeader]
--field 0
Spare
2
0b00
-- field 1
Checksum type
2
0b00
-- field 2
ACK
4
0b00
-- field 3
Packet type
4
5
-- field 4
Packet subtype
4
5
[SourceDataField]
noblock
fixed
[SourceDataFieldList]
TC Application Data
16
512
[Identifiers]
-- id0 APID
3

0
257
--id1 type
3
1
5
--id2 subtype
4
1
5
-- id3 command
0
2
512
```

1.1.2. Layout 2 and LAYOUT 3

The *.packet files for Layout 2 and 3 are similar. The **[SourceDataField]** section specifies the following Source Data Field characteristics:

- *block* (keyword): a structure with block
- *fixed/variable* (keyword): Blocks of fixed dimensions (Layout 2) or Blocks of variable dimensions (Layout 3)
- *n* (integer): number of Blocks (in the fixed case) or maximum number of Blocks (in the variable case) contained in the Source Data Field
- *m (*integer): index of the field specified in the [DataFieldHeader] section which gives the actual number Blocks contained in the Source Data Field

- *o* (integer)*:* offset to be added to the value given by the field m of the Data Field Header in order to obtain the correct value of the number of blocks contained in the Source Data Field.

The remaining sections depends on the Layout characteristics.

1.1.1.1. Layout 2 sections

The **[SourceDataFieldBlock]** section contains a list of fields *of each block*.

1.1.1.2. Layout 3 sections

In order to be supported by the PacketLib, the Blocks with variable dimensions shall consist of:

- a fixed part, which is the header of the **block** and contains at least one field which provides the actual number of the elements contained in the block;
- a sequence of **elements**.

E.g.: in the example of Paragraph 4.6, the first 3 words represents the Block header, which is followed by a variable number of Elements.
In the **.packet* file of the Layout 3, the sections are the following:
- the **[SourceDataFieldBlockFixed]** section, which defines the fixed part of the Block providing:
    - *n* (integer): max number of elements in the block
    - *m* (integer): index of the field specified in this section, which gives the actual number of elements contained in the block
    - *o* (integer): offset to be added to the value given by the field *m* of this section in order to obtain the correct value of the number of blocks contained in the Source Data Field.
    4. the list of fields of the fixed part of the Source Data Field

- the **[SourceDataFieldBlockVariable]** section, which defines the element part of the Block in terms of a list of fields.

1.1.1.3. Layout 2 example

See the Paragraph 4.5 for the definition of this packet.

```
-- name of packet
GRID events (LAYOUT 2)
[PacketHeader]
-- file containing the header description
headerESATM.header
[DataFieldHeader]
-- field 0
TM_VERSION
2
1
-- field 1
Spare
3
0
-- field 2: 0 NO, 1 YES
EXT_TIME_TAG
1
none
-- field 3
Checksum flag
```

```
2
0
-- field 4
Packet Type
4
none
-- field 5
Packet subtype
4
none
-- field 6
Time Tag second (MSB)
16
none
-- field 7
Time Tag second (LSB)
16
none
-- field 8
Time Tag millisecond
16
none
-- field 9
Spare
12
0
-- field 10
Nevents
4
none
[SourceDataField]
block
fixed
--type of number of block: fixed = number of block fixed equals to max number of block (fixed |
variable)
variable
--supposed number (or max number) of block
7
--index of field in the [DataFieldHeader] section wich rappresent the number of event (the number of
block) of the packet
10
-- offset to be added to the value given by the field m of the Data Field Header in order to obtain
the correct value of the number of blocks
0
[SourceDataFieldBlock]
-- field 0: OBT (preso da scheda BURST)
TE Internal Counter (MSB)
16
none
-- field 1: OBT
TE Internal Counter (LSB)
16
none
-- bar 01
-- EXT_TIME: counter preso da scheda esterna per sincronizzazione con Acquisition System di INFN/TS
MSB_EXT_TIME_1
4
none
Bar 01 PDA PHA (12 bits)
12
none
MSB_EXT_TIME_2
4
none
Bar 01 PDB PHA (12 bits)
12
none
-- bar 02
LSB_EXT_TIME_1
4
none
```

```
Bar 02 PDA PHA (12 bits)
12
none
LSB_EXT_TIME_2
4
none
Bar 02 PDB PHA (12 bits)
12
none
```

the .packet is not fully reported

```
-- bar 30
spare
4
0
Bar 30 PDA PHA (12 bits)
12
none
spare
4
0
Bar 30 PDB PHA (12 bits)
12
none
[Identifiers]
-- id0 APID
-- field number
3
-- type (0 header, 1 data field)
0
-- value
1294
--id1 type
4
1
15
--id2 subtype
5
1
1
```

1.1.1.4. Layout 3 example

See the Paragraph 4.6 for the definition of this packet.

```
-- name of packet
BURST events (LAYOUT 3)
[PacketHeader]
-- file containing the header description
headerESATM.header
[DataFieldHeader]
-- field 0
Spare
6
none
-- field 1
Checksum flag
2
none
-- field 2
Packet Type
4
none
-- field 3
Packet subtype
4
none
-- field 4
Time Tag second (MSB)
```

```
16
none
-- field 5
Time Tag second (LSB)
16
none
-- field 6
Time Tag millisecond
16
none
-- field 7
Spare
12
none
-- field 8
Nevents
4
none
[SourceDataField]
block
variable
--type of number of block: fixed = number of block fixed equals to max number of block (fixed |
variable)
variable
--supposed number (or max number) of block
79
--index of field in the [DataFieldHeader] section wich rappresent the number of event (the number of
block) of the packet
8
-- offset to be added to the value given by the field m of the Data Field Header in order to obtain
the correct value of the number of blocks
0
[SourceDataFieldBlockFixed]
-- number (or max number) of element
32
-- index of field in the [SourceDataFieldBlockFixed] section wich rappresent the number of element
(the number of bar) of the block
2
-- valore da sommare per ottenere il numero di element reali
1
-- field 0
1
1
none
-- field 1
spare
10
none
-- field 2
Number of bars - 1
5
none
-- field 3
TE Internal Counter (MSB)
16
none
-- field 4
TE Internal Counter (LSB)
16
none
[SourceDataFieldBlockVariable]
-- field 0
0
1
0
-- field 1
0
1
0
-- field 2
Bar ID (2 MSB)
```

```
2
none
-- field 3
PDA PHA (12 bits)
12
none
-- field 4
0
1
0
-- field 5
Bar ID (3 LSB)
3
none
-- field 6
PDB PHA (12 bits)
12
none
[Identifiers]
-- id0 APID
-- field number
3
-- type (0 header, 1 data field)
0
-- value
1294
--id1 type
2
1
15
--id2 subtype
3
1
3
```

1.1.3. Layout 4

The Source Data Field section is compound only by the keyword **rblock**.
After this, the definition of an rblock follow. This defintion is compound by the following sections:

1. **[RBlock Configuration]** with the following lines:
   1. specifies whether the fixed part is present (yes | no)
   2. specifies whether the variable part is present (yes | no). If yes, add a [RBlockX] sections (see below)
   3. an integer that specifies the number of [RBlockX] section (if variable part is present)
2. **[Fixed Part]** that contains a list of fields and represents the fixed part of a rblock.
3. A list of **[RblockX]** section (X starting from 1). Each of this section contains:
   1. n (integer): type of number of blocks of this variable part: fixed = number of block fixed equals to max number of block (fixed | variable)
   2. m (integer): number of blocks for fixed value into variable part, max number of blocks for variable value into variable part
   3. o (integer): for variable block, number of level of headers in which is present the field with the number of blocks of the variable part (0: fixed part)
   4. p (integer): for variable block, index of field of the header which represents the number of events (the number of blocks) of the packet

5. q (integer): offset to be added to the value given by the field p of this section in order to obtain the correct value of the number of blocks
6. (string): file name of the rblock

If the fixed part is not present, also the **[Fixed Part]** section is not present. If the variable part is not present, also the **[RblockX]** sections are not present.

1.1.1.1. LAYOUT 4 EXAMPLE

See the Paragraph 4.7 for the definition of this packet and Paragraph 5.4.3 for an explanation of the layout of the packet. Note that:

- the file 39_01.packet contains the definition of the Header, Data Field Header, SDF Fixed part and a link to the definition of the rblock type E
- the file rblock_type_E.rblock contains defintion of the rblock type E and a link to the definition of the contained rblock type 0, 1, 2 and 3
- the files rblock_type_0_1.rblock, rblock_type_2.rblock, rblock_type_3.rblock contain the definition of the rblock type 0, 1, 2 and 3

39_01.packet

```
-- name of packet
39.01
[PacketHeader]
-- file in cui e' presente la descrizione dell'header
headerESATM.header
[DataFieldHeader]
-- field 0
Sync Marker MSB
16
65499
-- field 1
Sync Marker LSB
16
37461
-- field 2
CRC flag
2
0b11
-- field 3
Packet Type
6
none
-- field 4
Packet Subtype
8
1
-- field 5
MM Page ID
16
none
--field 6
MM Packet Counter
16
none
--field 7
Century
8
32
--field 8
APID Sequence Counter
8
nome
--field 9
```

```
OBT12
8
none
--field 10
OBT13
8
none
--field 11
OBT14
8
none
--field 12
OBT15
8
none
--field 13
OBT16
8
none
--field 14
Not used
4
0
--field 15
OBT17
4
none
--field 16
OBT18
8
none
--field 17
OBT19
8
none
--field 18
Mode
8
none
--field 19
Pkt. Struct ID
4
1
-- field 20
Redundancy ID
4
1
[SourceDataField]
-- type of packet: noblock, block, rblock (with recoursive block)
rblock
[RBlock Configuration]
-- fixed part present (yes | no)
yes
-- variable part present (yes | no). If yes, add [RBlockX] sections.
yes
--number of rblock (if variable part is present)
1
[Fixed Part]
--field 0 - SOURCE DATA FIELD HEADER ----------------------------------------
Quaternion 1H
16
none
--field 1: parte bassa del float
Quaternion 1L
16
none
--field 2: parte alta del float
Quaternion 2H
16
none
--field 3: parte bassa del float
```

```
Quaternion 2L
16
none
--field 4: parte alta del float
Quaternion 3H
16
none
--field 5: parte bassa del float
Quaternion 3L
16
none
--field 6: parte alta del float
Quaternion 4H
16
none
--field 7: parte bassa del float
Quaternion 4L
16
none
--field 8: parte alta
GPS Position XH
16
none
--field 9: parte bassa
GPS Position XL
16
none
--field 10: parte alta
GPS Position YH
16
none
--field 11: parte bassa
GPS Position YL
16
none
--field 12: parte alta
GPS Position ZH
16
none
--field 13: parte bassa
GPS Position ZL
16
none
--field 14
Spare OCID
4
0
--field 15
Orbital phase
2
none
--field 16
Recovery sub-phase
1
none
--field 17
Grid configuration mode
4
none
--field 18
LUT 3/4 X
3
none
--field 19
LUT 3/4 Z: H
2
none
--field 20
LUT 3/4 Z: L
1
none
```

```
--field 21
LUT AC
3
none
--field 22
LUT R-TRIGGER
3
none
--field 23
LUT DIS
3
none
--field 24
LUT COEF
3
none
--field 25
LUT C-DIS
3
none
--field 26 (Number of grid events 1<=n<=73)
Number of Events
16
none
[RBlock1]
--type of number of blocks of this variable part: fixed = number of block fixed equals to max number
of block (fixed | variable)
variable
--number of blocks for fixed value into variable part, max number of blocks for variable value into
variable part (73)
73
-- for variable block, number of level of headers in which is present the field with the number of
blocks of the variable part (0: fixed part)
0
-- for variable block, index of field of the header which rappresent the number of events (the number
of blocks) of the packet
26
-- offset to be added to the value given by the field above of this section in order to obtain the
correct value of the number of blocks
0
-- file name of the rblock
rblock_type_E.rblock
[Identifiers]
-- ID0 APID
-- field number
3
-- type (0 header, 1 data field)
0
-- value
767
-- ID1 type
3
1
39
-- ID2 subtype
4
1
1
[Tail]
CRC
16
none
```

rblock_type_E.rblock

```
[RBlock Configuration]
-- fixed part present (yes | no)
yes
-- variable part present (yes | no). If yes, add [RBlockX] sections.
yes
--number of rblock (if variable part is present)
```

```
4
[Fixed Part]
-- field 0
Spare
3
0
-- field 1
MCAL valid
1
none
-- field 2
Number of cluster X elements
6
none
-- field 3
Number of cluster Z elements
6
none
-- field 4
Spare
3
0
-- field 5
Number of zero-suppressed elements
5
none
-- field 6
Number of extra-fired elements
8
none
-- field 7
Spare
1
0
-- field 8
Time Tag MSW: second, H
15
none
-- field 9
Time Tag mSW: second, L
12
none
-- field 10
Microseconds, H
4
none
-- field 11
Time Tag LSW: microseconds, L
16
none
-- field 12: OBTCORR
OBT Correction
16
none
-- field 13: ACTOPCON + ACLATCON + MCAL_HT: ACL11, ACL10, ..., ACL0, ACT2, ACT1, ACT0, MHT
Configuration (Top&Lat)
16
none
-- field 14
S 1 MGO
8
none
-- field 15
S 1 Upper thres
8
none
-- field 16
S 2 MGO
8
none
-- field 17
```

```
S 2 Upper thres
8
none
-- field 18
Spare
8
none
-- field 19
Spare
2
0
-- field 20
F_ht
1
none
-- field 21
F_burst
1
none
-- field 22
F_st
1
none
-- field 23
F_bkg
1
none
-- field 24
F_mcal
1
none
-- field 25
F_ac
1
none
-- field 26
Spare
4
0
-- field 27
FVC X Vector
12
none
-- field 28
Spare
4
0
-- field 29
FVC Z Vector
12
1
[RBlock1]
--type of number of blocks of this variable part: fixed = number of block fixed equals to max number
of block (fixed | variable)
variable
--number of blocks for fixed value into variable part, max number of blocks for variable value into
variable part (48)
48
-- for variable block, number of level of headers in which is present the field with the number of
blocks of the variable part (0: fixed part)
0
-- for variable block, index of field which rappresent the number of event (the number of block) of
the packet
2
-- for variable block, valore da sommare per ottenere il numero di eventi (blocchi) reali
0
rblock_type_0_1.rblock
[RBlock2]
--type of number of blocks of this variable part: fixed = number of block fixed equals to max number
of block (fixed | variable)
variable
```

```
--number of blocks for fixed value into variable part, max number of blocks for variable value into
variable part (48)
48
-- for variable block, number of level of headers in which is present the field with the number of
blocks of the variable part (0: fixed part)
0
-- for variable block, index of field which rappresent the number of event (the number of block) of
the packet
3
-- for variable block, valore da sommare per ottenere il numero di eventi (blocchi) reali
0
rblock_type_0_1.rblock
[RBlock3]
--type of number of blocks of this variable part: fixed = number of block fixed equals to max number
of block (fixed | variable)
variable
--number of blocks for fixed value into variable part, max number of blocks for variable value into
variable part (30)
30
-- for variable block, number of level of headers in which is present the field with the number of
blocks of the variable part (0: fixed part)
0
-- for variable block, index of field which rappresent the number of event (the number of block) of
the packet
5
-- for variable block, valore da sommare per ottenere il numero di eventi (blocchi) reali
0
rblock_type_2.rblock
[RBlock4]
--type of number of blocks of this variable part: fixed = number of block fixed equals to max number
of block (fixed | variable)
variable
--number of blocks for fixed value into variable part, max number of blocks for variable value into
variable part (192)
192
-- for variable block, number of level of headers in which is present the field with the number of
blocks of the variable part (0: fixed part)
0
-- for variable block, index of field which rappresent the number of event (the number of block) of
the packet
6
-- for variable block, valore da sommare per ottenere il numero di eventi (blocchi) reali
0
rblock_type_3.rblock
```

rblock_type_0_1.rblock

```
[RBlock Configuration]
-- fixed part present (yes | no)
yes
-- variable part present (yes | no)
no
[Fixed Part]
-- field 0
FEB
4
none
-- field 1
CHIP
5
none
-- field 2
Strip address
7
none
-- field 3
Total charge compressed value
16
none
-- field 4
Total width
8
none
```

```
-- field 5
Central strip charge (XSTRIPC3)
8
none
-- field 6
Left strip charge (central strip - 2) (XSTRIPC1)
8
none
-- field 7
Left strip charge (central strip - 1) (XSTRIPC2)
8
none
-- field 8
Right strip charge (central strip + 1) (XSTRIPC4)
8
none
-- field 9
Right strip charge (central strip + 2) (XSTRIPC5)
8
none
```

rblock_type_2.rblock

```
[RBlock Configuration]
-- fixed part present (yes | no)
yes
-- variable part present (yes | no)
no
[Fixed Part]
-- field 0
Spare
3
0
-- field 1
Bar ID
5
none
-- field 2
Energy Side B
8
none
-- field 3
Energy Side B
4
none
-- field 4
Energy Side A
12
none
```

rblock_type_3.rblock

```
[RBlock Configuration]
-- fixed part present (yes | no)
yes
-- variable part present (yes | no)
no
[Fixed Part]
-- field 0
Spare
7
0
-- field 1
FEB Addr.
4
none
-- field 2
Chip Addr.
5
none
```