



ProcessorLib Tutorial: how to implement a simple processor task using the ProcessorLib

Internal Report IASF Bologna n 613/2012

Prepared by: Name: V. Conforti Signature: Date: 10/09/2012
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	10 September 2012	First version

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFB0-TN-003	Issue:	1.0	DATE	10-SEP-12	Page:	1

TABLE OF CONTENTS

1. INTRODUCTION	2
2. DEVELOPMENT ENVIRONMENT.....	3
3. PROCESSOR ARCHITECTURE	4
3.1 TELEMETRY STRUCTURE CONFIGURATION FILES	5
3.2 PROCESSOR INFORMATION.....	5
3.3 FITS STRUCTURE DEFINITION.....	6
4. PROCESSOR IMPLEMENTATION	7
4.1 THE SKELETON	7
4.2 THE CODE	7
4.2.1 <i>Tutorial_Processor.h</i>	7
4.2.2 <i>Tutorial_Processor.cpp</i>	8
4.2.2.1 The constructor:.....	8
4.2.2.2 Load Configuration	9
4.2.2.3 Create Memory Structure	9
4.2.2.4 Set Value	9
4.2.2.5 Other methods.....	10
4.2.3 <i>Tutorial_FITS.h</i>	10
4.2.4 <i>Tutorial_FITS.cpp</i>	11
4.2.4.1 Constructor	11
4.2.4.2 Distructor	11
4.2.4.3 Init.....	11
4.2.4.4 Close	12
4.2.4.5 Write Data.....	12
5. COMPILE AND EXECUTE A PROCESSOR.....	15
6. EXECUTION OPTION	16
REFERENCE DOCUMENTS	17
REFERENCE SITES	17

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page:	2

1. Introduction

Data Management in Astrophysics needs particularly attention in terms of time and space. The amount of data to be acquired from the instrument (both space-borne and ground-base) can be very high and the data acquisition system must be fast as much as the bit rate.

In this document, the raw data collected from the instruments are named *telemetry* or L0 data, and are in binary format. Each experiment adopts for the telemetry a specific layout, which usually foresees a header followed by the instrument data block. The header contains the information that identifies the structure of instrument data block.

The binary format is very good for data transmission, but it is not suitable for human being. A standard scientific format data (not only for image) very used in Astronomy is the FITS (Flexible Image Transport System). Usually the instrument raw data have to be transformed into FITS format. In this document they are named L1.

This tutorial shows how to implement the task (*processor*) that performs the transformation of the data from L0 to L1.

This transformation is done using specific libraries: PacketLib, ProcessorLib and CFITSIO:

- CFITSIO is a C++ library, developed from NASA, that help to create and write a file fits.
- PacketLib is an IASFBO library that allows a good management of L0 data.
- ProcessorLib offers all method to transform L0 data in L1 and uses CFITSIO and PacketLib.

The input to a processor may be a telemetry file or a socket connection or a DISCOS connection (ready to receive telemetry packets).

The processor implementation is obtained by overriding some ProcessorLib methods, as explained in this guide.

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page:	3

2. Development Environment

We assume the following development environment:

- Operating System: Linux Cent OS 64 bit;
- Editor: Eclipse CDT (C/C++ Development Tool);
- Libraries: PacketLib, ProcessorLib, CFITSIO;
- Repository/Versioning: GIT / SVN

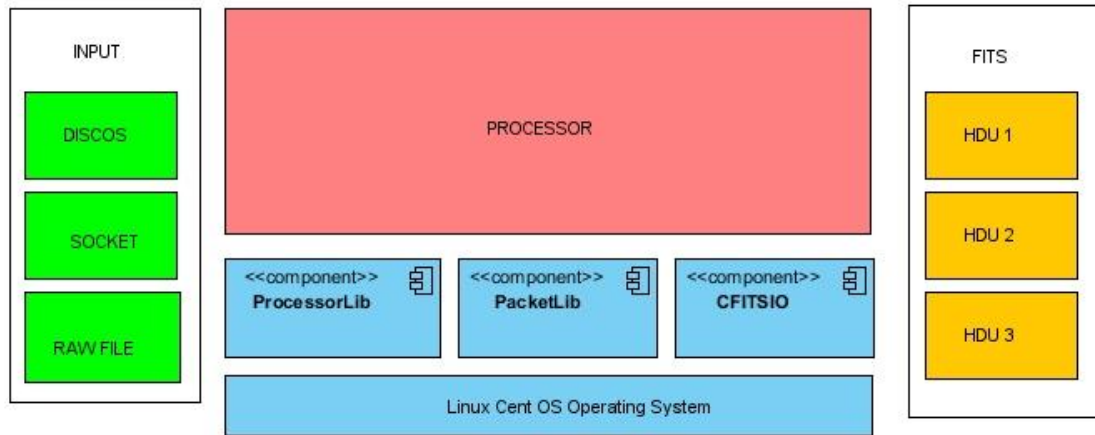
Environment variables:

- TBD

A virtual machine has been set-up with the above environment and can be imported using *Oracle Virtual Box*.

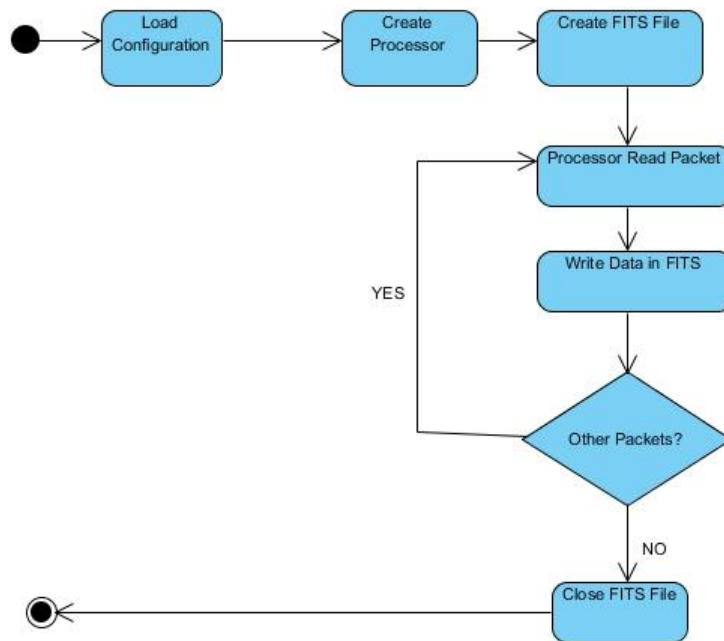
CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page:	4


3. Processor Architecture



In the left part of above figure there are accepted kinds of input. On the right there is the output which is a FITS file with one or more HDU (Header Data Units). The PROCESSOR uses the PacketLib to read and unpack the L0 data packets, and the CFITSIO library to write the FITS output file.

The lifecycle of the processor is reported in the following sequence diagram:



<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page:	5

The program starts loading the processor configuration, and allocates the memory needed to speedily manage the data packets and create the FITS file. For each input packet the processor reads the packet header, the data field header, fields and blocks (if present), then writes them in the FITS file. When packets are finished the FITS file is closed and program ends.

It is necessary to write one processor for any kind of telemetry packet foreseen for the given experiment.

Each processor must be carefully designed for a particular kind of packet, similarly must be defined the structure of the FITS file.

For this reason a processor needs:

- telemetry structure configuration files (which describe the telemetry),
- a processor information file (with description of processor),
- and the definition of the FITS file.

All this files, by convention, must be inserted in a *conf* (configuration) folder inside the project folder.

3.1 Telemetry Structure configuration files

The telemetry is defined by 3 kinds of files:

1. One .header file;
2. One .stream file;
3. One or more .packet files;

The *.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.

If there is only one kind of packet, there will be only one .packet file, otherwise it must be created one .packet file for each kind of packet.

In the header file there are some fields which describe the telemetry packets such as the version number, the APID (Application Process ID), the packet length and the source sequence counter (counter of packets).

The .packet file describes all fields of each packet. The packet has always two sections: the data field header, and the source data field. Some fields are required to describe the type of packet and how the information is contained in the packet.

Detailed information on the packet structure design is given in [RD1] document.


3.2 Processor Information

The processor information must be inserted in a Tutorial_Processor file.

Some of information are mandatory and common to all the processors, others are specific to the given processor.

E.g. it contains the reference to .stream file that describes the telemetry (mandatory) and other information to be included in the HDU1 of the FITS file.

It can be used to pass to the processor specific information to pass to processor.

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page:	6

Further information are given in [RD4].

3.3 FITS structure Definition

The FITS structure must be defined in your processor code. The FITS file could have one or more HDU (Header Data Units). The first HDU is called "*Primary HDU*" and will be created automatically with the information inserted in .processor file.


The others HDU are called *addition HDU* and are FITS extensions. The standard extension types are:

- Image Extension: n-dimensional array of pixels;
- ASCII Table Extension: store tabular information in ASCII format;
- BINARY Table Extension: store tabular information in binary format.

You must define the number of table column, and for each column you must define:

- The column name;
- The data type;
- The measurement unit;

Each of this definition must be inserted in array structure as required by the CFITSIO library.

<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page:	7

4. Processor Implementation

4.1 The Skeleton

A processor requires many files. At moment no tool is available to help the developer to prepare these files.

Waiting for these tool, a complete set of these files are given in Annex as templates to be manually customized for the specific case.

The skeleton of the application is composed as follows:

- conf/
 - Tutorial.packets (one or more) ;
 - Tutorial .stream (one);
 - Tutorial .header (one);
 - Tutorial .processor (one or more)
- include/
 - Tutorial_FITS.h;
 - Tutorial_Processor.h;
- src/
 - Tutorial_FITS.cpp;
 - Tutorial_Processor.cpp;
 - main.cpp.

In this example is used Tutorial as name of project. Same structure can be used for any project changing the name of project.

It's recommended the use of a Makefile (you can use and edit an existing processor makefile) and the Doxyfile in order to create automatically the code documentation.

4.2 The Code

The following sub-paragraphs explains the essential part of the code of each file composing the processor application.

4.2.1 Tutorial_Processor.h

The processor can take data input from a file or from a socket, so may be that the packets arrive asynchronously. Then the process and the insert step in FITS file is done packet by packet.


The processor must allocate data structure well defined in memory in order to accept one packet. Each packet uses the same memory so it's sufficient allocate the memory for the biggest packet.

First, it must be created a *struct* that define the data structure needed.

For example:

```
typedef struct
{
    float header[5];
    int fields[10];
} PACKET;
```

In this case, the structure can accept packets which have 5 fields of float values and 10 fields of integer values.

<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page:	8

It possible to create complex structures as required by the specific telemetry packet.

Then, it must be defined the specific processor class. One simple example is:

```
class Tutorial_Processor : public Processor
{
public:
    Tutorial_Processor();
    bool loadConfiguration(int argc, char** argv) throw(PacketException*);
    virtual char* processorVersion() { return "1.0.0 (not official version)"; };
    virtual char* processorAuthor() { return "Vito Conforti"; };
    char* processorDescriptor() { return "IASF Bologna Telemetry-Simulated-Processor"; };
    char* processorID() { return "Learning_ImplementationTutorial_Processor"; };

protected:
    virtual void createMemoryStructure();
    virtual bool setValue();
    virtual char** initCharValueForOutput_init();
    virtual int* initIntValueForOutput_init();
    virtual char** initCharValueForOutput_close();
    virtual int* initIntValueForOutput_close();

private:
    int apid;
};
```

This code is required to implement the Processor Class. All the above defined methods must be implemented. Other methods or other private member can be added as needed.


4.2.2 Tutorial_Processor.cpp

In this file you must include Tutorial_FITS.h and implement all methods declared in C++ .h file:

4.2.2.1 The constructor:

```
Tutorial_Processor::_Processor()
{
    addOutputFileProcessor(new Tutorial_FITS());
    apid = 1292;
}
```

This code is required to create an instance of the Tutorial_FITS object.

<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling					
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page:	9

4.2.2.2 Load Configuration

The *loadConfiguration* method is called when program starts. It sets the configuration using the run string parameters. In normal situation, this method is implemented using the same method of the Processor super class:

```
bool Tutorial_Processor::loadConfiguration(int argc, char** argv) throw(PacketException*)
{
    bool ret = Processor::loadConfiguration(argc, argv);
    return ret;
}
```

4.2.2.3 Create Memory Structure

The *createMemoryStructure* method allocates the memory required to host a single packet of maximum size:

```
void Tutorial_Processor::createMemoryStructure(){
    PACKET* packet = (PACKET*) new PACKET;
    arrayDataOutput = (void*) packet;
}
```

The link between the processor object and fits object is the *arrayDataOutput*. It's a variable of the ProcessorLib and it must be filled with the pointer to the memory just allocated.

4.2.2.4 Set Value

The *setValue* method is called by the *ProcessorLib* for each packet.

The packet data are taken from the ProcessorLib protected variable named "p".

The data are stored in the memory structure allocated above.


Each packet field is read using the PacketLib.

Here is reported a simple example:

```
bool Tutorial_Processor::setValue(){
    int nrow = 1;
    PACKET* packet = (PACKET*) arrayDataOutput;
    apid=(p->header->getFields(3))->value; // take apid of current packet

    packet->header[0] = (int) p->dataField->dataFieldHeader->getFieldValue(2);
    packet->header[1] = (float) p->dataField->dataFieldHeader->getFieldValue_5_1(3);
    packet->header[2] = (float) p->dataField->dataFieldHeader->getFieldValue_5_1(5);
    packet->header[3] = (float) p->dataField->dataFieldHeader->getFieldValue_5_1(7);
    ...

    nrowsFITS = nrow;
    return true;
}
```

<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling				
	Code: CIWS-IASFB0-TN-003	Issue:	1.0	DATE	10-SEP-12	Page: 10

It is noted that all fields of the structure must be filled respecting the data type of the structure. To this purpose it can be used the cast operator. The *nrowsFITS* variable must be set to the number of row that needed in the FITS file in order to write the data contained in the packet structure.

This is a simple example which needs only one row. Packet with complex layouts (e.g. data fields with more than one block) could need more than one row (e.g. one row for each block).

4.2.2.5 Other methods

In a simple application the following methods can be left empty (return NULL) :

```
initCharValueForOutput_init
initIntValueForOutput_init
initCharValueForOutput_close
initIntValueForOutput_close
```

The above methods pass specific data (integer or character array) to the *init* or *close* method of the FITS object.

4.2.3 Tutorial_FITS.h


The FITS class is responsible for writing the FITS file. The *Tutorial_FITS.h* file contains the declarations of the methods to be overridden and the members of the *FITSBinaryTable* (super class).

A simple example is:

```
class Tutorial_FITS : public FITSBinaryTable
{
public:
    Tutorial_FITS(); /// constructor
    ~ Tutorial_FITS(); /// distructor

    virtual bool init(char**c, int*i, char* filenameconfig = 0) throw (PacketException*);
    virtual bool close(char**c, int* i);
    virtual bool writeData(void* data, dword nrows);

private:
    // fits packet structure
    char **ttype;
    char **tform;
    char **tunit;
    int tfields;
};
```

<h1>CIWS</h1>		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling				
	Code: CIWS-IASFB0-TN-003	Issue:	1.0	DATE	10-SEP-12	Page: 11

The *init* method is called when the program starts. It creates the FITS file and the FITS tables. The *close* method closes the FITS file. The *writeData* method is called for each packet in order to write the data in the FITS file. The private member are used as pointers in order to use them in all methods.

4.2.4 Tutorial_FITS.cpp

In this file there is the implementation of all method of the FITS class:

4.2.4.1 Constructor

```
Tutorial_FITS:: Tutorial_FITS(){

    tfields=2; // number of column

    ttype = new char* [tfields+1];
    ttype[0] = "first field";
    ttype[1] = "second field";

    tform = new char* [tfields+1];
    tform[0] = "lI"; // signed 16 bit integer
    tform[1] = "lE"; // 32 bit floating point

    tunit = new char* [tfields+1];
    tunit[0] = "TeV";
    tunit[1] = "TeV";

}
```

The above example is for a binary table with 2 columns. For each column are defined the ttype (name of field) , tform (type of field) and tunit (unit of measure) values.

The possible tform values are reported in the CFITSIO documentation.

4.2.4.2 Distructor

It simply frees all the allocated memory:

```
Tutorial_FITS::~ Tutorial_FITS(){

    delete[] tform;

    delete[] ttype;

    delete[] tunit;

}
```

4.2.4.3 Init

The *init* method executes preliminary operation in order to write the FITS file:

```
bool Tutorial_FITS::init(char**c, int*i, char* filenameconfig) throw (PacketException){
```



```

int status = 0;
long nrows = 0; /* total number of rows to be updated - initial number of row */

/* extension name - is the name for the table */
char extname[] = "_Binary_packet";

// initialization of row
currentrow = 1;

char* filename = getFileName();

/* create new FITS file */
if (fits_create_file(fp_ptr, filename, &status))
    perror( status ); /* call perror if error occurs */

/* create HDU - 2 -> table of data field of packets*/
if ( fits_create_tbl(*fp_ptr, BINARY_TBL, nrows, tfields, ttype, tform,
    tunit, extname, &status) )
    perror( status );
return status;
}

```

In this simple example, there are some initialization for the name of table and the number of first row to be written.

The methods to create FITS file and table are in the CFITSIO library. Details of single function can be read in CFITSIO documentation. It should be careful that many parameter in the CFITSIO function accept only array data format.

4.2.4.4 Close

This function simply closes the FITS file:

```

if ( fits_close_file(*fp_ptr, &status) )
    perror( status );

```

4.2.4.5 Write Data

The `writeData` method reads data of the current packet stored in the shared memory, then writes them in FITS file:

```

bool Tutorial_FITS::writeData(void* data, dword nrows){
    int status = 0;
    int arrayColumn[12];

```



```

float arrayColumnf[12];

PACKET* currentPacket = (PACKET*) data;

int nblock = nrows; /* number of rows*/
long firstelem = 1 /* first element in row (ignored in ASCII tables) */
long firstrow = 1; /* first row to write*/
int nelements = 2; /* number of elements*/
long numrighe = 1; /* number of rows*/

// read data of packet from shared memory
arrayColumn[0] = (int) currentPacket->header[0];
arrayColumnf[1] = (float) currentPacket->header[1];

// set current HDU to write
if ( fits_movabs_hdu(*fptr, 2, 0, &status) )
    perror( status );

//write first column data
if(fits_write_col(*fptr,TSHORT,1,currentrow,firstelem,(long)numrighe,&arrayColumn[0],
&status))
    perror(status);

// write second column data
if ( fits_write_col(*fptr, TFLOAT, 2 , currentrow, firstelem,(long) numrighe ,
&arrayColumnf[1], &status))
    perror(status);
}

/* update the currentrow index for the next time this routine will be called */
currentrow = currentrow + 1;

return status;
}

```

In this simple example is shown how the data of shared memory are written in FITS file. It is good practice to select the current HDU before writing. The function that selectd the HDU is *fits_movabs_hdu* [RS3].

The method to write in fits file is *fits_write_col*. This method write a column in the table so the value to be written must be in array format. In this example it is passed an array of size 1. In order to write

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling				
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page: 14

one row it must call the *fits_write_col* for each field of the table. The details of this function are written in [RS1].

The variable *currentrow* is a private member of Tutorial_FITS class. It allow to keep tracks of the current row to be written.

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling				
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page: 15

5. Compile and execute a processor

The processor created must be compiled with 32 bit for compatibility reason with other libraries.

```
make CFLAGS="-m32"
```

It can add this option in Makefile.

Is recommended the use of an existing processor Makefile because already has the link to required library.


The use of an existing Makefile can required some modifies. For example the name of executable file, enabling/disabling of DISCOS. If it does not use DISCOS it must be disabled otherwise will be a an error in compile time.

Finally must be set some environment variable. The GTB group of IASF - Bologna has a file and instruction to import a profile file with all correct configurations.

The command to execute a processor is

```
./name_exe nameTutorial_Processor
```

The `name_exe` is the name of executable file, the `nameTutorial_Processor` is the name of processor. It can be added other parameter reported in the next chapter.

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling				
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page: 16

6. Execution Option

When the user lunch the program can set the following optional parameters:

```
-h --help          Display this usage information.
-o --msgout        Shows the console messages during acquisition.
```

The following parameters override the same parameters contained in the .processor configuration file:

```
-m --model          Instrument model under test (e.g. SEM, PFM).
-t --testlevel      Test level of the acquisition.
-T --testequipment  Test equipment connected with SC.
-c --campaign       Campaign of the acquisition (e.g. cnr, lab, cern).
-a --acquisition    Acquisition type (0 PLAYBACK, 3 HBR_SCOE).
-p --period         Period of the campaign.
-s --source         Filename for file as input, port for socket as input,channel for DISCoS system.
-d --destination    Only for single input: destination directory of the fits file (with end /)
-f --filenameever   Specify the file name version. If 1, add date and time to the file name.
-r --crcdisable     Disable the verification of the CRC.
-C --calibrationlogEnable calibration log flag.
-l --close          Specify the max number of packets for each file.
```

CIWS		Customizable Instrument Workstation Software (CIWS) for telescope-independent L0/L1 data handling				
	Code: CIWS-IASFBO-TN-003	Issue:	1.0	DATE	10-SEP-12	Page: 17

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] ProcessorLib 1.2.2 Detailed Design Report - A. Bulgarelli, F. Gianotti, M. Trifoglio - May 2003.
- RD [4] ProcessorLib 1.2.2 Programmers Guide - A. Bulgarelli, F. Gianotti, M. Trifoglio - May 2003.
- RD [5] CFITSIO User's reference Guide - An Interface to FITS Format Files for C Programmers - HEASARC - April 2012.
- RD [6] AGILE Science Console Design Concept - A. Bulgarelli, F. Gianotti, M. Trifoglio - November 2003.
- RD [7] AGILE Science Console and Preprocessing Software Requirement Document - A. Bulgarelli, F. Gianotti, M. Trifoglio - November 2003.

Reference sites

- RS [1] CFITSIO: <http://heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html>
- RS [2] FITS : <http://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html>
- RS [3] <http://www.aip.de/~weber/doc/fitsio/cfitsiohtml/node63.html>