



PacketLib Tutorial: how to create simple telemetry generator software using the PacketLib

Internal Report IASF Bologna n 612/2012

Prepared by: Name: V. Conforti Signature: Date: 25/09/2012
A. Bulgarelli
M. Trifoglio
F. Gianotti

Reviewed by: Name: A. Bulgarelli Signature: _____ Date: _____

Approved by: Name: M. Trifoglio Signature: _____ Date: _____



DISTRIBUTION LIST

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



DOCUMENT HISTORY

Version	Date	Modification
1.0	25 September 2012	First version




List of Acronyms

I/O	Input/Output
APID	Application ID
DHFH	Data Field Header Flag
TC	Telecommand
TM	Telemetry
PTC	Parameter Type Code
PFC	Parameter Format Code
PC	Parameter Code
PDM	Photon Detection Module

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-004	Issue:	1.0	DATE	25-SEP-12	Page:	1

TABLE OF CONTENTS

LIST OF ACRONYMS	4
1. INTRODUCTION	2
2. THE PACKETLIB	3
3. SOFTWARE ARCHITECTURE	4
4. TELEMETRY STRUCTURE	5
4.1 PACKETLIB STRUCTURE RULES	6
4.2 TUTORIAL STRUCTURE	7
5. COMPLEX TYPE	11
6. ENCODER IMPLEMENTATION	12
7. REFERENCE DOCUMENTS	15
8. ANNEX A - TELEMETRY GENERATOR CODE	16

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-004	Issue:	1.0	DATE	25-SEP-12	Page:	2


1. Introduction

The terms telemetry is intended as a data flow in binary format widely used to exchange data between astrophysics instruments.

Since the telemetry is a byte stream is crucial to have an encoder/decoder system which transform the information in byte stream and reverse.

The *PacketLib* is a library that allows to implement a telemetry generator easily. With this library is also possible to generate telemetry in according to ESA Standard Packet.

This document is intended as a tutorial to implement a simple telemetry generator.

<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-004	Issue:	1.0	DATE	25-SEP-12	Page:	3

2. The PacketLib

The PacketLib C++ library has been developed in order to facilitate the development of applications which handle data streams containing Source Packet compliant to the ESA Telemetry Standard as defined by [AD1].

This library is used by applications which require to decode or encode the data fields of the raw format packets.

PacketLib has many I/O layer in order to support many device (file, socket, shared memory).

The packet must have 3 sections:

- Header
- Data field header
- Source Data Field

The Header contains the common information about the telemetry flow.

The Data Field Header contains the meta information about the packet.

The Source Data Field contains the significant data detected by instrument.

PacketLib allows to define 4 layout of packets:

1. Layout 1: each packet has a fixed number of fields;
2. Layout 2: each packet has a fixed or variable number of fixed size blocks;
3. Layout 3: each packet has a fixed or variable number of variable size blocks ;
4. Layout 4: each packets define the block structure recursively.

It can define the structure telemetry using configuration files without implementing any hard-coded information.

In this tutorial will be presented a telemetry generator of packets with layout 2: each packet has a fixed number of blocks of fixed size.

The PacketLib has also a component which manage the exception when the program is run for read/write operation.

The main classes involve to manage the byte stream are :

- **Input Packet Stream** class represents a packet stream as input (useful for reading a stream);
- **Output Packet Stream** class represents a packet stream as output (useful for writing a stream);
- **Input** class reads physically the input from a different source (File, Socket) ;
- **Output** class writes the output on many source (File, Socket);
- **Packet** class is the core of the library because it represents a TM o TC packet;
- **Block** is the child class of Packet and allow to manage the blocks of its packet.

3. Software Architecture

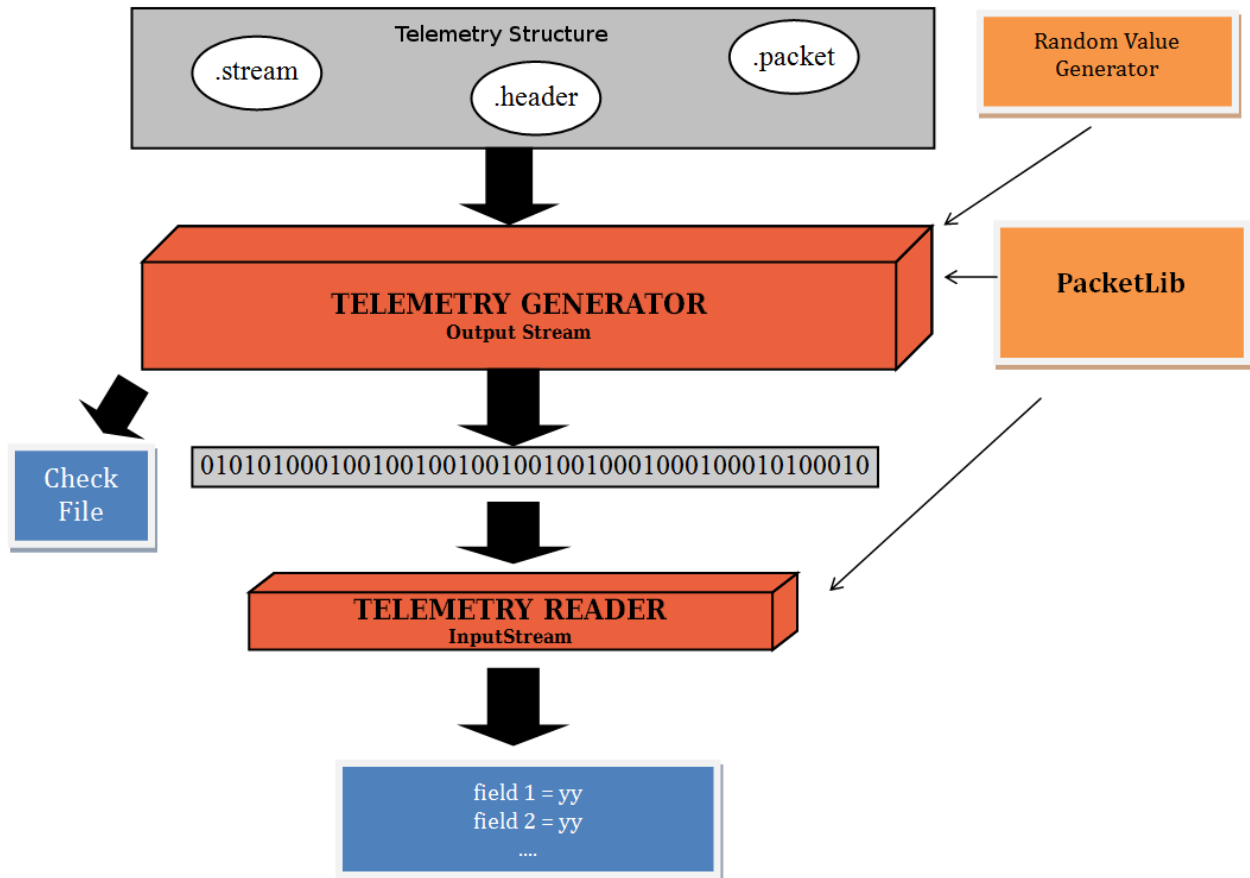


Figure 1 - Software Architecture

The gray component represents the telemetry file that will be passed in input to telemetry generator. The telemetry structure is defined from 3 kind o file:

1. *.stream file;
2. *.header file;
3. *.packet files.

In the complex telemetry flow can be used many *.packet files in order to distinguish different type of data of flow. For example it can be defined a kind of packet for the start telecommand, a kind of packet for the stop telecommand, a kind a packet of a instrument detector and so on. In this simple tutorial will be used only a *.packet file.

As this tutorial presents a simple telemetry generator , the values to insert in the telemetry file will be chosen casually. In order to test the software each random value inserted will be saved also in a *Check File*. The main program will use the PacketLib's method to write the telemetry file. After the creation of telemetry file, it's will be opened and read with another software to view the values just inserted.

4. Telemetry Structure

To define the telemetry structure means essentially to decide how the packet will be formed.

In this tutorial the data structure is defined by 3 files:

- *STRUCT.stream*;
- *headerSIMTEL.header*;
- *format.packet*.

The *STRUCT.stream* file represents the telemetry flow. It contains the information on the endianness of the data and the reference to the header section and the packets section.

The *headerSIMTEL.header* file describes the packet header information.

The *format.packet* files describes the packet configuration.

In this simple tutorial will be generated a telemetry file in according to the following structure:

Packet Header (3 x 16 bit word)

Version Number	Type	DHFH	APID
Source Sequence Counter			
Packet length			

Data Field Header (3x16 bit word - TBC)

Event time-tag (4 bytes)	PDM Address (1 bytes)	Pixel ID (1 bytes)
--------------------------	-----------------------	--------------------


Data Field (200 bytes)

Sample 1	Sample 2	...	Sample 100
----------	----------	-----	------------

Figure 2 - Packet Structure

The packet header used in this tutorial is equal to the standard ESA packet header:

- Version Number;
- Type;
- DHFH (Data Field Header Flag);
- APID (Application Process Identifier);
- Source Sequence Counter (counter of packets with same APID);
- Packet length (number of octets in Packet Data Field less 1);

<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-004	Issue:	1.0	DATE	25-SEP-12	Page:	6

The Data Field Header in according to the structure shown in Figure 2 has 3 fields:

- Event time tag (the time of detection of the event);
- PDM Address;
- Pixel ID (the identifier of pixel involved).

The Data Field is composed by 100 blocks which represents the samples detected.

4.1 PacketLib Structure Rules

In according to PacketLib [RD1] the telemetry structure files *.header and *.packet must satisfy this rules:

1. The total length of packet must be 16 bit or its multiple;
2. Each field must be have length 16 bit or less;
3. Each field must be defined by:
 - name of field,
 - dimension expressed in bit,
 - default value (expressed in binary or decimal format). It can be used the *none* value which indicates that there is not a predefined valued;
4. The comment are accepted. It must be have 2 score as prefix (*-- text of comment*);
5. The name of sections must be enclosed in square brackets;
6. The structure file must be edited with a simple text editor (not rich-text);


If the structure just defined is not length 16 bit or its multiple, It must be added the *spare* field to respect the rule 1.

If It need to has a field with length more of 16 bit, then It must be created two fields having respectively the suffixes: MSB (Most Significant Bit) and LSB (Least Significant Bit). The methods which use this kinds of field must be refer always to the MSB field and the PacketLib will consider correctly the entire value. In the next section will be explained the declaration of these complex types.

An example of field in a section is:

```
[DataFieldHeader]
--field 1 , here It is reported the space needed to respect the rule 1
Spare
8
0b00
```

Where the first row indicates the begin of *Data Field Header* section. The second line is a comment. It is advised uses always as a comment the index of field because It will be used in the PacketLib's methods. In the above example *Spare* is the name of field with length 8 bit. The last row is the default value that is 0. The prefix *0b* means that the value is expressed in binary format. Since

<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-004	Issue:	1.0	DATE	25-SEP-12	Page:	7

some field could be update automatically from PacketLib's methods, It is advised uses *none* as the default value in this cases.

4.2 Tutorial Structure

The stream.file has 3 sections:

- configuration;
- header format (with the name of header configuration file);
- packet format (with the packet configuration file).


Following the stream file content:

```
[Configuration]
--prefix
true
--stream format bigendian
true
--dimension of prefix
2
[Header Format]
headerSIMTEL.header
[Packet Format]
format.packet
```

In the configuration section It is declared the big endian stream format [RD4], and the use of prefix with dimension 2 bit. The prefix will be used for example to express a binary format in a field declaration. The *Header Format* and the *Packet Format* have the reference to respective files.

The header file (*headerSIMTEL.header*) which contain the description of the packet header is the following:

```
-- name of header
Header ESA standard
-- number of field with dimension of packet
5
[Field]
-- field 0
Version number
3
0b001
-- field 1
Type
1
0b1
```

<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-004	Issue:	1.0	DATE	25-SEP-12	Page:	8

```
-- field 2
DFHF
1
0b1
-- field 3
APID
11
1294
-- field 4
Source Sequence Counter
14
none
-- field 5
Packet Length
16
none
```

The packet file format contains a description of single packet. The *format.packet* files has the following sections:

- The name of the packet (in the first line);
- The *PacketHeader* section which refers to the header file;
- The *DataFieldHeader* sections that contains a list of fields that compounds the Data Field Header;
- The *SourceDataField*, different for each layout, It describes the structure of the Source Data Field.
- Identifiers section reports the fields of the source packet which shall be combined by the PacketLib in order to define an identifier which univocally identify the Source Packet itself.
- Tail section contain a list of fields that compound the tail of the packet.

In this simple example the packet has not the tail.

The structure of this tutorial uses the layout 2 that is packet with a fixed number of blocks with fixed size:

```
-- name of packet
tel pack tutorial
[PacketHeader]
-- name of header file
headerSIMTEL.header
[DataFieldHeader]
--field 0
Spare
```



```

8
0b00
--field 1
EventTimeTagMSB
16
0b00
-- field 2
EventTimeTagLSB
16
0b00
-- field 3
PDMAddress
8
0b00
-- field 4
PixelID
8
0b00
-- field 5 is the number of blocks included in current packet
Nevents
8
none
[SourceDataField]
-- layout 2 define a structure with block
block
-- the dimension of blocks that is fixed
fixed
-- the number of block that is fixed
fixed
-- number of blocks
100
-- index of field nevents in DataFieldHeader section that counts the blocks
5
-- offset (offset to be added to nevents to obtain the correct value - not use in this tutorial)
0
[SourceDataFieldBlock]
sample
16
0b00
[Identifiers]
-- id0 APID
3
0
1294

```

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling				
	Code: CIWS-IASFBO-TN-004	Issue:	1.0	DATE	25-SEP-12	Page: 10

The *nevents* field in Data Field Header section counts the blocks. It is updated each time a block is added. In Source Data Filed section it is declared the fixed block structure with 100 blocks, one for each sample.

Since this tutorial uses layout 2 there is a section *SourceDataFieldBlock* which defines the structure of the block. Each block has only a field of length 16 bit and 0 as default value. It represents the value of sample.

The *identifiers* sections of this tutorial is declared by once fields. The parameter are the following:

- The index of field in the file (that is 3 because the APID is the third field);
- The section of field. Are available 3 options:
 - a. 0: if the identifier field is placed in *.header file (that is the tutorial case);
 - b. 1: if the identifier field is placed in *DataFieldHeader* section;
 - c. 2: if the identifier field is placed in *SourceDataField* section;
- The value of field identifier that will be processed (that is 1294 for this tutorial).

It should be noted that the choice of dimension fields is crucial:

- **The under sizing causes loss of data;**
- **The over sizing causes performance decrease.**

5. Complex Type

In this simple tutorial there is not field of complex type data but It is useful to provide a reference in this tutorial.

In the *PacketLib Programmers Guide* document [RD2] are shown the methods which allow to get and to set telemetry fields:

- `getFieldValue(word index);`
- `setFieldValue(word index, float value);`

There are some specialized method, actually implemented, for some particular type (PTC, PFC) :

- `getFieldValue_5_1(word index);`
- `setFieldValue_5_1(word index, float value);`
- `getFieldValue_4_14(word index);`
- `setFieldValue_4_14(word index, long value);`
- `getFieldValue_4_13(word index);`
- `setFieldValue_4_13(word index, long value);`
- `getFieldValue_3_14(word index);`
- `setFieldValue_3_14(word index, long value);`
- `getFieldValue_3_13(word index);`
- `setFieldValue_3_13(word index, long value);`

The details of these specialized methods are reported in [RD2].

In order to use the above methods it is necessary define correctly the dimension of these fields. The following table show for each combination of PTC and PFC (used in PacketLib methods) the format definition (number of bit to assign to the field) and the range of the allowed values:

PTC	PFC	Format Definition	Lowest Value	Highest Value
5	1	4 octets, simple precision	$1,2 * 10^{-38}$	$3,4 * 10^{38}$
4	14	4 octets, signed integer	-2^{31}	2^{31-1}
4	13	3 octets, signed integer	- 8.388.608	8.388.607
3	14	4 octets, unsigned integer	0	$4,3 * 10^9$
3	13	3 octets, unsigned integer	0	16.777.215

The details about complex type are reported in [RD3].

6. Implementation

The software of this tutorial is very simple and implemented completely in a single *main.cpp* file. It can see the complete and documented code implementation in Annex A. Following the activity diagram:

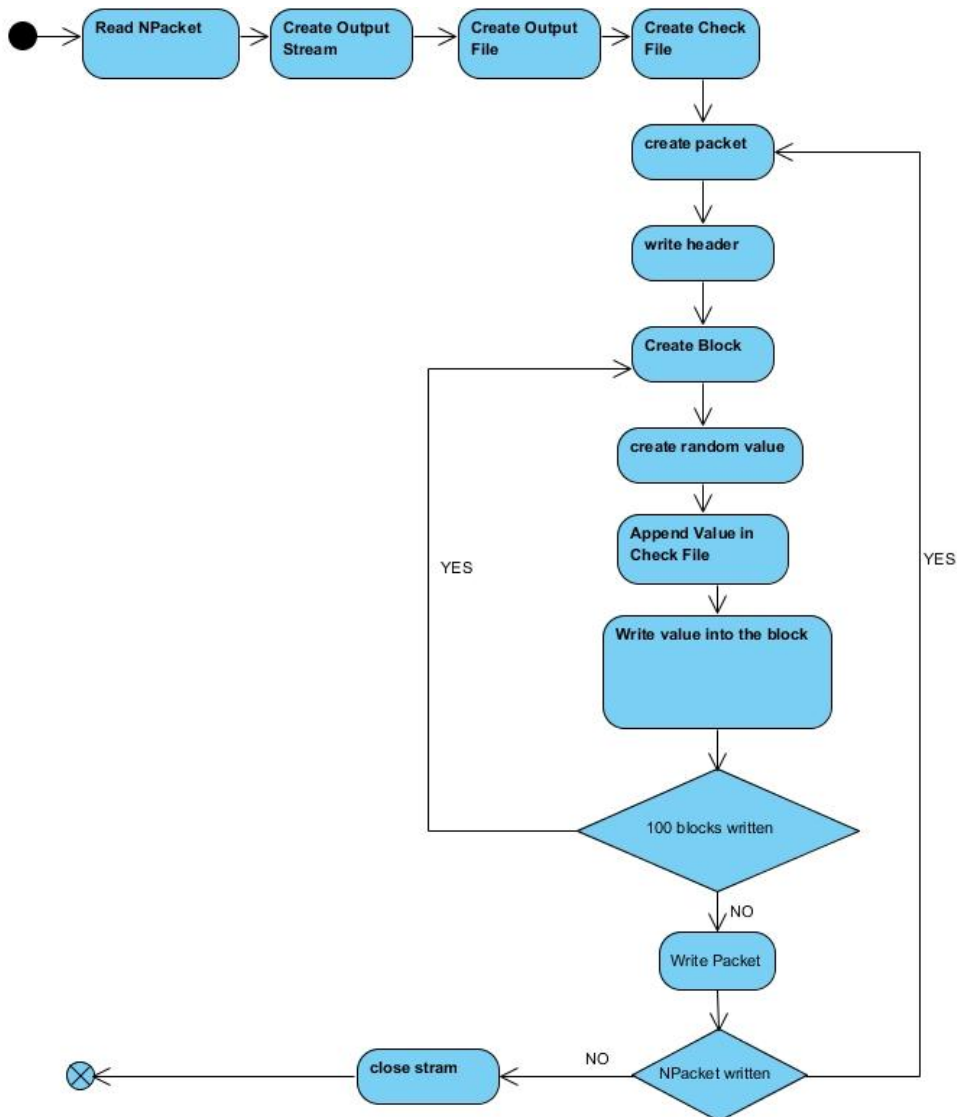



Figure 3 The activity Diagram

NPacket is the number of packet that will be created. It must be specified by the user in the run string.

<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling				
	Code: CIWS-IASFBO-TN-004	Issue:	1.0	DATE	25-SEP-12	Page: 13

The main PacketLib's method used to write data in the packet fields is the *setFieldValue* method.

Some example of *setFieldValue* methods are:

```
p->header->setFieldValue(5,1); // write 1 in the 5th field of the header packet

p->dataField->dataFieldHeader->setFieldValue_3_14(1, 5000); // write on data field header

p->dataField->sourceDataField->setFieldValue(5 ,10); // write on source data field of packet
```

Where p is the packet, followed by the section:

- Header;
- Data Field:
 - Data Field Header;
 - Source Data Field.

This instructions allow to point directly the specified field in according to telemetry structure previously defined.

This method require two parameter:

- The index of field;
- The value to write.

At the end of the execution program It can be found the telemetry file generated in the *output* folder. To verify the correct execution It could be compare the telemetry file with the Check File. In this tutorial the telemetry file and the Check File are respectively named *telemetry_generated.raw* and *TelemetryValue.txt*.

Since the telemetry file contains values in binary format, It can be use a graphical software that shows the real values: the *Packet Viewer*.

The figure 4 shows the telemetry just created using *Packet Viewer* software.

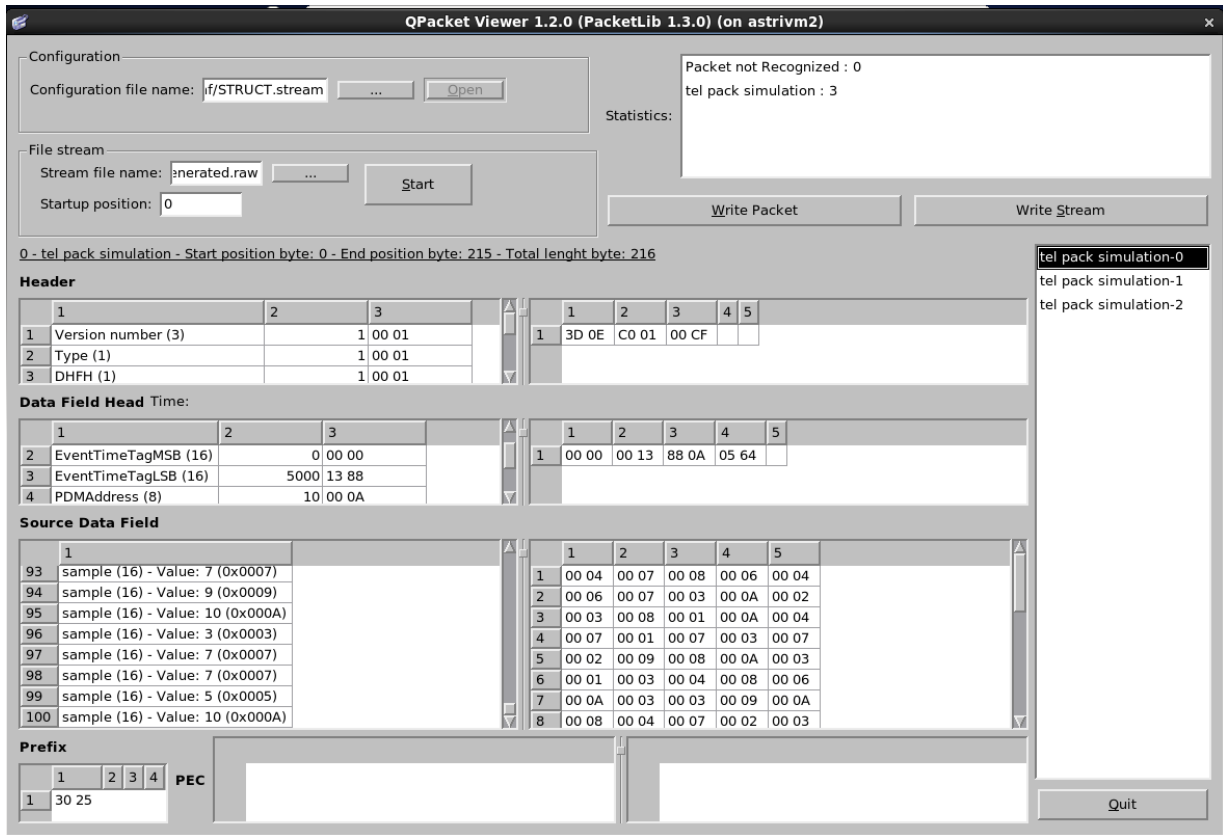


Figure 4 Packet Viewer

The interface of the Packet Viewer is very simple. It must be inserted the *.stream file in the configuration box and the telemetry file in the File Stream box. Then It can click on *Start* button and the other boxes appears:

- Statistics box reports the number of packets comply to the telemetry structure set in configuration box;
- Header, Data Field Header and Source Data Field show the content of packets. The values are both decimal and hexadecimal;
- The right box has the list of all packets. It allow to select a packet to view its contents.

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling				
	Code: CIWS-IASFBO-TN-004	Issue:	1.0	DATE	25-SEP-12	Page: 15

7. Reference documents

- RD [1] PacketLib 1.3.2 Interface Control Document - A. Bulgarelli, F. Gianotti, M. Trifoglio - February 2005.
- RD [2] PacketLib 1.3.6 Programmer's Guide A. Bulgarelli, F. Gianotti, M. Trifoglio - July 2005.
- RD[3] Space Engineering Ground systems and operations - telemetry and telecommand packet utilization. - European Cooperation ECSS for space standardization- ECSS-E-70-41A - January 2003.
- RD[4] White Paper: Endiannes or Where is Byte 0? - B. Blanc, B. Maaroui - Dec 2005.



```

int createValue( int i, int j, int tipoInput){
    int value = 0;
    value = rand()% 10 + 1;
    return value;
}
int main(int argc, char** argv) {

    // read from run string the parameter that specifies the number of packets to do
    int numberOfPackets = atoi ( argv[1] );
    ///> Create Stream of telemetry
    Output* out;
    OutputPacketStream ops;

    try{
        ///> Create the structure of the telemetry in memory
        ops.setFileNameConfig("conf/STRUCT.stream");
        ops.createStreamStructure();

        char** param = (char**) new char*[3]; ///> array declaration
        out = (Output*) new OutputFile(ops.isBigEndian());
        param[0] = "output/telemetry_generated.raw"; // define where will be saved the output
        param[1] = 0;

        ///> open output
        out->open(param);

        ///> connect the output
        ops.setOutput(out);

        ///< file to save value for telemetry packet
        ofstream telemetryValue;
        telemetryValue.open ("telemetryValue.txt");

        ///< Writing n packet in telemetry
        for (int sequence_counter = 1; sequence_counter <= numberOfPackets;
sequence_counter++){

            ///> creation single packet
            Packet* p = ops.getPacketType(1); //first structure of packet defined

            ///> update sequence counter in stream header

```



```

p->header->setFieldValue(5,sequence_counter);

// setting the value for data field header
p->dataField->dataFieldHeader->setFieldValue_3_14(1, 5000); //event time tag
p->dataField->dataFieldHeader->setFieldValue(3, 10);          //PDM Address
p->dataField->dataFieldHeader->setFieldValue(4, 5); // pixel ID

///<> writing source in data field
int value = 0;
for (int i = 0; i < 100; i++){
    value = createValue(sequence_counter, i, tipoInput);

    // writing on Check File
    telemetryValue << " PACKET - " << sequence_counter << " BLOCK - " << i
<< " Value - " << value << "\n" ;

    p->dataField->sourceDataField->setFieldValue(i , value );
}

// send packet
ops.writePacket(p); ///<> writing packet
}

///<> close random value file
telemetryValue.close();

///<> close the output
out->close();

}catch (PacketException* e){
    cout << e->getError() << endl;///<> if there is an exception then view corresponding
error
}
return 0;
}

```