| | |
|---|---|
| **DOCUMENT TYPE:** | **TECHNICAL NOTE** |
| **TITLE:** | **PROCESSORLIB 1.2.2 PROGRAMMERS GUIDE** |
| **DOCUMENT Ref. No.:** | AGILE-ITE-SD-001 **N° OF PAGES:** i-v, 26 |
| | IASF section of Bologna Report 349/02 |
| **ISSUE No.:** | 01 **DATE:** May 2003 |
| **PREPARED BY:** | A. BULGARELLI<br>F. GIANOTTI<br>M. TRIFOGLIO |
| **CHECKED BY:** | M. TRIFOGLIO |
| **SUBSYSTEM MANAGER:** | M. TRIFOGLIO |

**APPROVED BY:**

| | | |
|---|---|---|
| **SUBSYSTEM LEADER:** | G. DI COCCO | **DATE:** |
| **PROJECT LEADER:** | M. TAVANI | **DATE:** |
| **PAYLOAD MANAGER:** | A. ZAMBRA | **DATE:** |
| **PAPM:** | A. BERNABEO | **DATE:** |
| **CONFIGURATION:** | | **DATE:** |

## DISTRIBUTION LIST

| POS. | NAME | DEPT. | N° OF COPIES | FULL COPY |
|---|---|---|---|---|
| 1 | M. Tavani | IASF MI | 1 | 1 |
| 2 | A. Zambra | IASF MI | 1 | 1 |
| 3 | G. Di Cocco | IASF BO | 1 | 1 |
| 4 | C. Labanti | IASF BO | 1 | 1 |
| 5 | E. Celesti | IASF BO | 1 | 1 |
| 6 | M. Trifoglio | IASF BO | 1 | 1 |
| 7 | F. Gianotti | IASF BO | 1 | 1 |
| 8 | A. Bulgarelli | IASF BO | 1 | 1 |
| 9 | M. Feroci | IASF RM | 1 | 1 |
| 10 | F. Lazzarotto | IASF RM | 1 | 1 |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

## CHANGE RECORD

| ISSUE | DATE | PAGE | DESCRIPTION OF CHANGES | RELEASE |
|---|---|---|---|---|
| 0A | 16/09/2002 | All | First issue of the document | 1 |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

**SUMMARY**

## INTRODUCTION

The diagrams and the terms presented in this document are conformed with the UML-OMG 1.4 standard [2]. The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.
*The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components. The UML represents the culmination of best practices in practical object-oriented modeling.*
For the description of the architecture are used the *implementation* diagrams. This diagrams show aspects of physical implementation, including the structure of components and the run-time deployment system. They come in two forms:

- *component diagrams* show the structure of components, including the classifiers that specify them and the artifacts that implement them;
- *deployment diagrams* show the structure of the nodes on which the components are deployed.

For a logical overview of the software architecture are used the package diagram (a package is a grouping of element as component, code, etc.) and the class diagrams. For a description of the sequence of operations the activity diagrams are used.

## ACRONYMS

| | |
|---|---|
| AD | Applicable Document |
| AIV | Assembly, Integration and Verification |
| CCOE | Central Check-Out Equipment |
| CERN | European Organization for Nuclear Research |
| CsI | Ceasium Iodide |
| DAQ | Data Acquisition |
| EGSE | Electrical Ground Support Equipment |
| FEE | Front-End Electronics |
| FITS | Flexible Image Transport System |
| GPS | Global Positioning System |
| GRID | Gamma Ray Imaging Detector |
| GSE | Ground Support Equipment |
| HK | HouseKeeping |
| LAN | Local Area Network |
| MCAL | Mini-Calorimeter |
| MGSE | Mechanical Ground Support Equipment |
| ML | Milions |
| NFS | Network File System |
| PD | Photo Diode |
| PDHU | Payload Data Handling Unit |
| P/L | Payload |
| RD | Reference Document |
| SC | Science Console |
| SCOE | Specific Check-Out Equipment |
| ST | Silicon Tracker |
| TC | Telecommand |
| TE | Test Equipment |
| TM | Telemetry |

## REFERENCE DOCUMENTS

[1] BSSC, “ESA software engineering standard”, ESA PSS-05-0 Issue 2, February 1991.
[2] “OMG Unified Modeling Language Specification”, Version 1.4, September 2001.
[3] ESA Packet telemetry standard
[4] AGILE-ITE-SR-03, “User software requirements for the host computer of the agile minicalorimeter CAL-DFE test equipment”, Issue 02/A, March 2002
[5] ProcessorLib Detailed Design Report
[6] INTERFACE CONTROL DOCUMENT OF THE AGILE EGSE SOFTWARE

# 1. DESIGN MODEL

## 1.1 OVERVIEW

In this section are represented the UML class diagram for the ProcessorLib. For more details see [5].
The next picture shows the main class diagram of the library. In this diagram the main classes are represented:

- Processor: this class represent a single processor with all its functionality, as described in the next chapters
- Provider: a processor can receive the telemetry flow of data coming from a single provider. The selection of the provider can be performed by means of a configuration files (.processor)
- Monitor: this class represent an interface useful for the communication with a monitor. A monitor could be a software that shows some information about the processor
- OutputFileProcess is the class that represent the output format of the processor. The output could be a FITS file.

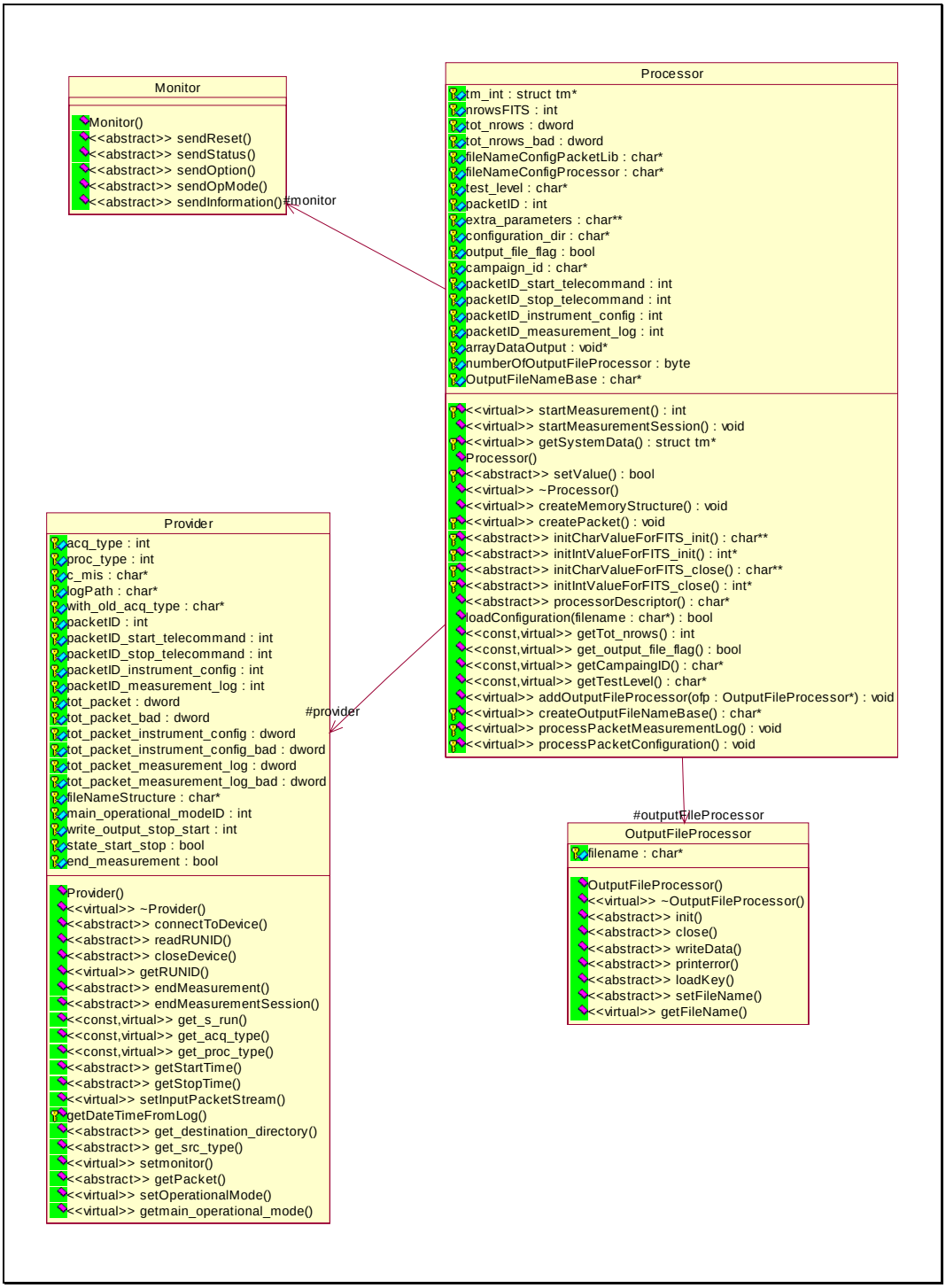


**Figure 1**: overview of ProcessorLib

## 1.2 MONITOR HIERACHY

Two types of Monitor are provided; the MonitorDISCOS that interface the processor with the DISCOS monitor, and a dummy monitor with no output. This hierarchy could be extended with other monitors.

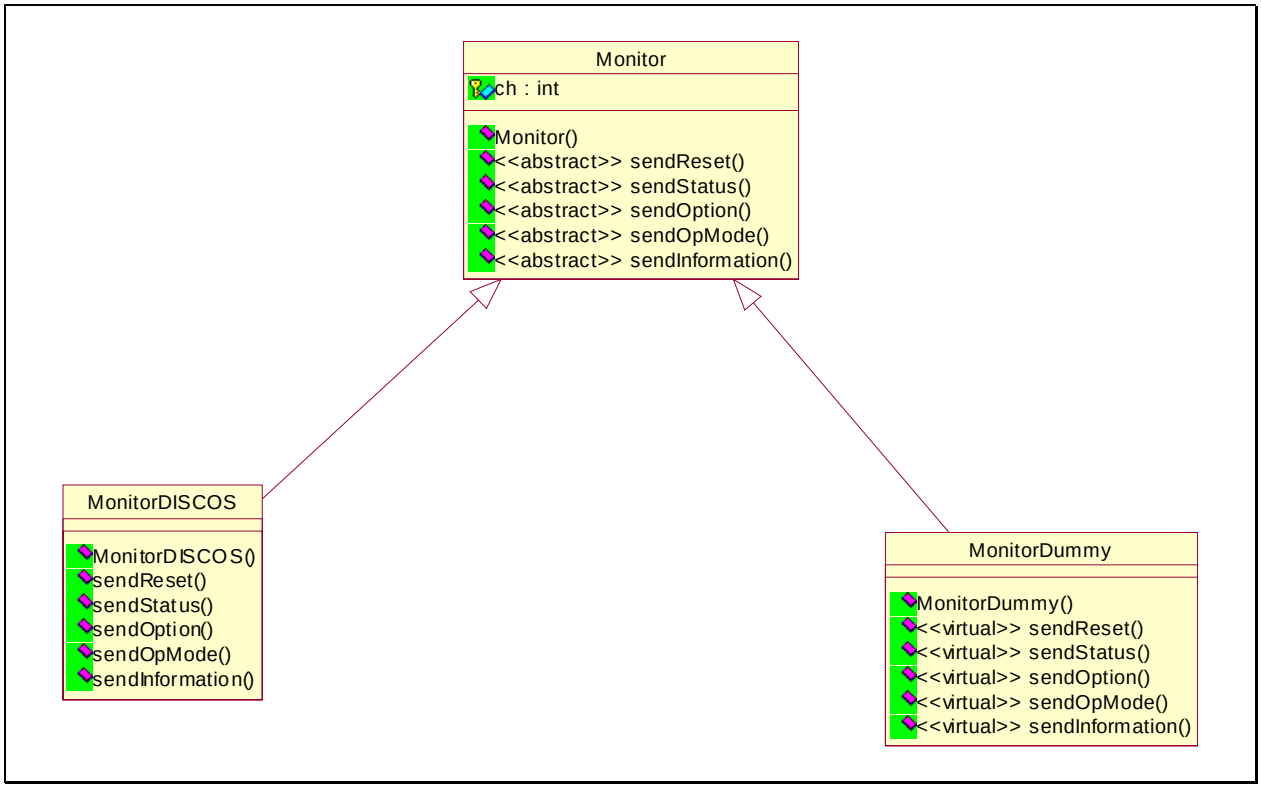


**Figure 2**: Monitor Hierarchy

## 1.3 PROVIDER HIERACHY

Two type of provider are present; the ProviderDISCOS that use as input the DISCOS shared memory, and the ProviderSingleInput. The last could be a single input file containing the telemetry or a socket.

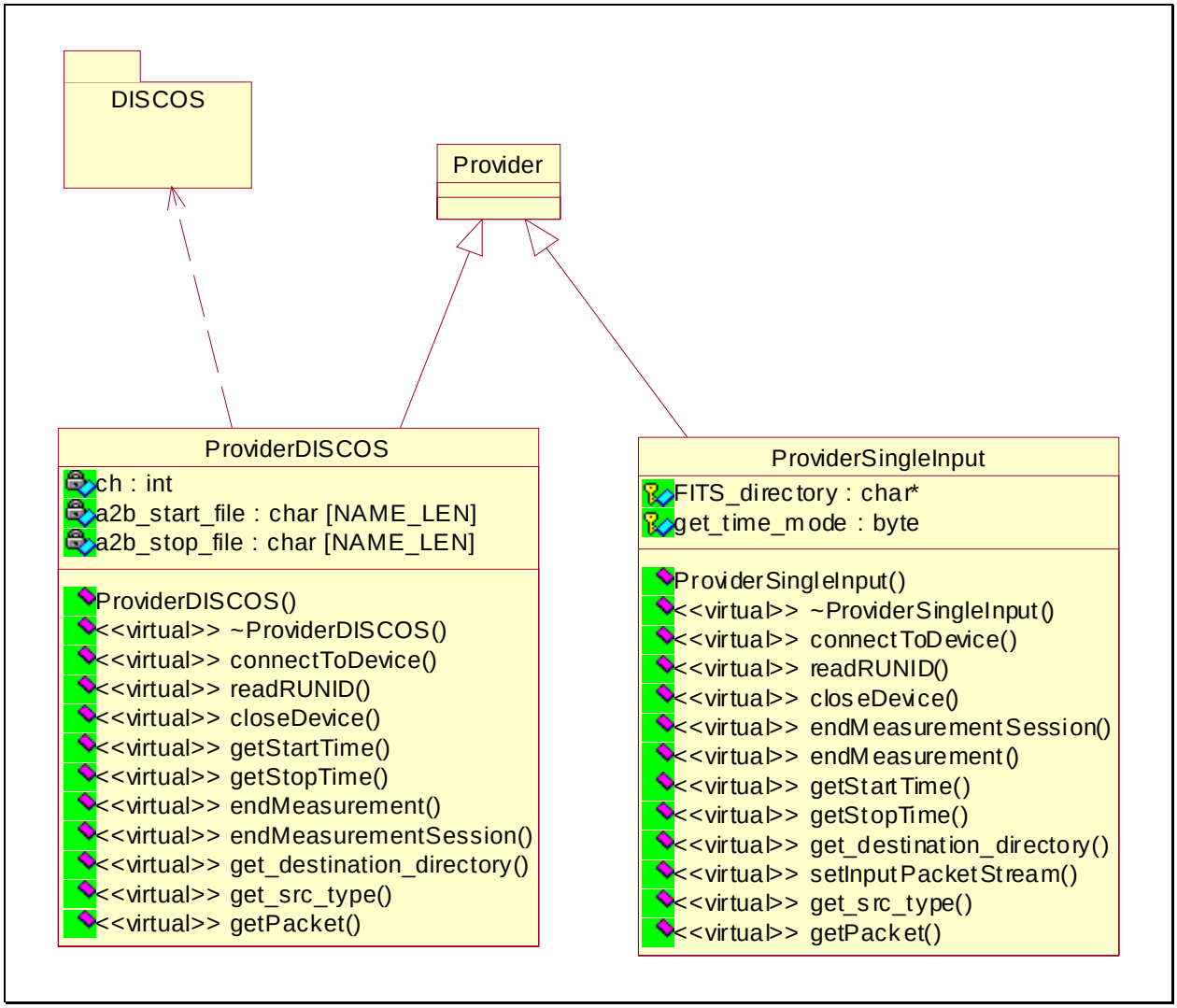


**Figure 3**: Provider Hierarchy

## 1.4 OUTPUTFILEPROCESSOR HIERARCHY

At the moment only the FITS output type are present, but this hierarchy could be extended with new type of outputs.

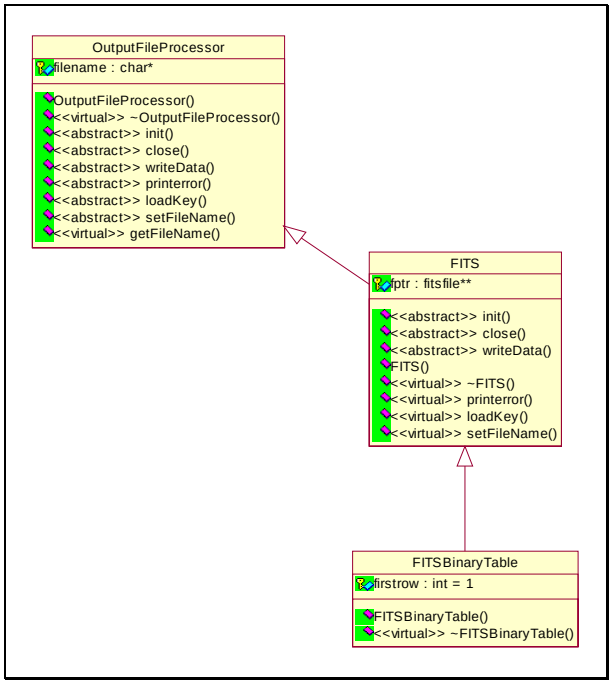


**Figure 4**: OutpuFileProcessor Hierarchy

# 2. MEASUREMENT AND MEASUREMENT SESSION

## 2.1 MEASUREMENT SESSION

The main purpose of the processor software is to obtain and process the data produced by the instruments under test.
Each measure starts with a start event (typically a start command) and end with a stop event (typically a stop command) as determined by the configuration file (.processor). A single measure is univocally identified with

- Runid
- Campaign
- Chain of acquisition (more generically, input type)

A set of measurements represent a measurement session. A measurement session does not need a unique identifier, but corresponds with a unique campaign and test level.

In the next picture is showed the entire steps performed for a measurement session. The yellow activities are performed by library; the red activities are performed by the particular processor written by user of the library.

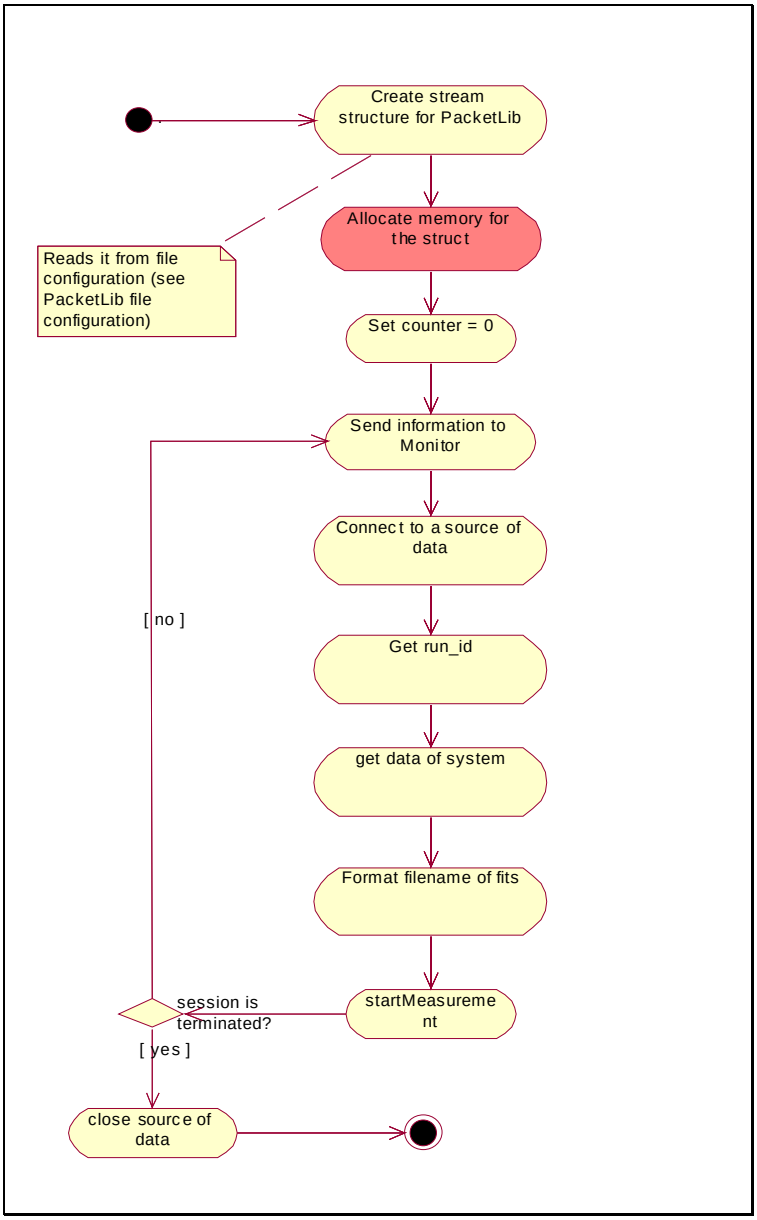


**Figure 5**: Measurement session

## 2.2 MEASUREMENT

In the next picture is showed the entire steps performed during a measurement. The yellow activities are performed by library; the red activities are performed by the particular processor written by user of the library.

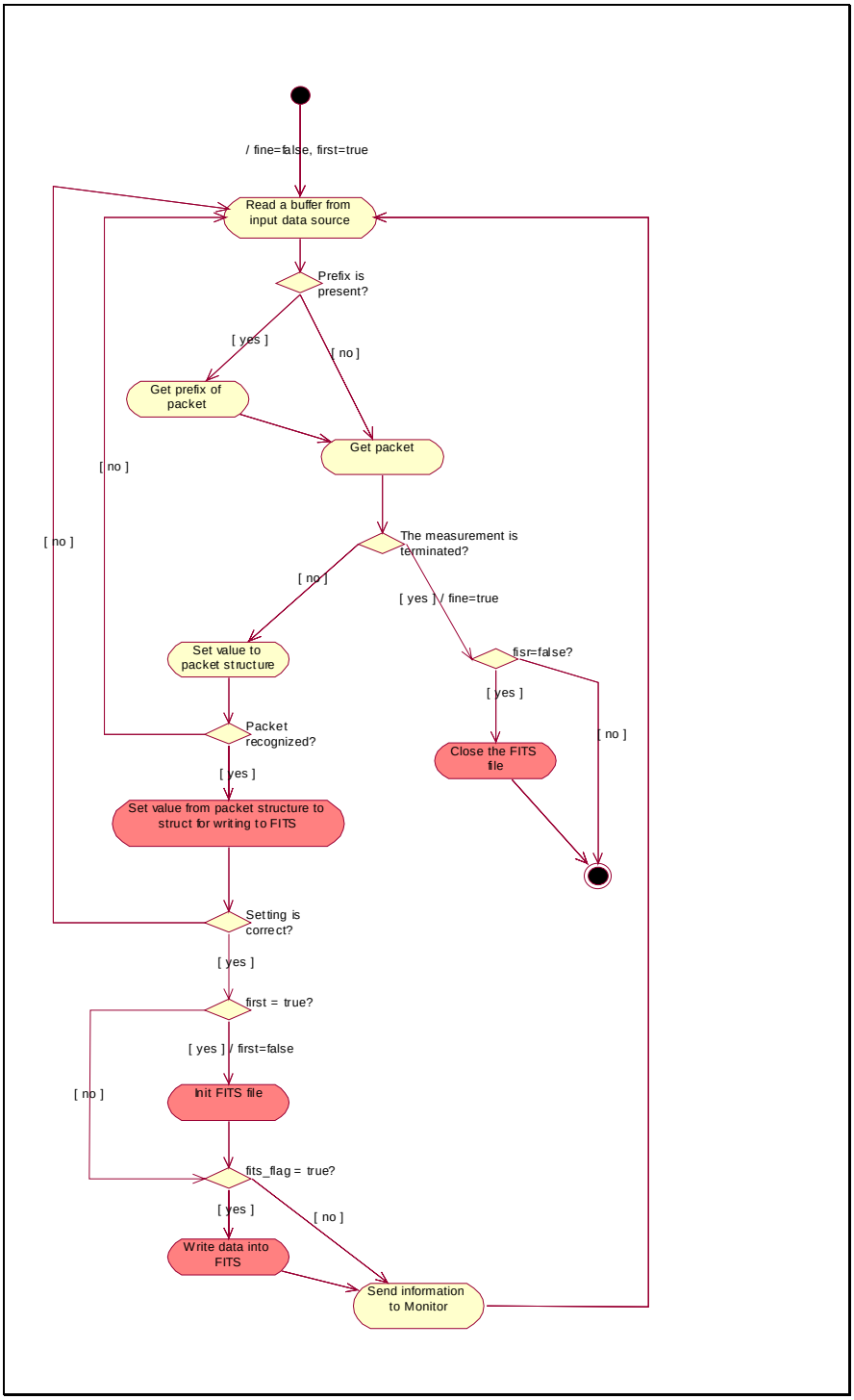


**Figure 6**: Measurement

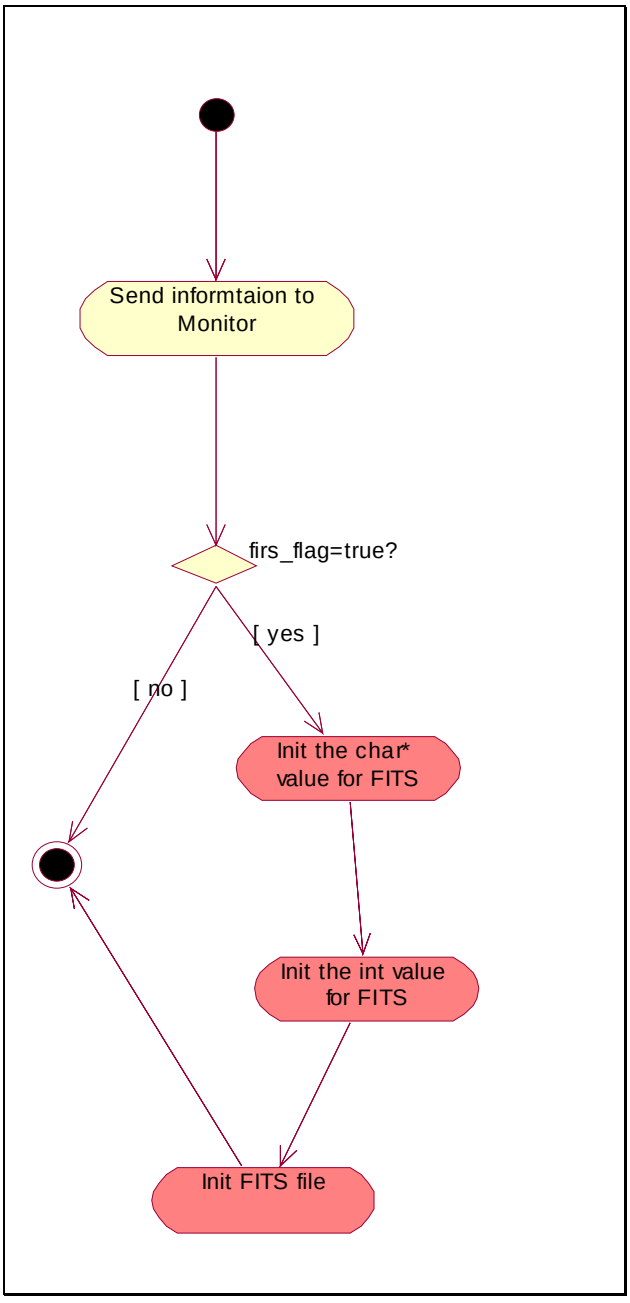


**Figure 7**: Init FITS file

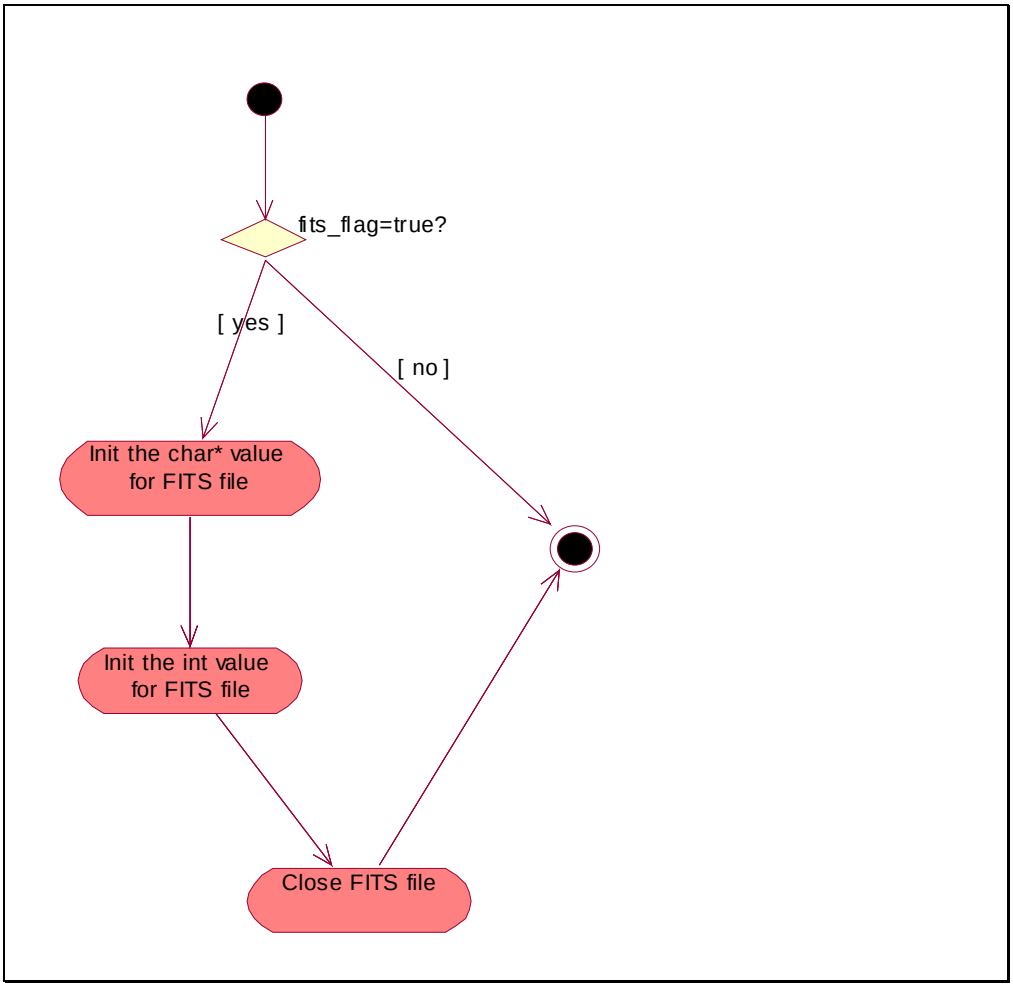


**Figure 8**: Close FITS file

## 3. HOW TO WORK PROCESSORLIB

In the Figure 9 are described the main functionality of the library.
The class showed in the last chapter implement, on the whole, both the general purpose functions included in all the processors (part A, in the next diagram of) and all the elements needed to customize them (part B), given the particular instrument and data flow to be processed.
A single processor can work in this operational mode:

1. **DISCoS mode**: the processor is attached with DISCoS software and gets the data from it.. DISCoS mode has two sub-mode:
   1.1. **DISCoS online mode**: the data source for DISCoS are obtained from an instrument or a software generator.
   1.2. **DISCoS playback mode**: DISCoS is used for providing data to processor, but this data are obtained from the raw file archived by Archiver.
2. **test mode**: a processor can run without DISCoS. This mode is useful during the development of a processor or when DISCoS is not finished (for parallel development within a software team).

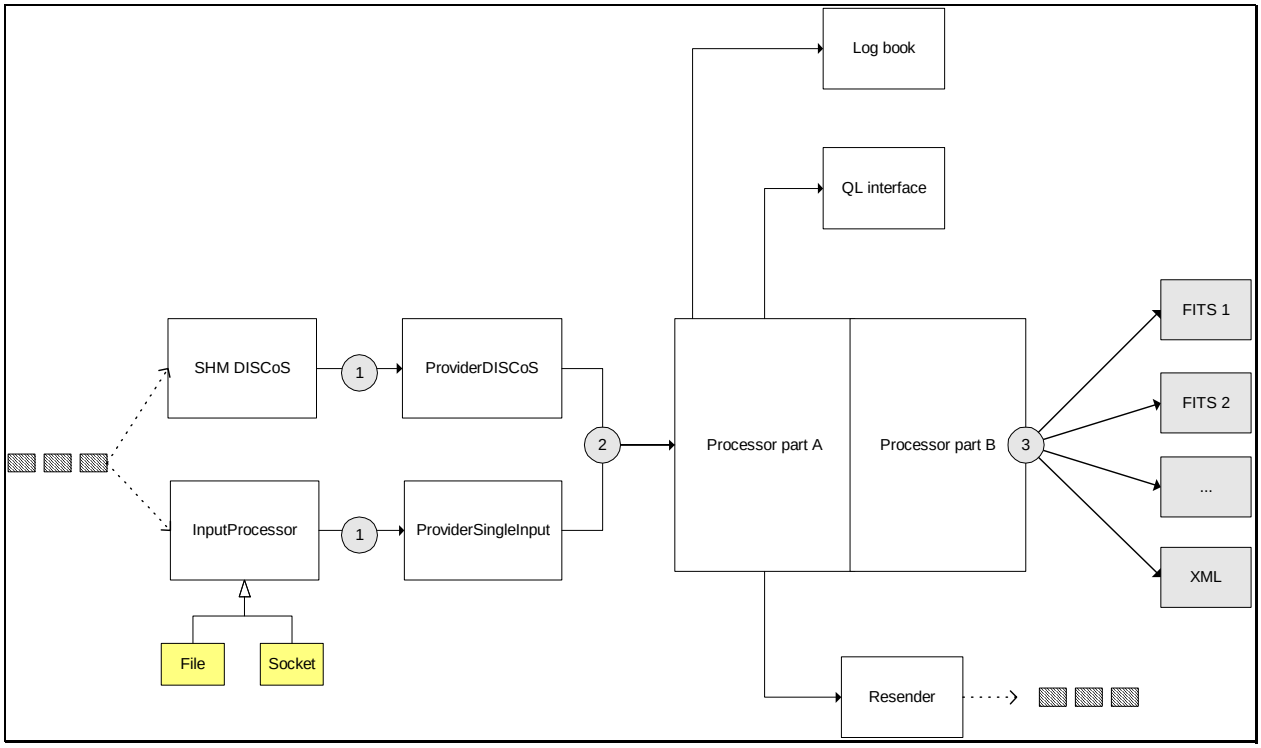


**Figure 9**: ProcessorLib general schema

## 3.1 INPUT INTERFACE

The input of the system is an input of bytes. This flow is managed by a set of **input interfaces** able to:

1) Connect to SHM DISCoS (in DISCoS mode)
2) Read the byte strema in input from File or Socket (in test mode)

The main purpose of these interfaces is to get from the byte input stream the necessary number of bytes to recognize the TM packet. The reading operation are performed in the following way:

- About the DISCoS mode, a buffer of fixed dimension are read from the DISCoS shared memory
- About the test mode, the TM packet header (of fixed dimension) is read. From this header the dimension of the source packet is read and, after this, a number of byte of the same dimension of the source packet is read.

The output of these *input interfaces* (see number 1 in the diagram) is a TM packet (see [6]).

## 3.2 PROVIDER

**Provider** reads the ESA source packet from driver and

a. filters the packet and send to processor only the ESA source packet recognized to processor (event data packets, instrument configuration packets, measurement log packets)
b. manages the run id of the measurement
c. manages the start and stop time of the measurement.

#### *3.2.1.1 PACKET*

The telemetry packet managed are the following:

- event data packet
- instrument configuration packet: with the configuration of a single measurement
- measurement log packet: the log of a measurement
- start telecommand
- stop telecommand

A measurement starts with the start TC and terminates with the stop TC. A measurement session is a collection of measurements. This means that in the input byte streamt there are measurement with start and stop TC.
In test mode it is possible to configure the processor to ignore the start and stop TC. In this way the input data flow becomes a single measurement.

### 3.2.2 RUN ID

Each measurement is characterized with its run ID. When a new measurement starts the run ID is incremented in the following way:

1) DISCoS mode: when a new start TC is read
2) Test mode: when a new start TC is read, but with three operational mode:
    a. The run ID of the first measurement is zero;
    b. The run ID of the first measurement is read from a configuration file, as specified in the .processor;
    c. The run ID of the first measurement is determined from the name of the input file. This operational mode is useful with the raw file generated by DISCOS system in DISCoS playback mode.

In test mode it is possible to ignore the start/stop TC sequence. In this case the first run ID will be the unique run ID of the current session (but in the case b) the run ID of the following session is incremented).

### 3.2.3 START TIME AND STOP TIME

For each measurement it is necessary to know the start and stop time of the measurement. This times are determined in the following way:

1) DISCoS mode:
    a. In playback mode the date and time are read from DISCoS logs
    b. In other cases the time are determined by DISCoS system
2) Test mode:
    a. Date and time are read by the system
    b. The date and time are read from DISCoS logs, when the input files are raw files generated by DISCoS system.

### 3.2.4 OUTPUT

The Provider output (see number 2 in the preceding diagram) is represent by the following TM packet:

- Event data
- Instrument configuration
- Measurement log

## 3.3 **PROCESSOR**

A Processor accept, as input, the TM packet coming from Provider, elaborates it and generates one or more output files for each measurement.
The output files could be in every file format (FITS, XML, text). In the following examples there are the following cases:

- A Processor that generates a FITS file with the events
- A Processor that generates two output files: event file and histogram file with the event accumulation.

In Figure 9 the processor is divided in two parts:

- **Part A**: this is the general purpose part, and this part manages
    **1)** The logic of measurement session (see Figure 5)
    **2)** The logic of a measurement (vedi Figure 6, Figure 7 e Figure 8)
    **3)** The reading of the TM packets coming from *Provider*
    **4)** Sending the input TM packet to many output destinations (not yet implemented)
    **5)** The interfacing with the QL
    **6)** The interfacing with the log book (not yet implemented)
- **Part B**: this is the more specialized part, and this part manages :
    **1)** The elaboration of data container into the TM packets
    **2)** The opening, closing and writing of the output files.

The part B should be written by users of the ProcessorLib.

For a processor working in test mode it is also possibile, if the check of start/stop TCs is enabled, the generation of a FITS file containing all the events sent between a stop and a start TC.

# 4. BUILDING A PROCESSOR

The realization of a processor means that it is necessary to create a C++ project and two libraries have to be linked:

1) PacketLib;
2) ProcessorLib

Two class must be realized:

1) A derived class of the Processor class that implements all the pure virtual methods
2) A derived class of the OutputFileProcessor class, one for each output file that implements all the pure virtual methods
3) The main(), with the calls for the instantiation of the processor (see below).
4) The writing of the configuration files:
    a. .processor, with all the parameter for the run-time working of the processor
    b. .stream that describes the byte input stream
    c. one .packet, for each TM packet that contains the description of the telemetry packet

See [6] for 4.a and 4.b.

## 4.1 .PROCESSOR

### 4.1.1 DISCOS MODE

```
[Processor]
-- Configuration file for PacketLib
CAL-CSIBarsTE_DISCOS.stream
-- output file flag
true
-- campaign ID
cer
-- test_level
0
-- packet ID with event data
1
-- packet ID of start telecommand
none
-- packet ID of stop telecommand
none
-- packet ID instrument configuration
none
-- packet ID measurement log
none
-- extra parameters ----------------------------
```

```
[Provider]
-- 0 DISCOS, 1 SingleInput
0
-- acquisition type (chain) 0 = ACQ_OLD (playback mode)
0
-- direcotory with log file (N.A. for DISCOS)
/home/archive/log/
-- only for playback mode (acq_type=ACQ_OLD) acq_type of current playback
mode
hbrs
-- extra parameters ----------------------------
[InputProvider]
-- channel of shared memory for event data
18
-- channel of shared memory for instrument configuration
none
-- channel of shared memory for instrument measurement log
none
[OperationalMode]
-- 0: ignore start/stop telecommand
-- 1: start measurement with start TC, stop measur. with stop TC or EOI
0
-- 1: write output data between a stop and a start
-- 0: don't write output
0
[Monitor]
-- 0 Dummy, 1 DISCOS
1
-- extra parameters ----------------------------
-- channel
28
[FITS key]
--key 1
TELESCOP
Agile
--key 2
INSTRUME
Grid
--key 3
MODEL
October 2001
--key 4
DETNAM
MCAL
--key 5
HOSTCOMP
TESRE T.E.
```

```
--key 6
DATATYPE
Diagnostic
```

### 4.1.2 TEST MODE

```
[Processor]
-- Configuration file for PacketLib
CAL-CSIBarsTE_File.stream
-- output file flag
true
-- campaign ID
cer
-- test_level
FEE
-- packet ID
1
-- packet ID of start telecommand
none
-- packet ID of stop telecommand
none
-- packet ID instrument configuration
none
-- packet ID measurement log
none
-- extra parameters ----------------------------
[Provider]
-- 0 DISCOS, 1 SingleInput
1
-- acquisition type (chain) 0 = ACQ_OLD (playback mode)
0
-- direcotory with log file (N.A. for DISCOS)
/home/archive/log/
-- only for playback mode (acq_type=ACQ_OLD),  acq_type of current playback
mode
hbrs
-- extra parameters ----------------------------
-- directory for writing FITS file
fits/
[InputProvider]
-- This section is only for SingleInput
-- mode for the determination of run id 0: start from 0, 1: read the run id
from a file
-- 2: for playback mode, read from filename
1
-- file name that contains run id
runid.run
```

```
-- reading of start and stop time of measurement 0: system date and time 1:
first packet in input
-- and last packet in input 2: start/stop packet 3: DISCoS log file (N.I)
0
-- file name in input
/data1/archive/raw/science/0506/cer05060_011018.hrt
[OperationalMode]
-- 0: ignore start/stop telecommand
-- 1: start measurement with start TC, stop measur. with stop TC or EOI
0
-- 1: write output data between a stop and a start
-- 0: don't write output
0
[Monitor]
-- 0 Dummy, 1 DISCOS
0
-- extra parameters ----------------------------
-- channel
28
[FITS key]
--key 1
TELESCOP
Agile
--key 2
INSTRUME
Grid
--key 3
MODEL
October 2001
--key 4
DETNAM
MCAL
--key 5
HOSTCOMP
TESRE T.E.
--key 6
DATATYPE
Diagnostic
```

# 5. AN EXAMPLE OF PROCESSOR

## 5.1 GRIDCALDFETE PROCESSOR

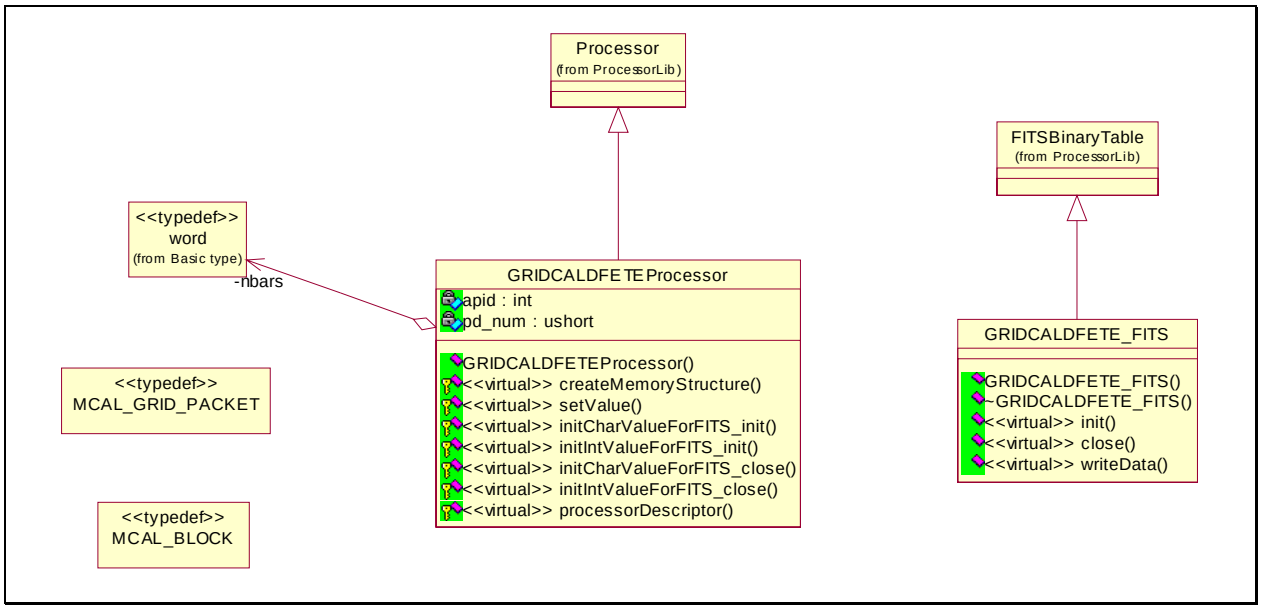


**Figure 10**: Class diagram of a processor

## 5.2 MAIN

The library supports the management of exceptions. This means that the block of code for generation of the processor must be in a try-catch block.

```
try
```

First of all it is necessary to instantiates the processor:

```
GRIDCALDFETEProcessor* gp = (GRIDCALDFETEProcessor*) new GRIDCALDFETEProcessor();
```

The second step is to load the configuration file:

```
gp->loadConfiguration("./CAL-DFE-TE/CAL-DFE-TE_File.processor");
```

After this, the measurement session should be started:

```
gp->startMeasurementSession();
```

At the end, the catch should be managed in the following way:

```
catch(PacketExceptionIO* e)


{
    cout << e->geterror();
}
```

```
        catch(PacketException* e)
        {
            cout << e->geterror();
        }
```

Di seguito è riportato il listato completo:

```
/***************************************************************************
                          main.cpp  -  description
                             -------------------
    begin                : Fri Mar  8 11:43:52 CET 2002
    copyright            : (C) 2002 by Andrea Bulgarelli
    email                : bulgarelli@tesre.bo.cnr.it
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream.h>
#include <stdlib.h>
#include "ProviderDISCOS.h"
#include "ProviderSingleInput.h"
#include "GRIDCALDFETEProcessor.h"
#include "common.h"
#include "MonitorDummy.h"
#include "PacketExceptionIO.h"

int main(int argc, char *argv[])
{
    try
    {
        struct tm* tm_int;
        time_t timevar1;
        time_t timevar2;
        time(&timevar1);

        GRIDCALDFETEProcessor* gp = (GRIDCALDFETEProcessor*) new GRIDCALDFETEProcessor();
        gp->loadConfiguration("./CAL-DFE-TE/CAL-DFE-TE_File.processor");
        gp->startMeasurementSession();

        time(&timevar2);
        cout << "Time: " << timevar2-timevar1 << endl;
        cout << "Media: " << gp->getTot_nrows() /   ((timevar2-timevar1)?(timevar2-timevar1):1) <<
endl;

        return 0;

    }
    catch(PacketExceptionIO* e)
    {
        cout << e->geterror();
    }
```

```
    catch(PacketException* e)
    {
        cout << e->geterror();
    }
}
```