Internal Report IASF 367/03

INTRODUCTION OF XML RELATED TECHNOLOGIES INTO THE PACKET PROCESSOR MODULE USED IN AN EGSE ENVIRONMENT FOR THE AGILE'S TEST EQUIPMENTS

E. FRANCESCHI, A. BULGARELLI, F. GIANOTTI, M. TRIFOGLIO

IASF/CNR - Sezione di Bologna

March 2003

Internal Report IASF 367/03

INTRODUCTION OF XML RELATED TECHNOLOGIES INTO THE PACKET PROCESSOR MODULE USED IN AN EGSE ENVIRONMENT FOR THE AGILE'S TEST EQUIPMENTS

E. FRANCESCHI, A. BULGARELLI, F. GIANOTTI, M. TRIFOGLIO

IASF/CNR - Sezione di Bologna

March 2003

SUMMARY – The so-called "packet processor" is a software module that manages a certain data flow for a specific payload (that is to say, a detector or another instrument placed, or to be placed, on a scientific satellite). The packet processors so far used by the IASF-BO (to build the Test Equipments needed for the functional tests and the scientific calibration of various instruments) are based onto a purposely developed set of C++ classes that, for the present, do not deal with XML formatted data at all (the FITS format is the only expected format for the data output, and the input files – needed to properly configure each processor – are simply statically created text files). We have therefore tried to introduce into the EGSE framework – referring to that of the AGILE mission – some XML-related technologies (and also the still experimental FITSML format, defined by the NASA/GSFC *XML Group*), planning the ad-hoc project here described, which involves the revision of a previously developed packet processor.

1 Introduction

AGILE[1] is a Small Scientific Mission (currently planned to be operational in 2005) devoted to high-energy astrophysics: it will operate in the 30 MeV – 50 GeV with imaging capabilities also in the 10–40 KeV range [1]. The AGILE mission is supported by the ASI (Italian Space Agency) and scientifically developed in CNR (Italian National Research Council) and INFN (National Institute for Nuclear Physics) laboratories.

As in all space missions, an extensive stage of test is needed, before the flight ready status can be achieved, to investigate and evaluate the potential issues related to each payload. Every instrument must be considered from different points of view: physical (mechanical, electrical, magnetic, thermal), functional, and also performance-related. To this purpose, a number of Test Equipments [TE] are required in order to stimulate, command and acquire data from the item under test.

IASF (the Space Astrophysics and Cosmic Physics Institute of the CNR), Section of Bologna[2], participates to the procurement of the TEs for the AGILE MiniCalorimeter subsystem [2] and the AGILE Payload EGSE (Electrical Ground Support Equipment) [3]. In particular, IASF provides the computer system, named Science Console [SC], which is in charge of acquisition, archiving,

[1] http://agile.mi.iasf.cnr.it/
[2] http://www.bo.iasf.cnr.it/

and processing of the instrument data during the Assembly, Integration and Verification [AIV] and the Calibration activities. A generic TE (or EGSE) layout is the one shown in Figure 1.

The software module that realizes the packet processors developed for the AGILE's TEs is based on a C++ library called ProcessorLib — that is itself based on the PacketLib, another C++ open-source library created as a framework for writing applications which deal with satellite telemetry source packets. But these libraries totally lack classes devoted to the management of XML formatted data, both in input and in output. In the basic configuration, actually, there is no need of getting information from an external database, and the FITS (Flexible Image Transport System) format[3], for the data output, is the only expected format.

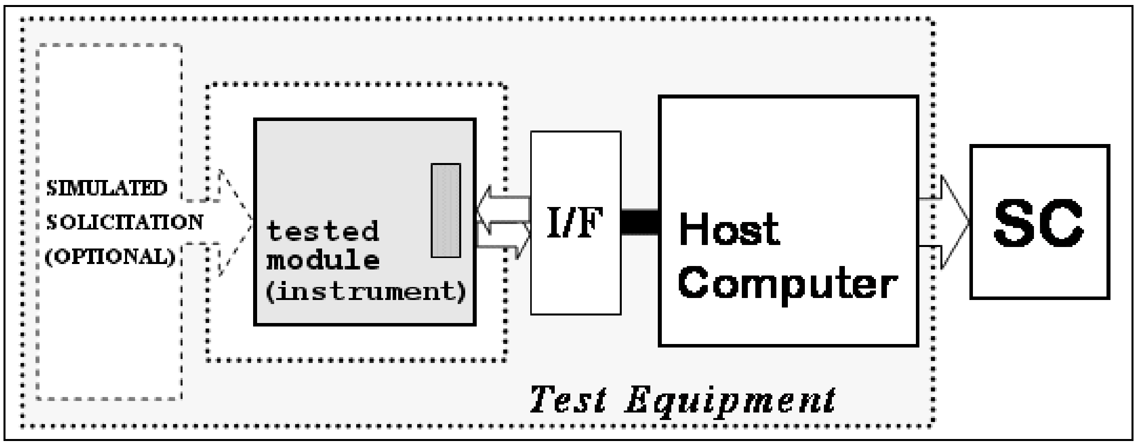


Figure 1 – A generic Test Equipment (or EGSE) layout

2 Operating context

Evaluating the use of XML related technologies, we have first of all created an XML database (called "packet descriptors database") that contains the description of the structure (down to the bit level), along with some additional information needed to control the carried data, for each kind of packet that the system shall be able to deal with — taking also care (for compatibility purposes) of the packet typology foreseen by the already developed C++ libraries. The processing of a packet is thus bound to the presence in the database of a corresponding consistent description: the packet processor itself is initialized on the basis of the data supplied by that DB, by means of configuration files – the same ones formerly used, but now dynamically generated.

Each processor, made with reference to a particular TE, must afterwards – during the processing of a therefore specific packet type – retrieve some of the support data from the packet descriptors database; this is because that additional but now focused information can be required by the modules that may follow (a QuickLook module[4], for instance – as in the project scheme shown in Figure 2), and has thus to be attached to the processor output data.

[3] The FITS format, utilized for the first time in the late seventies, is, at the present time (endorsed by NASA and the International Astronomical Union), the most used format in order to store astronomical data. Several links to information about that format can be found on http://fits.gsfc.nasa.gov/.

[4] The term QuickLook [QL] means a monitoring action with the display of log data, where the log has been generated by a simple computation (possibly also based on some combination of other quicklook curves). A QL module is usually intended to make it easy to identify particular features in a section of log.

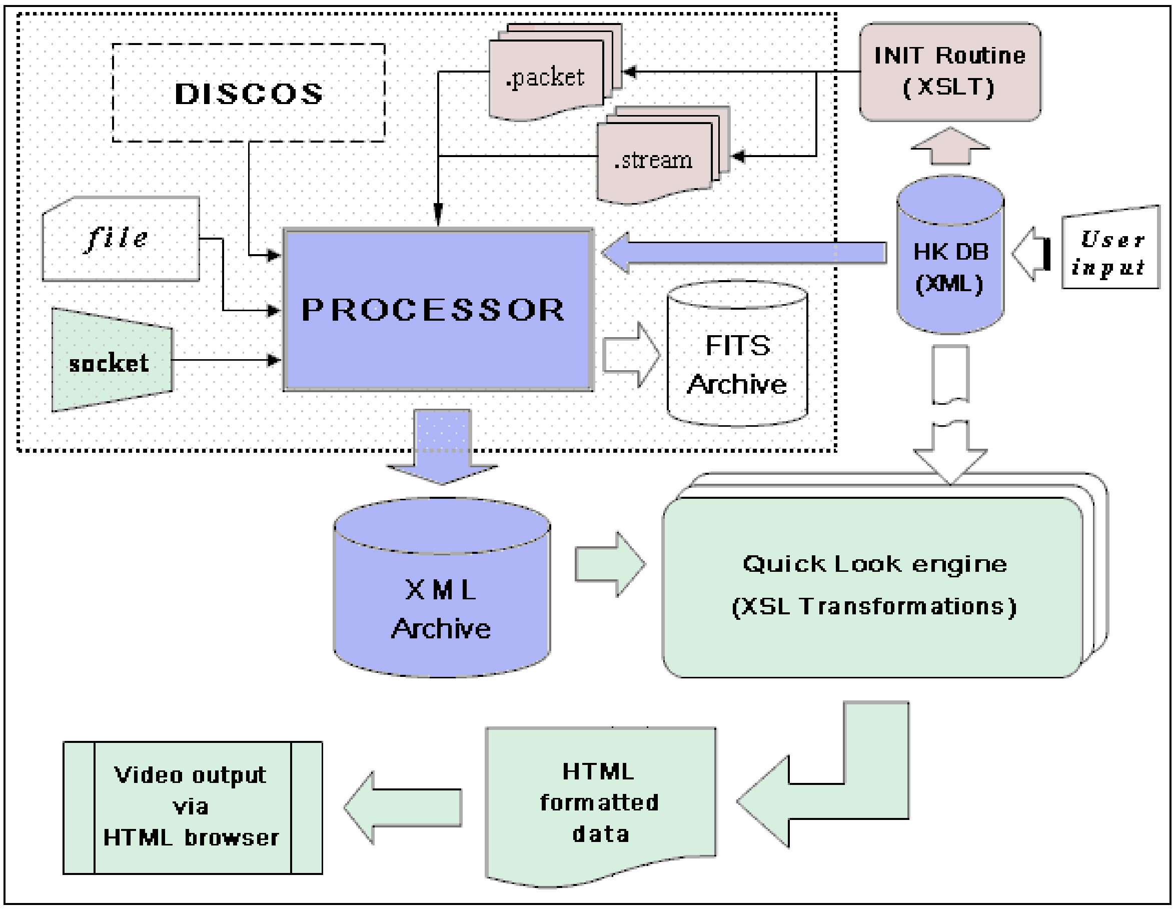


Figure 2 – The Project Scheme (with the core modules in blue, the initialization pattern in light brown, and the following foreseen blocks in light green)

The processor output data itself shall be expressed in an XML format, but the question is which particular XML language should be used: a new one, to be specially created (like the one used for the descriptors DB mentioned above); or, instead, an already developed one, to be searched and taken somewhere else.

3 Identifying an XML language to represent the processor output data

Two possible alternatives have been taken into account regarding the format to adopt in order to file the data resulting from the packet processor. On the one hand, the building from zero of a language designed to that purpose; on the other end, the adoption – with the necessary adjustments – of an already existent schema.

Defining a new language from format specifications due to a particular case (that of the specific FITS format used by AGILE) might easily lead to results lacking generalization. Letting aside the quality of the final outcome, such an operation would be very expensive — the time required to develop it being rather long, as already experienced in setting up the XML-Schema [8][9] to record the packet descriptors.

It has therefore seemed advisable to choose the other option, though that choice too is not without dangers. Basically, the difficulty lies in finding a language actually suitable for the purpose, that is

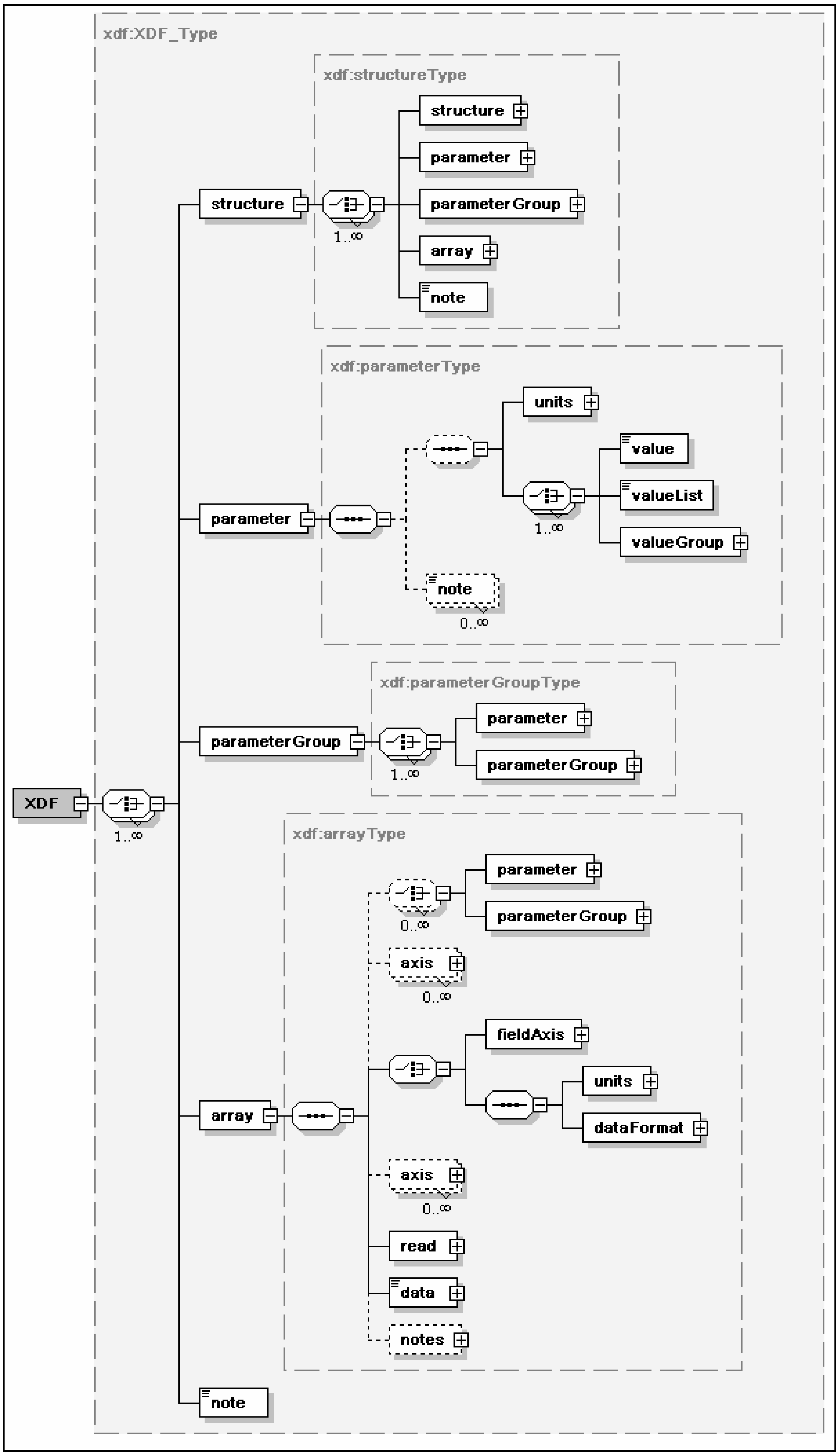


Figure 3 – The XDF language: an overall view

to say, capable to meet the special problems the case presents. In fact, what is needed is a scheme which can adequately represent all, or at least nearly all, the information which the FITS files formerly used contain.

3.1 The XDF and FITSML languages

XDF (eXtensible Data Format for scientific data, [12]) is the language that, at first, has been singled out as potentially suitable (following a research carried out on the XML technologies at present available). Designed by the XML Group of the NASA Goddard Space Flight Center [GSFC], and still in development, XDF is a mark-up language which intends to act as reference standard for the representation of scientific data, aside from the specific typology and the special field of application to which they are related. The aim is to supply a means which might enable exchange of information from one scientific field to any other. The XDF may then be ‘extended’, as the case requires, to every specific context, so as to better meet the specific needs which might occur in processing the data regarding a particular field.

In the XDF, data are described according to the view that researchers and scientists have of them. The XDF syntax regards any datum as an object, assembled starting from two typologies of basic objects: one single parameter to which a certain value or a certain range of values corresponds; or an array field, that is to say a grid of sampled values belonging to scalar or vector fields, within an n-dimensional space. These two basis objects are often used to form lists. A *record* can thus be defined as a list of values related to a given set of parameters, and a *table* as a list of such records.

The `<XDF>` tag thus becomes a container for parameters, field arrays and tables (see Figure 3). In the two latter cases one has to deal practically with `<array>` elements, within which there can be both the `<data>` (as ordered lists of strings, or of numerical values) and the related `<axes>`. Arrays and parameters may then be grouped in a `<structure>`, as shown very schematically in Figure 4. Unique identifiers can also be joined to the axes so that they can be used in more points of the structure without the need to redefine them locally on every occasion. The list of the values for every axe can be drawn up manually, or can be generated by exploiting a mechanism of

```
<XDF>
  <structure name="example">
    <array name="name of the array" description="textual explanation">
      <units>units of measurement used</units>
      <dataFormat>
        data format used: integer, float (with a given precision), etc...
      </dataFormat>
      <axis name="X-axis">
        <values>list of values for elements along one dimension</values>
      </axis>
      <axis name="Y-axis">
        <values>list of values along onother dimension</values>
      </axis>
      <read>
        infos on the ordering of the data values
        and definition of the rules to handle them
      </read>
      <data>
        this is the place for real data...
      </data>
    </array>
    <array>
      some other array of data...
    </array>
  </structure>
</XDF>
```

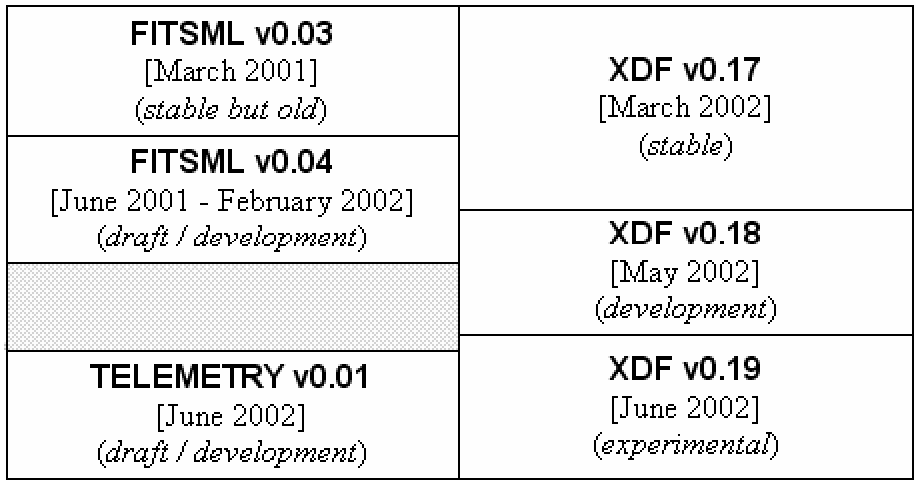
Figure 4 – The XDF language: a sample file

automatic indexing supplied by the language, or by one implemented by means of a suitable script; or can even be introduced by referring to an external source (a Java applet, for instance) which in reply to given parameters may produce the sequence required.

To represent the data, a similar flexibility can be used. For the real data, however, it will also be possible to exploit the powerful mechanism of “Notation Identities” supplied by XML architecture (see "4.7 Notation Declarations" in [6]). That means that data can be represented both as plain text – typically as tabular data – or be codified according to any binary format, as long as there is an application (related to its `<read>` tag) capable to interpret them. Worth mentioning is the possibility to explicitly indicate – so that they can be automatically processed – the several units of measure that have been used, whatever they are (also provided is the use of multiplicative constants, in a floating-point or exponential form). Moreover, such indication can be wholly ‘exportable’, if given by using entities purposely prearranged (at DTD level), because in any case it can then be referred to an equivalent form expressed through units of the International System.

But, though quite interesting, the XDF does not appear to meet all the requirements of the ongoing project, which must process exactly the same information which was formerly kept in a FITS file. A more detailed analysis of the peculiarities of the language shows that the XDF clearly does not intend to introduce any formalism whatever suitable to the representation of metadata, as they are typically gathered (within the data files) at the beginning, in special headers, strictly linked – as to form and content – to the *specific* field of application. Which is further confirmation of the fact that XDF intends to propose itself just as a set of standard specifications, from which possibly to start afterwards to develop other, more specific XML languages.

Actually, a language based on the XDF and whose very aim is the representation of astronomic data does exist: it is called FITSML (FITS Mark-up Language, [13]). This language, devised by the same group that works on XDF, is nothing but – as its very name suggests – an XML revisiting of the FITS standard. It aims not so much at redefining *in toto* the original data, as at a simpler and more immediate *remapping* of the FITS syntax within an XML logic (also thanks to the inclusion of the XDF scheme), thus avoiding to disconcert the users (who for more than twenty years have been accustomed to the same format) and therefore favor the migration to the one that, in their plans, should become the new standard.



FITSML v0.03 [March 2001] *(stable but old)*	**XDF v0.17** [March 2002] *(stable)*
FITSML v0.04 [June 2001 - February 2002] *(draft / development)*	
▨	**XDF v0.18** [May 2002] *(development)*
TELEMETRY v0.01 [June 2002] *(draft / development)*	**XDF v0.19** [June 2002] *(experimental)*

Figure 5 – XDF and derived languages: last developed versions

4 Defining a stable version for FITSML

The notes and definition files regarding the FITSML language can be found on the XDF home page[5], in the section entitled "Markup Languages that Inherit from XDF" (the table in Figure 5 lists those languages and gives, at the same time, an idea of the connection that exists between their versions and those developed for XDF).

Looking over that web page, we have noticed a somewhat chaotic situation, both for FITSML and, even if to a lesser extent, XDF. For example, with regard to the version of FITSML that is now being developed (labeled "v0.04"), it was given both a DTD description and an XML-schema (the latter one being, incidentally, still presented only as “PR”, or “Proposed Recommendation”); but a quick comparison between the two grammars has enabled to note, from the very beginning, significant differences – in spite of the same version number. And the search that was subsequently carried out at a wider range on the Web, in order to establish which description was the one better suited for our purpose, could not lead to any satisfactory conclusion. On the contrary: all the DTD files thus identified presented, on the whole, several inconsistencies between version number, development date (both internally declared), and timestamp of the very file — not to mention the contents, which were different even among DTDs with the same, therefore only *apparent*, version number. Not even the description given as syntactic tree (this one too taken from the XDF home page) did not offer any sure reference, as it was still linked to the old 0.03 version.

What is more, we have found several mistakes in the FITSML DTD v0.04 itself, as well as in the FITSML XSD v0.04-PR. In the end we have resolved to contact the authors of the language: and then a temporary collaboration, lasted a few e-mails, has been enough (thanks to Edward J. Shaya, of the GSFC XML Group) to correct the errors and to define a stable and consistent XML-Schema for the FITSML v0.04, along with a purposely revised version 0.17 of the correlated XDF schema. Also, the XSD files published by the GSFC XML Group on its web repository[6] have been accordingly updated: the two language schemes used[7], only partially outlined in the following figures, can thus be freely looked up going to that web site.

5 Mapping of the FITS format in use with AGILE

Once established – and made ready for use – the XML language for the data coming from the processor, the next step was to find a mapping suitable for the specific extension of the FITS format which is being used within the AGILE mission.

A FITS file can be schematized, for our purposes, with a sequence of three blocks, as shown in Figure 6: the first part contains *keywords*[8] whose value is very general; its purpose is little more (it depends on circumstances) than simply introducing the second part, which, in turn, beside supplying information on the context, describes the special format – conceptually, but not physically, a table – in which data are organized in the third part. A parameter corresponds to each column,

---

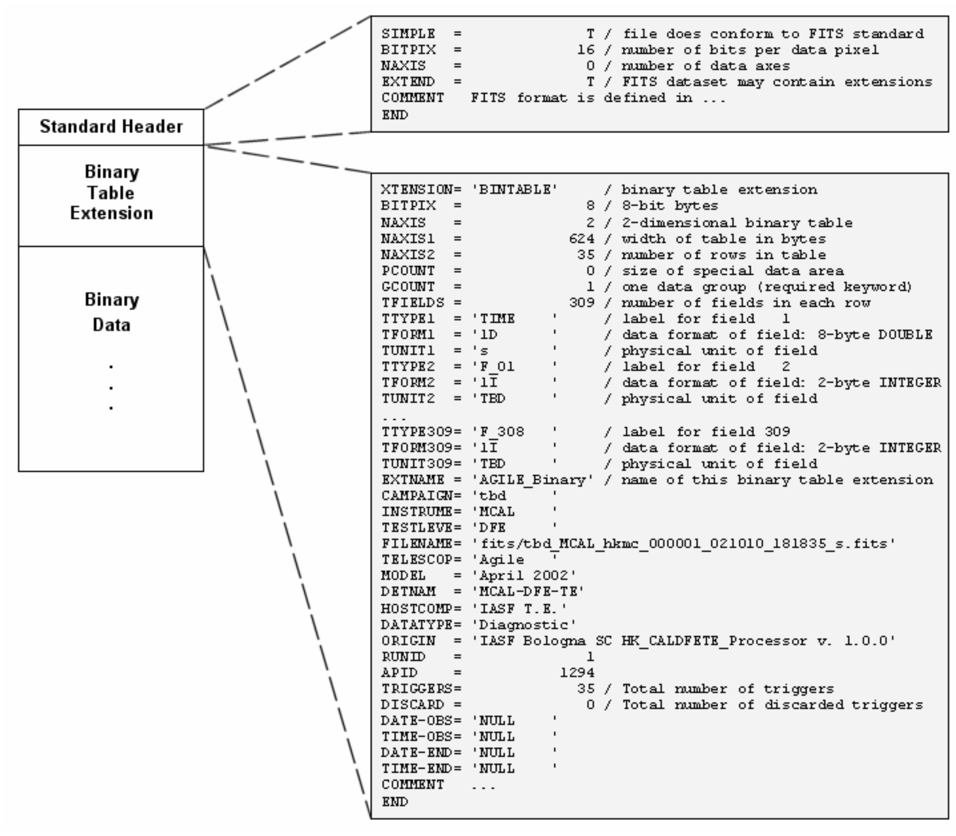
[5] http://xml.gsfc.nasa.gov/XDF/

[6] This directory contains schemes, samples, and also some documentation, related to all the languages that the GSFC XML Group has developed during the last four years. It is publicly accessible at the URL http://xml.gsfc.nasa.gov/DTD/.

[7] Actually, the schemes used (for the XDF as well as for the FITSML) are not a mere copy of those published on the Web, because of some further little modifications we thought appropriate – that anyway do not change the real structure of the two languages at all.

[8] For each informatory field, in the first two blocks, there are 80 bytes, strictly arranged in the sequence: ‘keyword’ – value – optional explanatory comment.

and each line therefore represents a packet of telemetry, which is then prefixed with the relevant timestamp.



```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                   16 / number of bits per data pixel
NAXIS   =                    0 / number of data axes
EXTEND  =                    T / FITS dataset may contain extensions
COMMENT   FITS format is defined in ...
END
```

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                  624 / width of table in bytes
NAXIS2  =                   35 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                  309 / number of fields in each row
TTYPE1  = 'TIME    '           / label for field   1
TFORM1  = '1D      '           / data format of field: 8-byte DOUBLE
TUNIT1  = 's       '           / physical unit of field
TTYPE2  = 'F_01    '           / label for field   2
TFORM2  = '1I      '           / data format of field: 2-byte INTEGER
TUNIT2  = 'TBD     '           / physical unit of field
...
TTYPE309= 'F_308   '           / label for field 309
TFORM309= '1I      '           / data format of field: 2-byte INTEGER
TUNIT309= 'TBD     '           / physical unit of field
EXTNAME = 'AGILE_Binary' / name of this binary table extension
CAMPAIGN= 'tbd     '
INSTRUME= 'MCAL    '
TESTLEVE= 'DFE     '
FILENAME= 'fits/tbd_MCAL_hkmc_000001_021010_181835_s.fits'
TELESCOP= 'Agile   '
MODEL   = 'April 2002'
DETNAM  = 'MCAL-DFE-TE'
HOSTCOMP= 'IASF T.E.'
DATATYPE= 'Diagnostic'
ORIGIN  = 'IASF Bologna SC HK_CALDFETE_Processor v. 1.0.0'
RUNID   =                    1
APID    =                 1294
TRIGGERS=                   35 / Total number of triggers
DISCARD =                    0 / Total number of discarded triggers
DATE-OBS= 'NULL    '
TIME-OBS= 'NULL    '
DATE-END= 'NULL    '
TIME-END= 'NULL    '
COMMENT   ...
END
```

Figure 6 – Sample FITS header (for the AGILE mission)

To reproduce the data (which, in order to be interpreted with an ordinary stylesheet should no longer be in binary form[9]) the structures available from XDF will be sufficient, whereas to describe the "binary table extension" also the FITSML typical ‘additions’ should necessarily be used.

We have thus taken and studied a sample FITS file, previously generated by an already developed packet processor (related to the Test Equipment for the AGILE's MiniCalorimeter Detector Front-End). The header has been extrapolated (see Figure 6) and all that was no longer pertinent (because redundant, or simply useless, from the new XML point of view[10]) has been rejected; then, the best method to encapsulate the information thus filtered has been pursued.

As a matter of fact, the mapping is surely non univocal, and it is besides determined by the XDF structure that will be chosen to represent the real data. We have also taken into account – anticipating the addition of a following QuickLook module, as mentioned above (see again the scheme outlined in Figure 2) – in what way the pieces of information taken from the packet descriptors database (the ones later needed to obtain the out-of-range values) could be included, and how complex then the XSLT code needed to obtain the desired output data would have to be, in relation to the particular solution that could be adopted.

[9] In an XML logic, a textual representation of data might seem quite obvious. But this is not at all true if one uses XDF, given its unusual peculiarities (as seen above).

[10] See the notes at the end of the XSD document related to FITSML v0.04 (available on http://xml.gsfc.nasa.gov/DTD/).

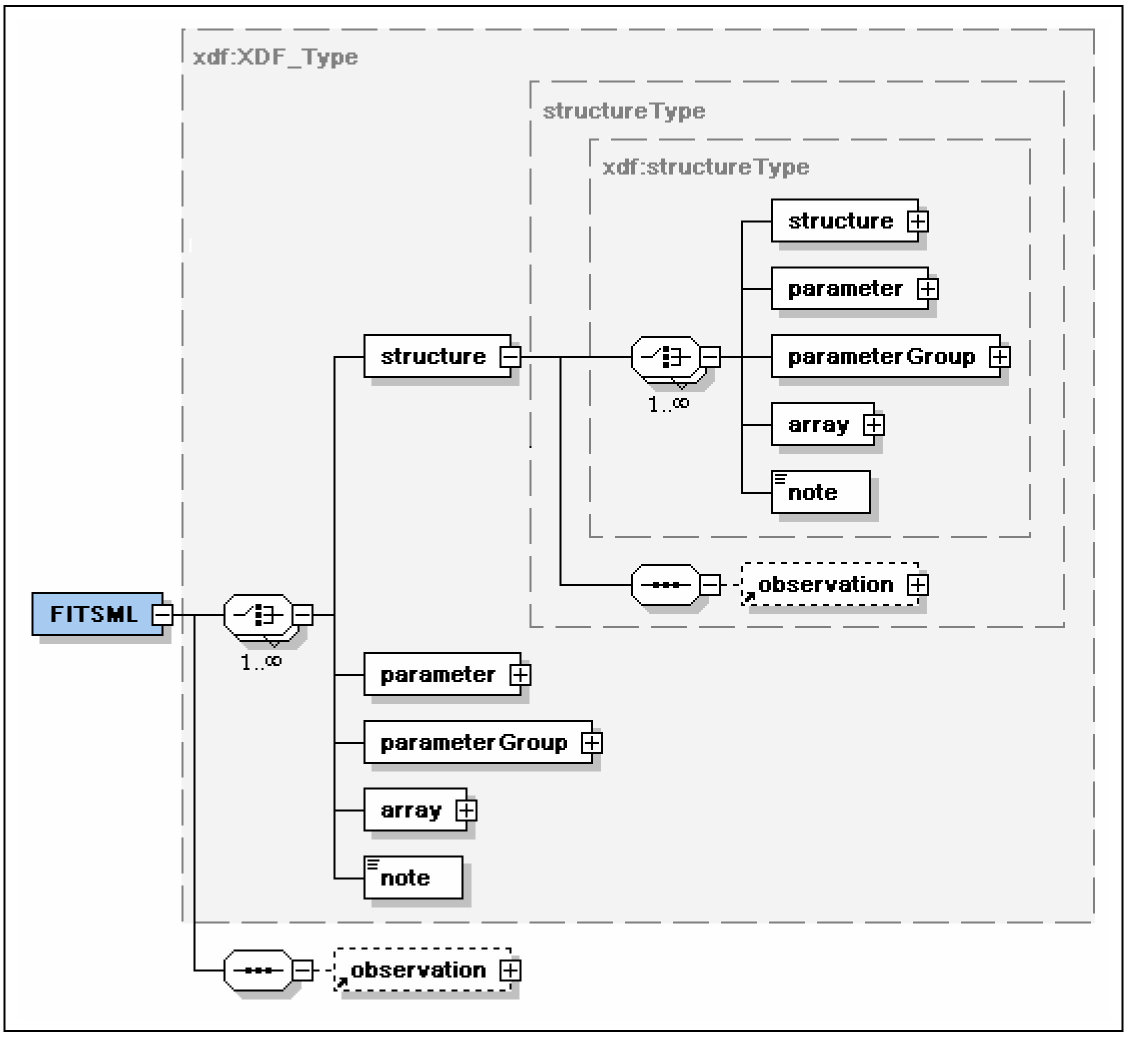


Figure 7 – FITSML v0.04: the root element and the "structureType"

The key-point lies in how the choice to represent the set `TTYPEn-TFORMn-TUNITn` (related to each parameter) is made. At first it was thought convenient to give of them a simple, straightforward representation, based on the tag `<parameter>` (of the same XDF and inherited, unchanged, from the FITSML – see Figure 7) in a merely sequential lay-out, or possibly a tabular one – which could be obtained by assuming its inclusion inside an `<array>`. Thus, together with the value itself of the datum (put into "parameter/value"), all the descriptive information related to it (`TUNIT` into "parameter/units", `TTYPE` into "parameter/@name", and `TFORM` into "parameter/@datatype"[11]) could be obtained by means of a most easy and effective XPath expression [10]; as for the upper and lower bounds needed for warnings and alarms, four `<note>` tags could be used, conveniently diversified by means of the `mark` attribute (a characters string). But such direct correlation between datum and its description, if from the one hand may enable a more immediate implementation of the stylesheet, on the other hand may imply a substantial increase of the dimensions of the resulting file. The redundancy that would be introduced into the data would undoubtedly be excessive and unjustified.

[11] Referring to the W3C Recommendation (the one dated 16 November 1999) for the XML Path Language Version 1.0 [10], the symbol '@' stays for the *AxisSpecifier* "`attribute::`", and so the expression "A/@B" means "the attribute B of the element A".

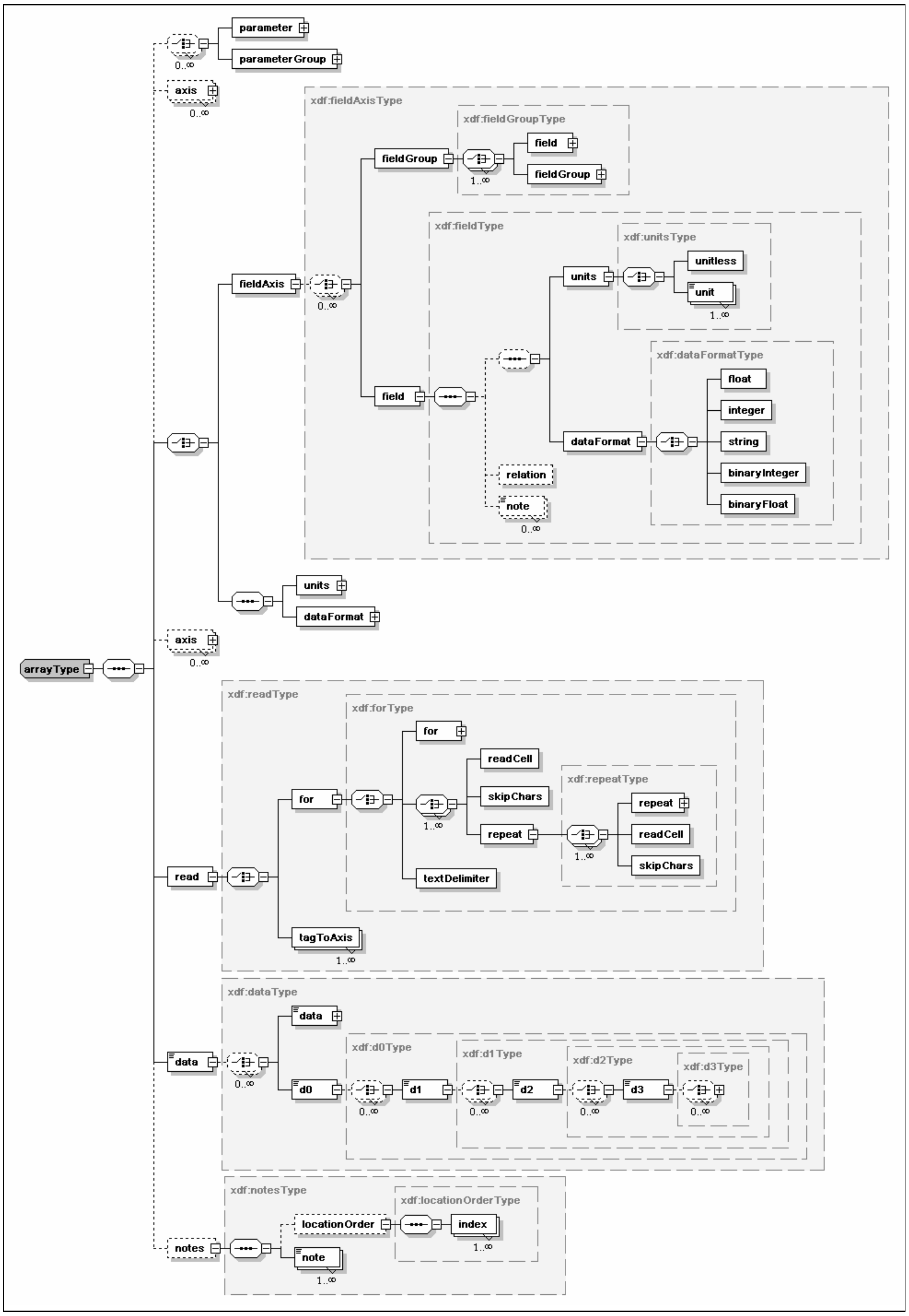


Figure 8 – XDF v0.17: detailed view of the "arrayType"

A rather more complex structure has been therefore chosen, which, in accordance with some suggestions implicitly given by the authors of the language, enables to exploit at the very best some potentialities of the XDF, so as to obtain one only descriptive structure, detached, but at the same time clearly and organically related to the data – once more organized in a strictly matrix form. In fact, it is question to use, for the whole set of the data contained in the file, one only `<array>`, within which all the related information can be gathered in one `<fieldAxis>` (with a `<field>` for each parameter), and the data under one `<data>` tag (with its lines defined by `<d0>` and its columns by `<d1>`); the link between the two structures is made by an ad-hoc created "`read/tagToAxis`" element (see the chart of Figure 8, and compare it with the demo output file shown in Appendix C).

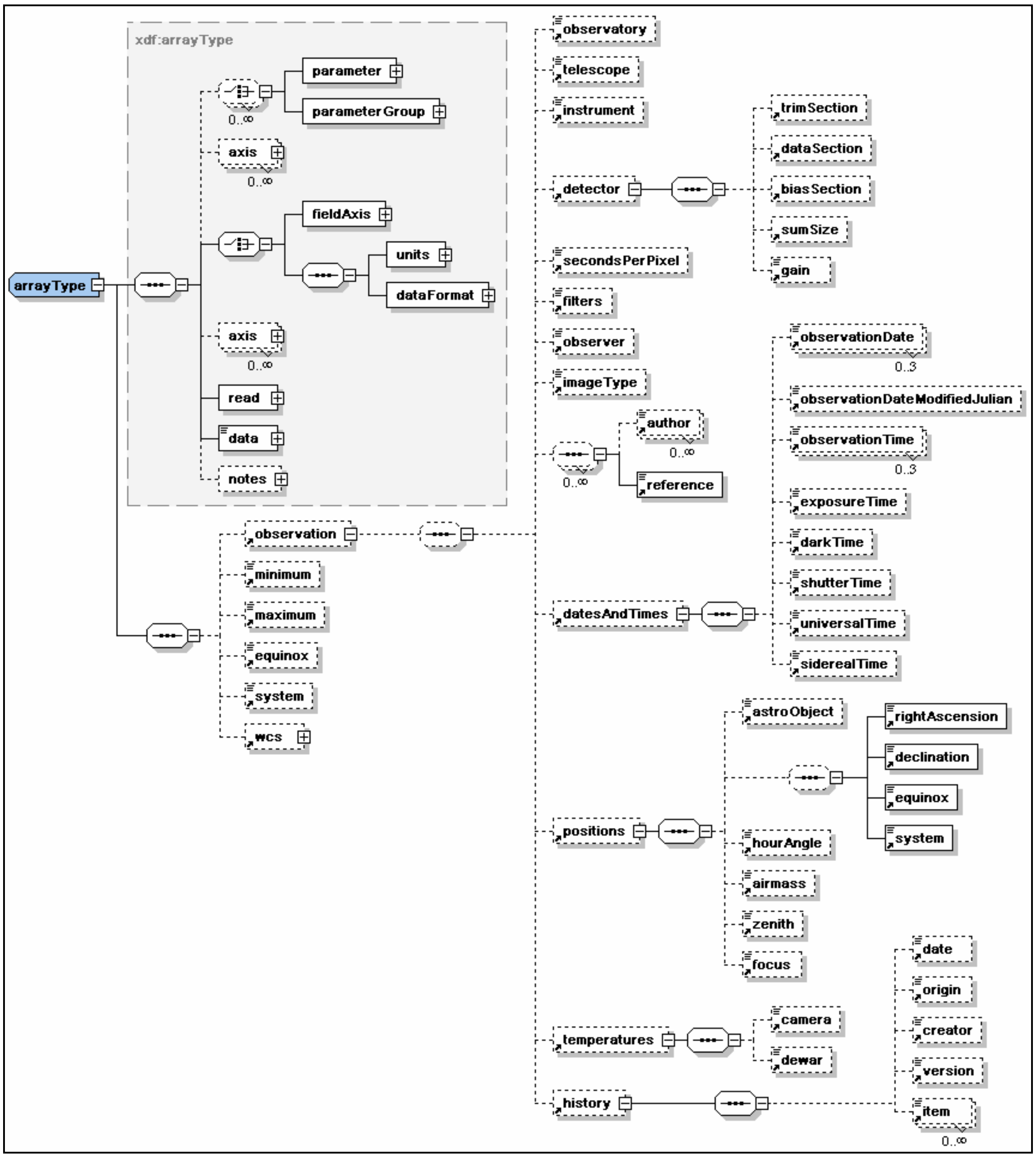


Figure 9 – FITSML v0.04: the "arrayType" redefined (and detailed view of the "observation" tag)

The XPath expression [10], useful to establish the link between the data and the corresponding boundaries, is in this case surely less evident, but it does exist and is univocal.

Assuming to scan the data table with two XSLT loops [11], nested as follows:

```
<xsl:for-each select="xdf:FITSML/xdf:array/xdf:data/xdf:d0"><!-- scanning lines -->
<xsl:for-each select="xdf:d1"><!-- scanning columns -->
```

and the current context position saved in a proper XSLT variable:

```
<xsl:variable name="p" select="position()"/>
```

the expression that can then be used to retrieve a limit value and test the current data is, for instance:

```
<xsl:when test=". &lt; ../../../xdf:fieldAxis/xdf:field[$p]/xdf:note[@mark='alarmLow']">
```

By resorting to the `<array>` and `<field>` elements it is also possible to use 'children' added by the FITSML to the related datatype (see Figure 9 and Figure 10). As for the transcription of the limit values, for instance, instead of utilizing only four attributes chosen among those defined by the XDF (thus obtaining a result at least a little botched up), one can consider the possibility of a combined use, for a given `<field>`, of the `<minimum>`/`<maximum>` additions, and of the `underflowValue`/`overflowValue` attributes. For this particular aspect, however, the first hypothesis has been chosen – that is the one that makes use of four `<note>` elements (already attached to the original XDF `fieldType`), conveniently defined. In this way the consulting mechanism will be, at least, homogeneous.

As for the other pieces of information contained in the FITS header (more disconnected than the ones mentioned up to here) the point has been to find out which ones were explicitly expected by the FITSML and which ones were not, trying at the same time to make out which ones were to be considered as belonging to each specific packet and which ones were instead a constant of the instrument or of the whole mission; with the consequent choice of pre-defined tags or of generic `<note>` elements (but marked out with the original keyword) — then included at the same level of the data (in the `<array>`), or as a common feature that depends directly on the root element (`<FITSML>`). The final result can be seen in Figure 11.

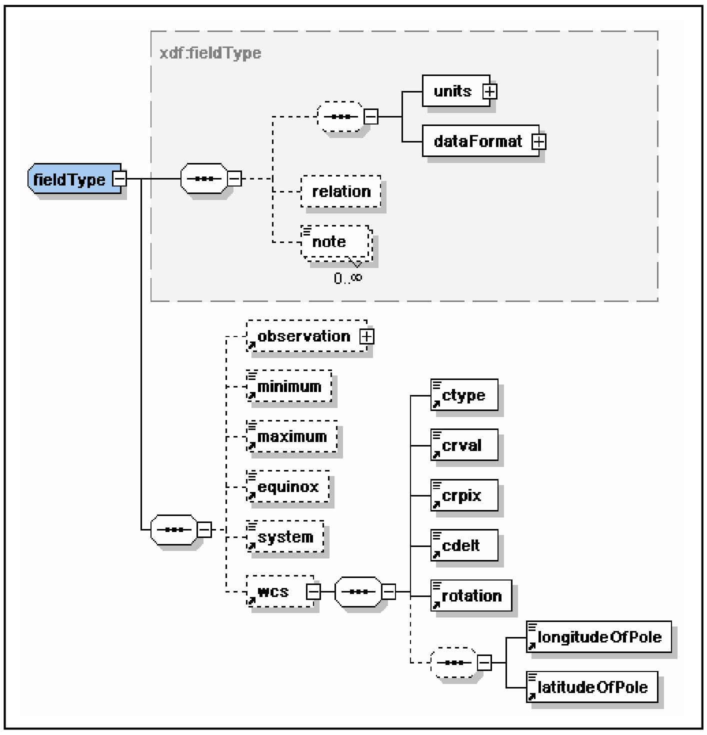
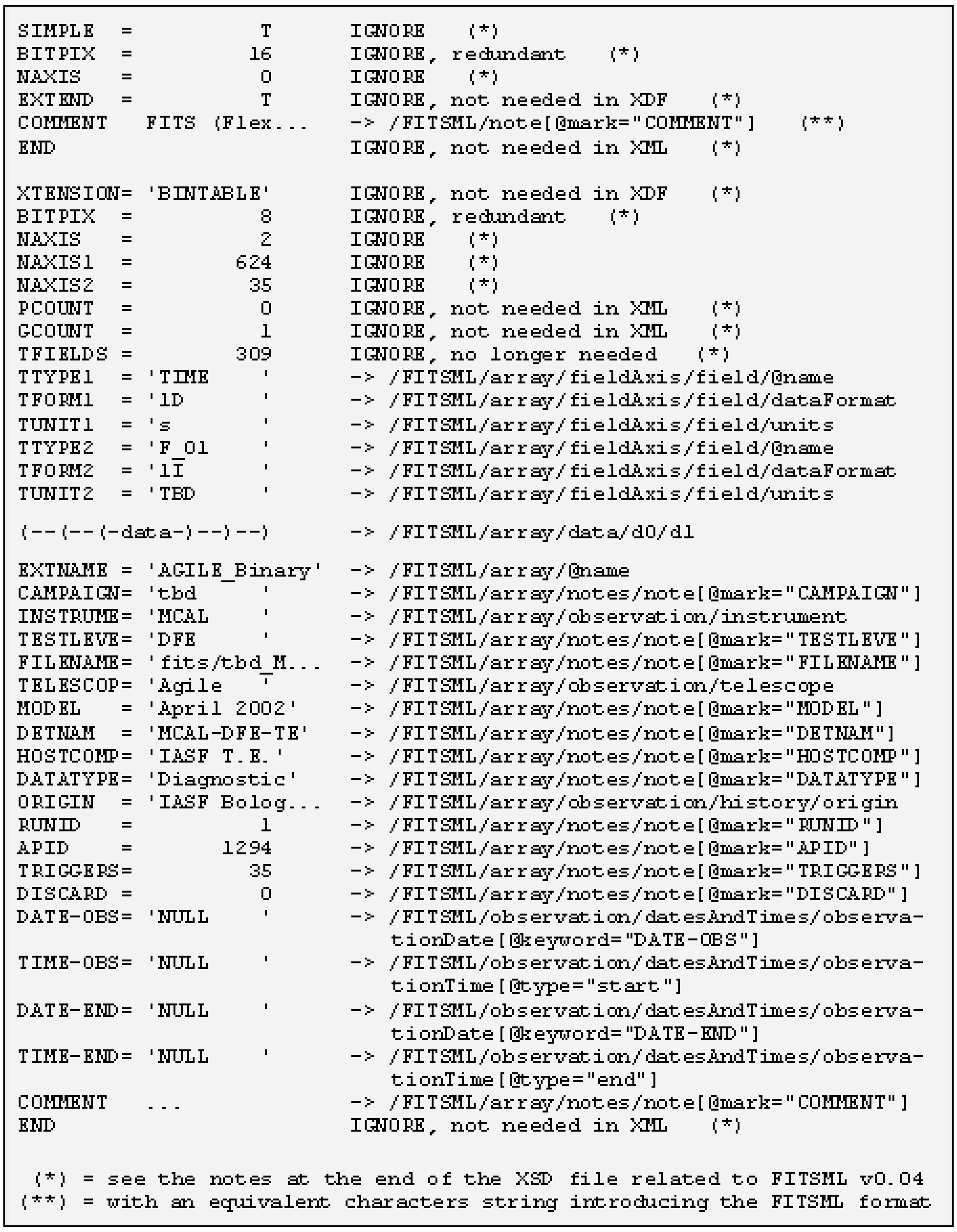


Figure 10 – FITSML v0.04: the "fieldType" redefined (and detailed view of the "wcs" element)

Once thus defined the necessary FITS to FITSML mapping, we have finally started to develop the planned demo processor (one that should implement as requested the DB queries and the XML format for the data output), taking as a model one packet processor already developed, based upon the PacketLib and the ProcessorLib libraries.



```
SIMPLE  =           T      IGNORE   (*)
BITPIX  =          16      IGNORE, redundant   (*)
NAXIS   =           0      IGNORE   (*)
EXTEND  =           T      IGNORE, not needed in XDF   (*)
COMMENT   FITS (Flex...    -> /FITSML/note[@mark="COMMENT"]   (**)
END                        IGNORE, not needed in XML   (*)

XTENSION= 'BINTABLE'       IGNORE, not needed in XDF   (*)
BITPIX  =           8      IGNORE, redundant   (*)
NAXIS   =           2      IGNORE   (*)
NAXIS1  =         624      IGNORE   (*)
NAXIS2  =          35      IGNORE   (*)
PCOUNT  =           0      IGNORE, not needed in XML   (*)
GCOUNT  =           1      IGNORE, not needed in XML   (*)
TFIELDS =         309      IGNORE, no longer needed   (*)
TTYPE1  = 'TIME    '       -> /FITSML/array/fieldAxis/field/@name
TFORM1  = '1D      '       -> /FITSML/array/fieldAxis/field/dataFormat
TUNIT1  = 's       '       -> /FITSML/array/fieldAxis/field/units
TTYPE2  = 'F_01    '       -> /FITSML/array/fieldAxis/field/@name
TFORM2  = '1I      '       -> /FITSML/array/fieldAxis/field/dataFormat
TUNIT2  = 'TBD     '       -> /FITSML/array/fieldAxis/field/units

(--(--(-data-)--)--)       -> /FITSML/array/data/d0/d1

EXTNAME = 'AGILE_Binary'   -> /FITSML/array/@name
CAMPAIGN= 'tbd     '       -> /FITSML/array/notes/note[@mark="CAMPAIGN"]
INSTRUME= 'MCAL    '       -> /FITSML/array/observation/instrument
TESTLEVE= 'DFE     '       -> /FITSML/array/notes/note[@mark="TESTLEVE"]
FILENAME= 'fits/tbd_M...   -> /FITSML/array/notes/note[@mark="FILENAME"]
TELESCOP= 'Agile   '       -> /FITSML/array/observation/telescope
MODEL   = 'April 2002'     -> /FITSML/array/notes/note[@mark="MODEL"]
DETNAM  = 'MCAL-DFE-TE'    -> /FITSML/array/notes/note[@mark="DETNAM"]
HOSTCOMP= 'IASF T.E.'      -> /FITSML/array/notes/note[@mark="HOSTCOMP"]
DATATYPE= 'Diagnostic'     -> /FITSML/array/notes/note[@mark="DATATYPE"]
ORIGIN  = 'IASF Bolog...   -> /FITSML/array/observation/history/origin
RUNID   =           1      -> /FITSML/array/notes/note[@mark="RUNID"]
APID    =        1294      -> /FITSML/array/notes/note[@mark="APID"]
TRIGGERS=          35      -> /FITSML/array/notes/note[@mark="TRIGGERS"]
DISCARD =           0      -> /FITSML/array/notes/note[@mark="DISCARD"]
DATE-OBS= 'NULL    '       -> /FITSML/observation/datesAndTimes/observa-
                              tionDate[@keyword="DATE-OBS"]
TIME-OBS= 'NULL    '       -> /FITSML/observation/datesAndTimes/observa-
                              tionTime[@type="start"]
DATE-END= 'NULL    '       -> /FITSML/observation/datesAndTimes/observa-
                              tionDate[@keyword="DATE-END"]
TIME-END= 'NULL    '       -> /FITSML/observation/datesAndTimes/observa-
                              tionTime[@type="end"]
COMMENT   ...              -> /FITSML/array/notes/note[@mark="COMMENT"]
END                        IGNORE, not needed in XML   (*)


 (*) = see the notes at the end of the XSD file related to FITSML v0.04
(**) = with an equivalent characters string introducing the FITSML format
```

Figure 11 – FITS to FITSML mapping (adopted in the AGILE context)

6 PacketLib and ProcessorLib: a quick overview

The PacketLib [4] is a C++ open-source software library developed for Linux/Unix platforms with the intent to provide a common reference for those applications that have to manage telemetry source packets — provided that the packets are compliant with the ESA Telemetry and Telecommand Standards[12]. Its main ambition is to allow a strategic software re-use and, consequently, a more rapid development of TEs and EGSE applications based on an object-oriented architecture.

[12] PSS-04-106 "Packet Telemetry Standard, Issue 1, January 1988" and PSS-04-107 "Packet Telecommand Standard, Issue 2, April 1992" (http://esapub.esrin.esa.it/pss/pss-ct03.htm).

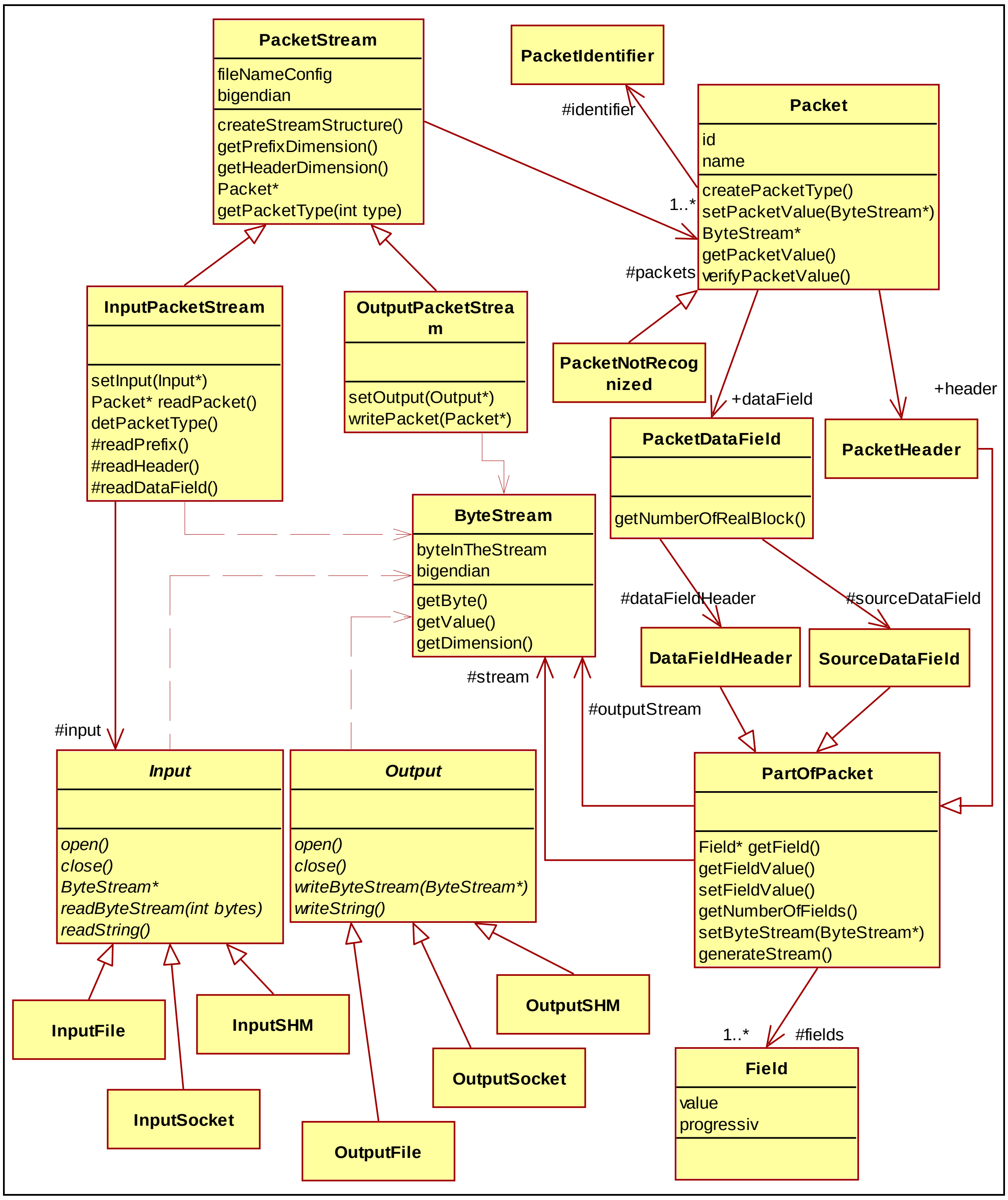


Figure 12 – The PacketLib classes

The classes of this library give the programmer an abstract representation of the packets (see Figure 12), further characterized – at this time – into three different typologies. The next layer (see Figure 13) can then, freed from the byte stream issues, provide the telemetry management functions required by the application level.

The ProcessorLib [5] is the first example of application based on the framework given by PacketLib. It contemplates as input three packet sources (file, socket, and shared memory – see again the outline shown in Figure 2), that are then considered as providers at the following packet

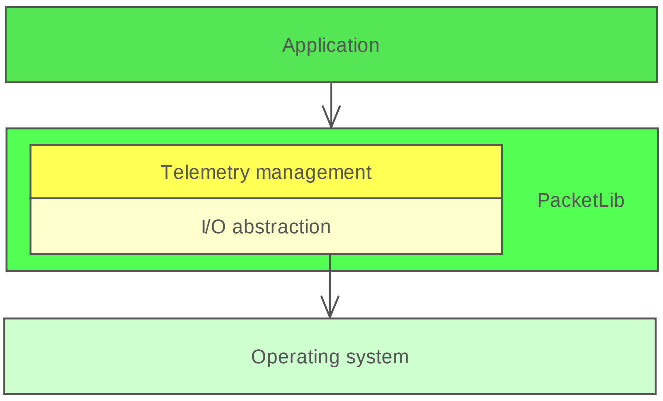


Figure 13 – Abstraction layers provided by PacketLib

processor stage. The selection of the input packet stream used for each specific processor is made on the basis of an associated configuration file (the file "`.processor`" – see Appendix B). The classes defined in the ProcessorLib (see Figure 15) implement, on the whole, both the general purpose functions included in all the processors (part A, in the functional diagram of Figure 14) and all the elements needed to customize them (part B), given the particular instrument and data flow to be processed. However, the ProcessorLib classes does not yet provide for the XML output anticipated in Figure 14.

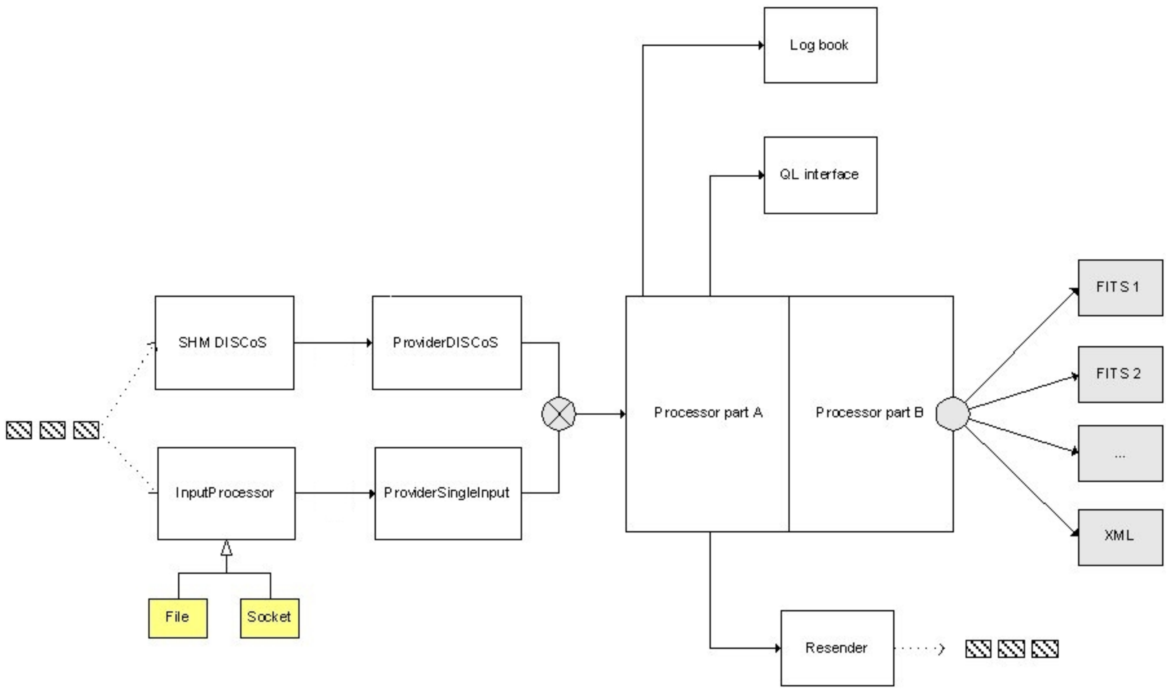


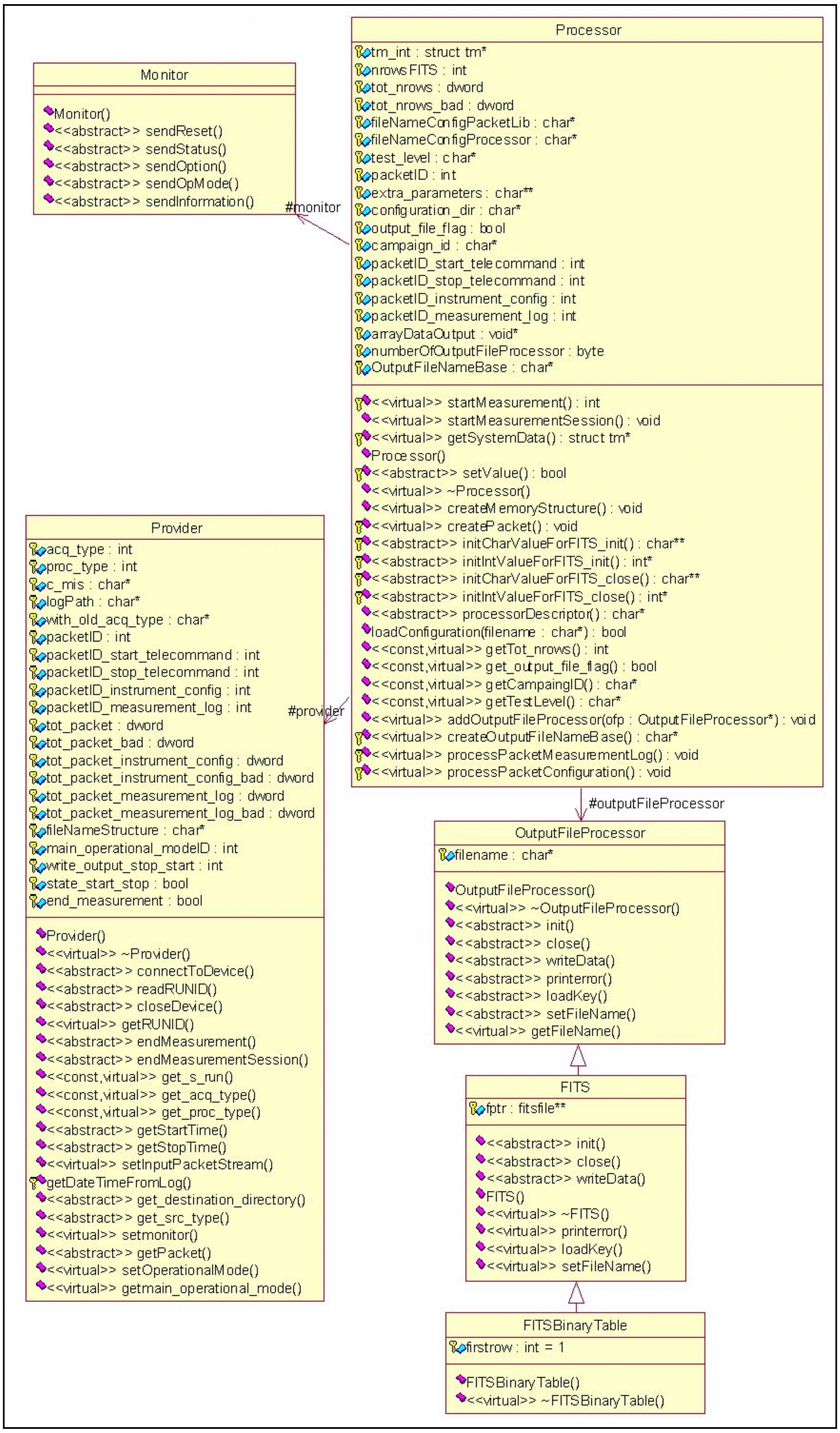
Figure 14 – ProcessorLib: functional scheme

7 Revision of the packet processor module

A practical study has necessarily been carried out taking into account a particular case. We have considered the one denominated HKCALDFETE (i.e. that of the TE related to the Housekeeping [HK] data flow[13] coming from the MiniCalorimeter[14] Detector Front-End), and tried to improve

[13] An HK packet has no scientific meaning: it may contain only pieces of information related to the state of health of a certain instrument or even of the whole spacecraft.

[14] The MiniCalorimeter is one of the four detectors of AGILE [1]; the others are: the Super-Agile instrument, the Silicon Tracker, and the Anticoincidence System.



Monitor
Monitor()
<<abstract>> sendReset()
<<abstract>> sendStatus()
<<abstract>> sendOption()
<<abstract>> sendOpMode()
<<abstract>> sendInformation()
#monitor
Processor
tm_int : struct tm*
nrowsFITS : int
tot_nrows : dword
tot_nrows_bad : dword
fileNameConfigPacketLib : char*
fileNameConfigProcessor : char*
test_level : char*
packetID : int
extra_parameters : char**
configuration_dir : char*
output_file_flag : bool
campaign_id : char*
packetID_start_telecommand : int
packetID_stop_telecommand : int
packetID_instrument_config : int
packetID_measurement_log : int
arrayDataOutput : void*
numberOfOutputFileProcessor : byte
OutputFileNameBase : char*
<<virtual>> startMeasurement() : int
<<virtual>> startMeasurementSession() : void
<<virtual>> getSystemData() : struct tm*
Processor()
<<abstract>> setValue() : bool
<<virtual>> ~Processor()
<<virtual>> createMemoryStructure() : void
<<virtual>> createPacket() : void
<<abstract>> initCharValueForFITS_init() : char**
<<abstract>> initIntValueForFITS_init() : int*
<<abstract>> initCharValueForFITS_close() : char**
<<abstract>> initIntValueForFITS_close() : int*
<<abstract>> processorDescriptor() : char*
loadConfiguration(filename : char*) : bool
<<const,virtual>> getTot_nrows() : int
<<const,virtual>> get_output_file_flag() : bool
<<const,virtual>> getCampaingID() : char*
<<const,virtual>> getTestLevel() : char*
<<virtual>> addOutputFileProcessor(ofp : OutputFileProcessor*) : void
<<virtual>> createOutputFileNameBase() : char*
<<virtual>> processPacketMeasurementLog() : void
<<virtual>> processPacketConfiguration() : void
#provider
Provider
acq_type : int
proc_type : int
c_mis : char*
logPath : char*
with_old_acq_type : char*
packetID : int
packetID_start_telecommand : int
packetID_stop_telecommand : int
packetID_instrument_config : int
packetID_measurement_log : int
tot_packet : dword
tot_packet_bad : dword
tot_packet_instrument_config : dword
tot_packet_instrument_config_bad : dword
tot_packet_measurement_log : dword
tot_packet_measurement_log_bad : dword
fileNameStructure : char*
main_operational_modeID : int
write_output_stop_start : int
state_start_stop : bool
end_measurement : bool
Provider()
<<virtual>> ~Provider()
<<abstract>> connectToDevice()
<<abstract>> readRUNID()
<<abstract>> closeDevice()
<<virtual>> getRUNID()
<<abstract>> endMeasurement()
<<abstract>> endMeasurementSession()
<<const,virtual>> get_s_run()
<<const,virtual>> get_acq_type()
<<const,virtual>> get_proc_type()
<<abstract>> getStartTime()
<<abstract>> getStopTime()
<<virtual>> setInputPacketStream()
getDateTimeFromLog()
<<abstract>> get_destination_directory()
<<abstract>> get_src_type()
<<virtual>> setmonitor()
<<abstract>> getPacket()
<<virtual>> setOperationalMode()
<<virtual>> getmain_operational_mode()
#outputFileProcessor
OutputFileProcessor
filename : char*
OutputFileProcessor()
<<virtual>> ~OutputFileProcessor()
<<abstract>> init()
<<abstract>> close()
<<abstract>> writeData()
<<abstract>> printerror()
<<abstract>> loadKey()
<<abstract>> setFileName()
<<virtual>> getFileName()
FITS
fptr : fitsfile**
<<abstract>> init()
<<abstract>> close()
<<abstract>> writeData()
FITS()
<<virtual>> ~FITS()
<<virtual>> printerror()
<<virtual>> loadKey()
<<virtual>> setFileName()
FITSBinaryTable
firstrow : int = 1
FITSBinaryTable()
<<virtual>> ~FITSBinaryTable()

Figure 15 – The ProcessorLib classes

the previously developed beta version of the associated processor (built onto the ProcessorLib library seen before), so to include the XML input and output management.

We have also considered – as a planning choice – that “the end *does* justify the means”. So we have preferred to develop a simpler source code rather than a more general and complex one: that is to say that not only the prototype will be bound to a specific processor instance, but that it will not even consider the use of parameters; and also, for instance, that there will not be a special new class created ad-hoc to contain the FITSML-related methods.

That said, some improvements – at the source code level – have nevertheless been made. First of all, the `MCAL_HK_PACKET` structure (representing the considered HK packet) has been simplified, so to reduce the required counters to one only, and make therefore the packet data acquisition routine more efficient (in the `setValue()` method) and the whole `writeData()` method much more compact (see the C++ classes source code, in Appendix A).

Then, the arrays used to implement the communication channel between the two classes (the one that inherits from the `FITSBinaryTable` class and the one that is built onto the `Processor` class) has been totally redrawn, with the inclusion of all the packet fields names. These names are now required for the XML output, and are indirectly taken from the packet descriptors database by way of a previously properly initialized "`.packet`" configuration file[15] (see the initialization pattern in Figure 2).

Finally, it has been incidentally improved the overall fault-tolerance, with a more extensive exception handling application: also all the FITSML-related output routines, for instance, have been surrounded by a `try-catch` statement.

The FITSML-related methods (see Figure 16) have been built in order to encapsulate only atomic events, so to be as easily as possible identified, used, and also, in the future, recycled. The `HKCALDFETE_FITSML` class (at the beginning merely derived from the previous `HKCALDFETE_FITS` class) still inherits from the `FITSBinaryTable` class, since that class is not really bound – in spite of its name – to the FITS format, but it is just something more than its `OutputFileProcessor` parent class.

```
int FITSMLoutCreateHead(char* FITSMLnameAttr);
int FITSMLoutArrayTagOpen(char* FITSMLarrayNameAttr);
int FITSMLoutFieldAxisAndReadTags(char** fName, char** units, char** dForm,
                                  char** almLo, char** wrnLo, char** wrnHi, char** almHi);
int FITSMLoutDataTagOpen();
int FITSMLoutRowOpen(long int time32bitValue);
int FITSMLoutTimeColumn(double timeValue);
int FITSMLoutColumn(word paramValue);
int FITSMLoutRowClose();
int FITSMLoutDataTagClose();
int FITSMLoutArrayNotesTag(ushort ntot, char** label, char** nota);
int FITSMLoutArrayObservationTag(char** tele, char** instr, char** origin);
int FITSMLoutArrayTagClose();
int FITSMLoutRootObservationTag(char* startDate, char* endDate,
                                char* startTime, char* endTime);
int FITSMLoutTailClose();
```

Figure 16 – FITSML-related methods prototypes

To implement the XML queries (to the packet descriptors database), we have chosen two open-source C++ APIs coming from the “Apache XML Project” , called Xalan-C (version 1.4.0) and Xerces-C (version 2.1.0). At present, these libraries look as the best solution – at least on a Linux platform – as for parsing, validation and transformation of XML documents.

[15] This configuration file is required by the PacketLib, and can thus be easily consulted using its methods and properties.

To utilize the Xalan/Xerces classes and methods one needs, in the first place, to include all the required header files (see the long list of "`#include`" directives at the beginning of the "`HKCAL-DFETE_FITSML.cpp`" file, in Appendix A); and must then insert the proper static initializers and terminators, preferably once (typically in the class constructor and destructor, respectively), to initialize and terminate (together with the class private members) the Xalan/Xerces platforms.

```
int evalXPathExprInXMLFileContextAs(
                       char* strResult,
                       const char* xml_filename,
                       const char* context,
                       const char* xpathExpr);
int eval4XPathExprOnTheSameContext(
                       char* SR0, char* SR1, char* SR2, char* SR3,
                       const char* xml_filename,
                       const char* context,
                       const char* XP0, const char* XP1, const char* XP2, const char* XP3);
```

Figure 17 – XPathEvaluator methods prototypes

The two methods that we have implemented using the Xalan/Xerces APIs (with the aim to get the required pieces of information from the packet descriptors database) are the ones shown in Figure 17. They both provide the wanted data (as a characters string) given the database file name, the XPath expression, and the context to evaluate it. But the second allows up to four queries to be issued on the same context at the same time, so to obtain – with a very little effort (a mere replication of source code lines) – an important performance improvement. Actually, a lot of time must be spent verifying the context consistency, with an investigation that is as much more time consuming as bigger the database to scan is; and the packet descriptors database easily grows very quickly. A still better optimization, in a more general case, could likely be obtained by trying to execute the required parsing only once per XML document (generating an appropriate `Xalan-ParsedSource` object) and then using just its binary representation.

As for the inherited methods (that is to say, `init()`, `writeData()`, and `close()` – see Figure 18), they have been entirely redesigned.

The `init()` method (formerly devoted to produce the Standard Header and the Binary Table Extension of the FITS file – see Figure 6) is now used to generate the first portion of the new "`.xml`" file with the usual opening XML processing instructions, the root tag and, inside the `<array>` node, the `<fieldAxis>` element along with the related `<read>` item. To compile the table of the parameter descriptive data (within the `<fieldAxis>` element), a call to the above-mentioned method (the one that uses the Xalan/Xerces APIs) has been iteratively used, building each time the correct context string — thus taking special care to indicate the right *namespace* prefixes as well.

The `writeData()` method fills, as before, the real data – inclusive of the associated timestamps – in the output file. But its code is now more compact, thanks to the only loop needed to scan the simplified structure used (see the "`typedef`" declaration at the beginning of "`HKCALDFETEProc-essor.h`", in Appendix A).

The `close()` method, finally, is now much more significant than before, also because of the changes compulsorily introduced by the GSFC XML Group in the FITSML format, contextually to the recent adoption of the XML-Schema in place of the simpler DTD grammar formerly used. The `<redefine>` mechanism used in XML-Schema to implement the inheritance [7] is in fact much more binding than the *parametric entities* mechanism used by DTD; so that in FITSML (following the XML-Schema specifications) the header fields must now unavoidably be put, instead of in the opening, in the closing of the file[16]. Notice, at last, that here the most part of the data comes from the "`.processor`" configuration file — as one can see comparing the sample file shown in Appendix B with the method source code in Appendix A.

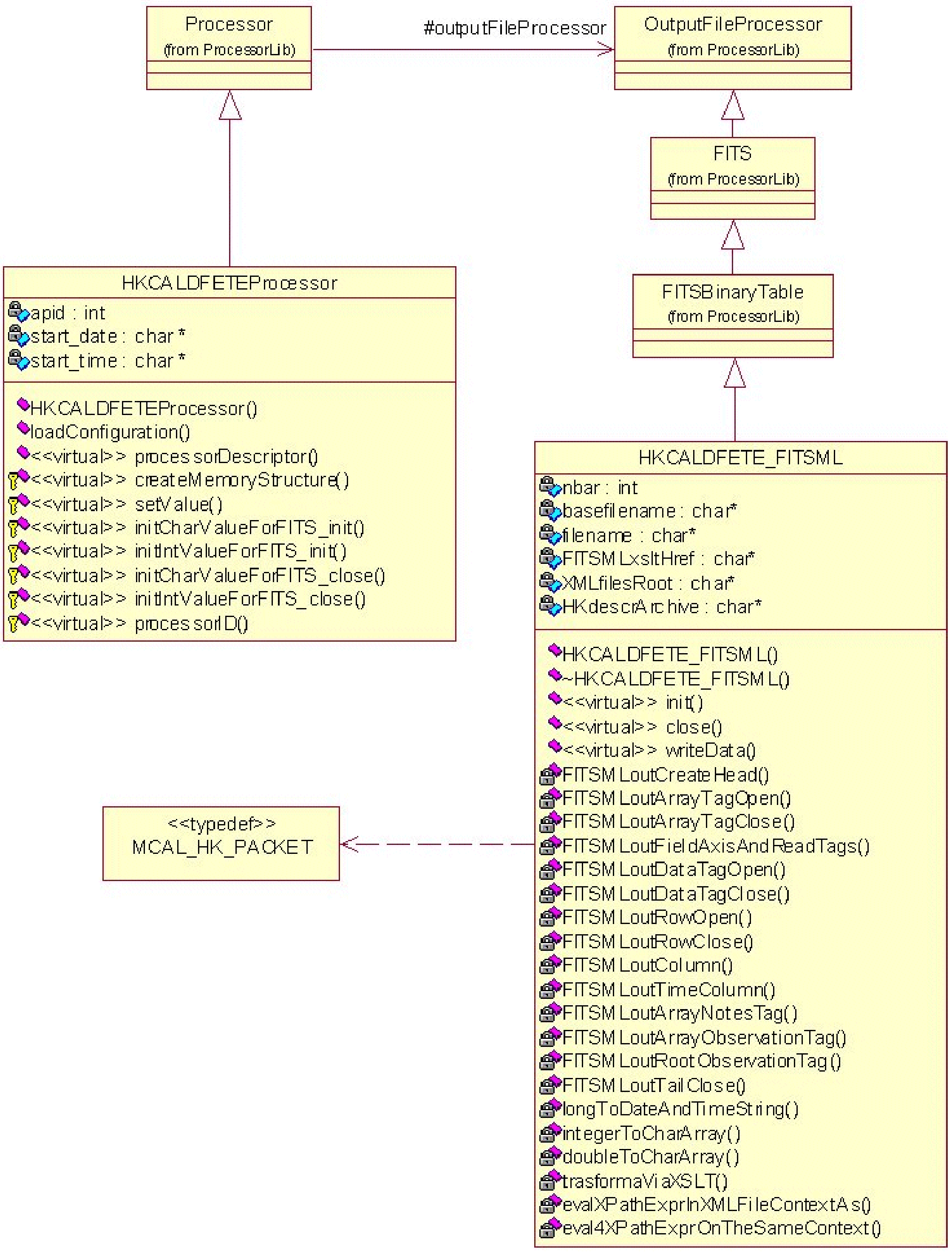
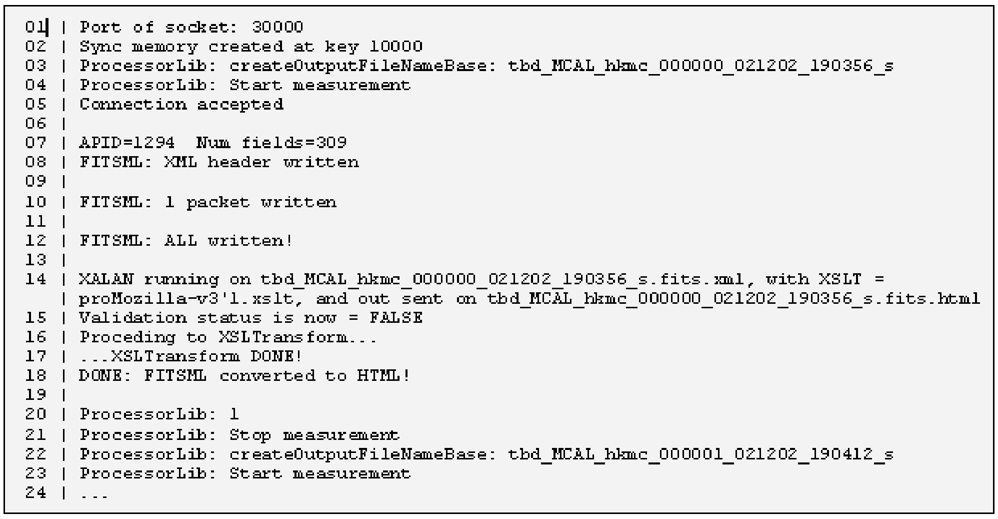


Figure 18 – The HKCALDFETE classes

[16] For the same reason an upgrade of all the Perl and Java APIs developed by the GSFC XML Group will be accomplished soon.

The dump of the console output, taken from a test session based on a socket input stream (see again Figure 2), is shown in Figure 19. This print-out reveals, at line 14, the presence of a call to another (experimental) method we have built using the Xalan platform, called `transform-ViaXSLT()`, that clearly takes an XSLT stylesheet to transform the processor output into an HTML document – and thus tries to achieve the already mentioned QuickLook function. That method could also be used to validate the XML output, but – apart from a preliminary test – we have chosen to disable this feature, so to avoid an ineffective overhead. The FITSML output, any-way, has been externally tested as *well-formed* and *valid* by means of several (also commercial) tools.



```
01 | Port of socket: 30000
02 | Sync memory created at key 10000
03 | ProcessorLib: createOutputFileNameBase: tbd_MCAL_hkmc_000000_021202_190356_s
04 | ProcessorLib: Start measurement
05 | Connection accepted
06 |
07 | APID=1294  Num fields=309
08 | FITSML: XML header written
09 |
10 | FITSML: 1 packet written
11 |
12 | FITSML: ALL written!
13 |
14 | XALAN running on tbd_MCAL_hkmc_000000_021202_190356_s.fits.xml, with XSLT =
   | proMozilla-v3'1.xslt, and out sent on tbd_MCAL_hkmc_000000_021202_190356_s.fits.html
15 | Validation status is now = FALSE
16 | Proceding to XSLTransform...
17 | ...XSLTransform DONE!
18 | DONE: FITSML converted to HTML!
19 |
20 | ProcessorLib: 1
21 | ProcessorLib: Stop measurement
22 | ProcessorLib: createOutputFileNameBase: tbd_MCAL_hkmc_000001_021202_190412_s
23 | ProcessorLib: Start measurement
24 | ...
```

Figure 19 – Console output from a test session

8 Conclusions

All the tried out XML technologies have confirmed their effectiveness: the XML-Schema grammar, used to define the new ‘centralized’ packet descriptors database; the XPath queries, used to extract the needed pieces of information from that DB; and the XSLT *stylesheets*, briefly tested to implement the QuickLook module. The XDF and FITSML languages – once revised – have proved to meet the expectations. And also the interface between the Xalan/Xerces APIs and the already developed C++ libraries has given no particular problems.

As for the performance there are several issues to be considered, between which the most significant are those related to the time consuming XPath/XSLT processing and the space wasting XML formatted files. But there are already some tools (usually integrated in a XML Parser) that implement real-time compression algorithms aware of the XML format, with obvious benefits for network transfer and storage of large files. Besides, as regards the XML transformation tasks, there is a new class of *hardware* products that has recently started to emerge, the “XML accelerator”, that practically offloads the XML data-handling tasks from an application server, with a significant improvement in the conversion time.

In any case, it is beyond doubt that the chance to manage the processor output data in an XML format is worth the trouble.

9 References

About the AGILE Mission

[1] M. Tavani, et al., *Science with AGILE*, Proceedings of the 5th Compton Symposium, AIP Conference Proceedings, Vol. 510, p. 476, 2001, M. McConnell ed.

[2] M. Trifoglio, F. Gianotti, J. B. Stephen, A. Bulgarelli, *User Software Requirements for the Host Computer of the AGILE Minicalorimeter CAL-DFE Test Equipment*, AGILE-ITE-SR-003, Te.S.R.E. Report 315/01, August 2001

[3] Bulgarelli, F. Gianotti, M. Trifoglio, *Interface Control Document of the AGILE EGSE Software*, Technical Note IASF Bologna, 2002

[4] Bulgarelli, F. Gianotti, M. Trifoglio, *PacketLib: a C++ library for scientific satellite telemetry applications*, ADASS Conference 2002, Baltimore

[5] Bulgarelli, F. Gianotti, M. Trifoglio, *ProcessorLib Programmers Guide*, AGILE-ITE-SD-001, Te.S.R.E. Report 349/02, September 2002

About XML architectures

[6] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, 06-10-2000, http://www.w3.org/TR/REC-xml

[7] D. C. Fallside, *XML Schema Part 0: Primer*, W3C Recommendation, 02-05-2001, http://www.w3.org/TR/xmlschema-0/

[8] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, *XML Schema Part 1: Structures*, W3C Recommendation, 02-05-2001, http://www.w3.org/TR/xmlschema-1/

[9] P. V. Biron, A. Malhotra, *XML Schema Part 2: Datatypes*, W3C Rec., 02-05-2001, http://www.w3.org/TR/xmlschema-2/

[10] J. Clark, S. De Rose, *XML Path Language (XPath) Version 1.0*, W3C Rec., 16-11-1999, http://www.w3.org/TR/xpath

[11] J. Clark, *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation, 16-11-1999, http://www.w3.org/TR/xslt

[12] E. Shaya, B. Thomas, C. Cheung, *Specifics on a XML Data Format for Scientific Data*, Astronomical Data Analysis Software and Systems X, ASP Conference Series, Vol. 238, pp. 217-220, 2001, Harnden-Primini-Payne Eds.

[13] B. Thomas, E. Shaya, C. Cheung, *Converting FITS into XML: Methods and Advantages*, Astronomical Data Analysis Software and Systems X, ASP Conference Series, Vol. 238, pp. 487-490, 2001, Harnden-Primini-Payne Eds.

Appendix A The C++ classes source code from the demo HKCALDFETE packet processor

```
/***************************************************************************
                          HKCALDFETEProcessor.h  -  description
                          ---------------------
    begin                : Fri Jun 6 2002
    copyright            : (C) 2002 by Andrea Bulgarelli
                           (C) 2002 by Enrico Franceschi (FITSML revision)
 ***************************************************************************/

#ifndef HKCALDFETEProcessor_H_HEADER_INCLUDED_C3772E73
#define HKCALDFETEProcessor_H_HEADER_INCLUDED_C3772E73
#include "Processor.h"
#include "ProcessorDefinition.h"

const int ncols = 1 + 308;  // 1+ nr of params/cols/fields for this instrument (moved here by EF)

typedef struct
{
  double time;   // double only for compatibility with the former format (a longint should be enough!)
  unsigned long internal_counter;         // counter 'inherited' from the pre-existent structure!?!
  word item[ncols-1];

/*word CT[91];            ---no more: structure semplified by EF
  word R1[90];
  word R2[16];
  word BS[8];
  word RH[4];
  word AH[64];
  word S[35];*/
} MCAL_HK_PACKET;


//##ModelId=3C8892D203D6
//##Documentation
//## Class for CAL-DFE-TE processor for MCAL. This processor works on HK event type


class HKCALDFETEProcessor : public Processor
{
  public:
    HKCALDFETEProcessor();

    bool loadConfiguration(char* filename) throw(PacketException*);

    virtual char* processorDescriptor();

  protected:
    virtual void createMemoryStructure();

    virtual bool setValue();

    /** Array of characters strings with the values needed for the initialization of the FITSML file */
    virtual char** initCharValueForFITS_init();

    /** Array of integers with the values needed for the initialization of the FITSML file */
    virtual int* initIntValueForFITS_init();

    /** Array of characters strings with the values needed for the closing of the FITSML file */
    virtual char** initCharValueForFITS_close();

    /** Array of integers with the values needed for the closing of the FITSML file */
    virtual int* initIntValueForFITS_close();

    virtual char* processorID() { return "hkmc"; };

  private:
    int apid;

    char *start_date, *start_time;   //added by EF

};

#endif                            /* HKCALDFETEProcessor_H_HEADER_INCLUDED_C3772E73 */


##########################################################################################
```

```
/***************************************************************************
                          HKCALDFETEProcessor.cpp  -  description
                          -----------------------
    begin                : Fri Jun 6 2002
    copyright            : (C) 2002 by Andrea Bulgarelli
                           (C) 2002 by Enrico Franceschi (FITSML revision)
 ***************************************************************************/

#include "HKCALDFETEProcessor.h"
//#include "HKCALDFETE_FITS.h"
#include "HKCALDFETE_FITSML.h"

HKCALDFETEProcessor::HKCALDFETEProcessor()
{
  //That's very IMPORTANT, don't forget it!!!!!!!!!!!!!!!!!!
  addOutputFileProcessor(new HKCALDFETE_FITSML());
  //addOutputFileProcessor(new HKCALDFETE_FITS());
  apid=0;
}


bool HKCALDFETEProcessor::loadConfiguration(char* filename) throw(PacketException*)
{
  //don't forget to call this method
  Processor::loadConfiguration(filename);
  //add extra code here for the extra parameters
  return true;
}


void HKCALDFETEProcessor::createMemoryStructure()
{
  //1) creates the needed memory structures
  //nb: the number of events contained in a single packet is to be determined at run-time
  MCAL_HK_PACKET* mcal_hk_packet = (MCAL_HK_PACKET*) new MCAL_HK_PACKET;

  //2) important, don't forget it  --before calling setValue()
  arrayDataOutput = (void*) mcal_hk_packet;
}


bool HKCALDFETEProcessor::setValue()
{
  double time_tag;
//int ncols = 1 + 308;      --defined as const in the header file
  int nn;

  MCAL_HK_PACKET* mcal_hk_packet = (MCAL_HK_PACKET*) arrayDataOutput;

  apid=(p->header->getFields(3))->value;

  time_tag = (double)(p->dataField->dataFieldHeader->getFieldValue(4) << 16)
             + (double)p->dataField->dataFieldHeader->getFieldValue(5);
  time_tag += (double)p->dataField->dataFieldHeader->getFieldValue(6);
  mcal_hk_packet->time = time_tag;                    /* time tag: the same for each packet */

  for(nn=0; nn < ncols-1; nn++)
    mcal_hk_packet->item[nn] = (word) p->dataField->sourceDataField->getFieldValue(nn);

  tot_nrows++;
  //IMPORTANT !!!!!!!!  --(still now?!?)
  nrowsFITS =  1;
  return true;
}


char** HKCALDFETEProcessor::initCharValueForFITS_init()             //output array totally changed
{     // so to capture and supply to HKCALDFETE_FITSML all the field names (taken from the .packet)
  char** c;
//char *start_date, *start_time;   --moved as 'private' at class level
//int ncols = 1 + 308;        --defined as const in header file
  int nn;

  provider->getStartTime(start_date, start_time);    // this call is still in this point, considering
                                 // that its out values may ride on the run-time call triggering
  c = new (char*)[ncols+1];
  c[0] = p->getName();               // packet name (...anyway, here it's practically always the same!)
  for(nn=1; nn < ncols; nn++)      // it takes the names of all the fields (already put in the .packet)
    c[nn] = (p->dataField->sourceDataField->getFields(nn-1))->name;
  c[ncols] = 0;

  return c;
}
```

```
int* HKCALDFETEProcessor::initIntValueForFITS_init()
{
  int* i;                               // in the FITSML context, this ARRAY is almost USELESS
                                        // [In fact, i[0]=apid  is needed only for a mere printf() !!]
  i = new int[2];
  i[0] = apid;
  i[1] = 0; //provider->get_acq_type();
  return i;
}


char** HKCALDFETEProcessor::initCharValueForFITS_close()
{
  char** c;
  //char* start_date, *start_time;   --declared as 'private' at class level
  char *stop_date, *stop_time;

  provider->getStopTime(stop_date, stop_time);

  c = new (char*)[6];                     // filled in the same order in which it will be used
  c[0] = processorDescriptor();
  c[1] = start_date;
  c[2] = start_time;
  c[3] = stop_date;
  c[4] = stop_time;
  c[5] = 0;
  return c;
}


int* HKCALDFETEProcessor::initIntValueForFITS_close()
{
  int* i;

  i = new int[5];                         // filled in the same order in which it will be used
  i[0] = provider->get_s_run()->run_id;
  i[1] = apid;
  i[2] = tot_nrows;
  i[3] = tot_nrows_bad;
  i[4] = 0;
  return i;
}


char* HKCALDFETEProcessor::processorDescriptor()
{
  return "IASF Bologna SC HK_CALDFETE_Processor v. 1.0.0";
}


########################################################################################

/***************************************************************************
                          HKCALDFETE_FITSML.h  -  description
                          -------------------
    begin                : Fri Jun 6 2002
    copyright            : (C) 2002 by Enrico Franceschi
 ***************************************************************************/

#ifndef HKCALDFETE_FITSML_H
#define HKCALDFETE_FITSML_H
#include "FITSBinaryTable.h"
#include "ProcessorDefinition.h"
#include "file.h"


class HKCALDFETE_FITSML : public FITSBinaryTable
{
  public:
    HKCALDFETE_FITSML();  /* _INITIALIZATION_ of private members on constructor call (in .cpp file) */
    ~HKCALDFETE_FITSML();
    virtual bool init(char** c, int* i, char* filenameconfig = 0) throw(PacketException*);
    virtual bool close(char** c, int* i) throw (PacketException*);
    virtual bool writeData(void* data, int nrows) throw (PacketException*);
  private:
    int nbar;   //-(probably no more useful, or at least not here!?!)
    char* basefilename;
    char* filename;
    File FITSMLoutFile;
    // *WARNING* The output file format is *NOT* compatible with any
    //           XSLT stylesheet labeled v0.4.17.2'0 to v0.4.17.3'1 !!
    char* FITSMLxsltHref;      // = "proMozilla-v3'1.xslt"; **SEE ABOVE**
    char* XMLfilesRoot;        // = "fits/";                **SEE ABOVE**
    char* HKdescrArchive;      // = "HKdscrDB_3-3-ns.xml";  **SEE ABOVE**
    // Methods which create the XML outfile (to be MOVEd in a purposely created class!!) [out=0 IIF OK]
    int FITSMLoutCreateHead(char* FITSMLnameAttr);
    int FITSMLoutArrayTagOpen(char* FITSMLarrayNameAttr);
    int FITSMLoutArrayTagClose();
    int FITSMLoutFieldAxisAndReadTags(char** fName, char** units, char** dForm,
                                  char** almLo, char** wrnLo, char** wrnHi, char** almHi);
    int FITSMLoutDataTagOpen();
    int FITSMLoutDataTagClose();
    int FITSMLoutRowOpen(long int time32bitValue);
    int FITSMLoutRowClose();
    int FITSMLoutColumn(word paramValue);
    int FITSMLoutTimeColumn(double timeValue);
    int FITSMLoutArrayNotesTag(ushort ntot, char** label, char** nota);
    int FITSMLoutArrayObservationTag(char** tele, char** instr, char** origin);
    int FITSMLoutRootObservationTag(char* startDate, char* endDate, char* startTime, char* endTime);
    int FITSMLoutTailClose();
    // Three utilities used only locally (it would be better to put these utils in the Utility class!!)
    char* longToDateAndTimeString(long int absoluteSeconds);
    char* integerToCharArray(int n);
    char* doubleToCharArray(double x);
    // Interface with Xalan and Xerces for parsing, translation and XPath expressions evaluation.
    int transformViaXSLT(const char* xml_file, const char* xsltfile, const char* xoutfile,
                       bool validation);
    int evalXPathExprInXMLFileContextAs(char* strResult, const char* xml_filename,
                                    const char* context, const char* xpathExpr);
    int eval4XPathExprOnTheSameContext(char* SR0, char* SR1, char* SR2, char* SR3,
                                   const char* xml_filename, const char* context, const char* XP0,
                                   const char* XP1, const char* XP2, const char* XP3);
};

#endif


########################################################################################


/***************************************************************************
                          HKCALDFETE_FITSML.cpp  -  description
                          --------------------
    begin                : Fri Jun 6 2002
    copyright            : (C) 2002 by Enrico Franceschi
 ***************************************************************************/


#include "HKCALDFETE_FITSML.h"
#include "ProcessorDefinition.h"
#include "HKCALDFETEProcessor.h"
#include <stdio.h>
#include <time.h>
//#include <string.h>
//#include <errno.h>
#include "PacketExceptionIO.h"
#include "file.h"

//-----------------------------------------for Xalan and Xerces
#include <Include/PlatformDefinitions.hpp>
//#include <iostream.h>
#include <xercesc/util/PlatformUtils.hpp>
#include <XalanTransformer/XalanTransformer.hpp>
#include <cassert>
#include <xercesc/framework/LocalFileInputSource.hpp>
#include <XalanDOM/XalanDocument.hpp>
#include <XalanDOM/XalanElement.hpp>
#include <XPath/XObject.hpp>
#include <XPath/XPathEvaluator.hpp>
#include <XalanSourceTree/XalanSourceTreeDOMSupport.hpp>
#include <XalanSourceTree/XalanSourceTreeInit.hpp>
#include <XalanSourceTree/XalanSourceTreeParserLiaison.hpp>
#include <iostream>
#include <sstream>
#include <string>
//using namespace std;

//-----------------------------------------legacy definitions
#define NC_TIME         1
#define NC_INT_COUNTER  2
#define NC_MC_SIGNAL    3
#define NPRECOL         2
```

```
HKCALDFETE_FITSML::HKCALDFETE_FITSML() : FITSMLxsltHref("proMozilla-v3'1.xslt"),
                                         XMLfilesRoot("fits/"),
                                         HKdescrArchive("HKdscrDB_3-3-ns.xml")
{
  // Inizializes Xerces, and then Xalan
  XMLPlatformUtils::Initialize();
  XalanTransformer::initialize();
}


HKCALDFETE_FITSML::~HKCALDFETE_FITSML()
{
  // Terminates Xalan, Xerces, and the ICU component (if included)
  XalanTransformer::terminate();
  XMLPlatformUtils::Terminate();
  //XalanTransformer::ICUCleanUp();
}


bool HKCALDFETE_FITSML::init(char** c, int* i, char* filenameconfig) throw (PacketException*)
{
  int status;
  int ii;
//int ncols = 1 + 308;  --defined into HKCALDFETEProcessor.h
//char* basefilename;
//char* filename;

  long nrows = 0;           /* total number of rows to be updated  */  //-(still useful?)
  int apid = i[0];

  char* nameAttrForFITSMLTag;
  char* extName = "AGILE_Binary";          /* EX 'extension' name, NOW: 'array' name */
  char** tname = new (char*)[ncols];
  char** tunit = new (char*)[ncols];
  char** tform = new (char*)[ncols];
  char** alrmLo = new (char*)[ncols];
  char** warnLo = new (char*)[ncols];
  char** warnHi = new (char*)[ncols];
  char** alrmHi = new (char*)[ncols];

  filename = getFileName();                         // filename formerly used for the FITS file
  basefilename = new char[strlen(filename)+1];      // here used only as a basis reference
  memcpy(basefilename,filename,strlen(filename)+1);
  strcat(filename,".xml");                      // filename REALLY used for the current session
  firstrow = 1;  //important ---(still now, really?)

  if(config!=0)
    nameAttrForFITSMLTag = config->output_header_key[5];
  else
    throw new PacketExceptionIO("ERROR: no configuration file set.");

  tname[0] = "timestamp";
  tunit[0] = "<unit>second</unit>";
  tform[0] = "<binaryInteger bits=\"64\" signed=\"no\"/>";
  alrmLo[0] = "";
  warnLo[0] = "";
  warnHi[0] = "";
  alrmHi[0] = "";

  char* HKdescrDBfile = new char[strlen(XMLfilesRoot)+strlen(HKdescrArchive)+1];
  strcpy(HKdescrDBfile,XMLfilesRoot);
  strcat(HKdescrDBfile,HKdescrArchive);
  string* contestoBase = new string("/hk:HK_DESCRIPTORS_DB/hk:HK_PACKET_DESCRIPTOR[@NAME=\"");
  *contestoBase += c[0];  //"MCAL-DFE Periodic HK";
  *contestoBase += "\"]/hk:HK_ITEM[hk:TM_PARA_DESC=\"";
  int outCode = 0;
  const char* xp0 = "hk:ALARM_LOW";          // <---IMP: take care to the ordering of these strings!!!
  const char* xp1 = "hk:WARNING_LOW";        //           (SEE below, the assignements afterwards done)
  const char* xp2 = "hk:WARNING_HIGH";
  const char* xp3 = "hk:ALARM_HIGH";
  char* sr0;
  char* sr1;                // an array it is not allowed here (as well as before)...
  char* sr2;                // ...because of the need to specifically map each result!
  char* sr3;

  for(ii=1; ii < ncols; ii++)
  {
    tname[ii] = c[ii];
    tunit[ii] = "<unitless/>";
    tform[ii] = "<integer type=\"decimal\" width=\"5\" signed=\"no\"/>";
    string* contesto = new string (*contestoBase);
    *contesto += tname[ii];
    *contesto += "\"]";
    printf("Iteration nr. %2d:\n  Context=%s\n",ii,(char*)contesto->c_str());
    sr0 = new char[6]; sr1 = new char[6]; sr2 = new char[6]; sr3 = new char[6];
    // nb: actually, the maximum value for these (16-bit long) fields can be calculated in advance!

    outCode = eval4XPathExprOnTheSameContext(
                             sr0,sr1,sr2,sr3,HKdescrDBfile,(char*)contesto->c_str(),xp0,xp1,xp2,xp3);
    if (outCode == 0)
    {                                                 // see the ordering above given to the xp-i strings!
      printf("  XPathExpr=%s    -->  Risultato=%s.\n",xp0,(alrmLo[ii] = sr0));
      printf("  XPathExpr=%s    -->  Risultato=%s.\n",xp1,(warnLo[ii] = sr1));
      printf("  XPathExpr=%s    -->  Risultato=%s.\n",xp2,(warnHi[ii] = sr2));
      printf("  XPathExpr=%s    -->  Risultato=%s.\n",xp3,(alrmHi[ii] = sr3));
    }
    else
    {
      printf("  ERROR code %d.\n",outCode);
      throw new PacketException("ERROR in parsing HK-Descriptors DB.");
    }
    //delete sr0;
    //delete sr1;
    //delete sr2;
    //delete sr3;
  }

  printf("\nAPID=%d  Num fields=%d\n",apid,ncols);
  status = 0;

  if ((status = FITSMLoutCreateHead(nameAttrForFITSMLTag)))    /* FITSML file opened only HERE!! */
    printerror(status);      /* call printerror if error occurs */

  if ((status = FITSMLoutArrayTagOpen(extName)))
    printerror(status);

  if ((status = FITSMLoutFieldAxisAndReadTags(tname,tunit,tform,alrmLo,warnLo,warnHi,alrmHi)))
    printerror(status);

  if ((status = FITSMLoutDataTagOpen()))
    printerror(status);

  printf("FITSML: XML header written\n\n");
  return status;
}


//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

bool HKCALDFETE_FITSML::writeData(void* data, int nrows) throw (PacketException*)
{
  int status = 0;
//int ncols = 1 + 308;  --defined into HKCALDFETEProcessor.h
  int nn;

  MCAL_HK_PACKET* mcal_hk_packet = (MCAL_HK_PACKET*) data;

  if(nrows!=1)       // REM: for the MCAL instrument the packet typology is "MONO-block"!!
    throw new PacketExceptionIO("ERROR: HKCALDFETE packet must be of MONO-block type!");
  else
  {
    if ((status = FITSMLoutRowOpen((long int)mcal_hk_packet->time)))
      printerror( status );    // ADDED output, as date & time string, of the 32-bit LSW of the
                               // TIMESTAMP --NEEDED for COMPATIBILITY with XSLT-sheet v3'1 !!
    if ((status = FITSMLoutTimeColumn(mcal_hk_packet->time))) /* writing TIME TAG (in seconds) */
      printerror( status );

    for(nn=0; nn < ncols-1; nn++)
      if ((status = FITSMLoutColumn(mcal_hk_packet->item[nn])))
        printerror( status );

    if ((status = FITSMLoutRowClose()))
      printerror( status );
  }

  /* update the firstrow index for the next time this routine will be called */
  firstrow = firstrow + nrows;                        //---(still useful?)

  printf("FITSML: %d packet written\n",nrows);
  return status;
}


//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§
```

```
bool HKCALDFETE_FITSML::close(char** c, int* i) throw (PacketException*)
{
  int status = 0;

  const unsigned char notesNumber = 12;
  char* noteMarks[notesNumber] = {"CAMPAIGN","TESTLEVE","FILENAME","MODEL"  ,
                                  "DETNAM"  ,"HOSTCOMP","DATATYPE","RUNID"  ,
                                  "APID"    ,"TRIGGERS","DISCARD" ,"COMMENT"};
  char* notes[notesNumber];
  char* telescope[2] = {"TELESCOP",""};
  char* instrument[2] = {"INSTRUME",""};
  char* history[2] = {"ORIGIN",""};
  char* startDate;
  char* endDate;
  char* startTime;
  char* endTime;

//char* filename;
  char* dummyComment ="Any comment here!";

  /* the 19 required values are defined here (with a not-so-easy manual matching with the array items!) */

  if(config!=0)    // Here the .processor file is temporarily forced to contain, in
  {                //  the [FITS key]section, at least 5 keys (which then must meet
                   //  the sequence: TELESCOP, MODEL, DETNAM, HOSTCOMP, DATATYPE).
    notes[0] = config->campaign_id;              /* CAMPAIGN */
    notes[1] = config->test_level;               /* TESTLEVE */
    notes[2] = filename;                         /* FILENAME */
    notes[3] = config->output_header_key[3];     /* MODEL */
    if(strlen(notes[3]) == 0)
      throw new PacketExceptionFileFormat("ERROR: no data value found in [FITS key] section.");
    notes[4] = config->output_header_key[5];     /* DETNAM */
    if(strlen(notes[4]) == 0)
      throw new PacketExceptionFileFormat("ERROR: no data value found in [FITS key] section.");
    notes[5] = config->output_header_key[7];     /* HOSTCOMP */
    if(strlen(notes[5]) == 0)
      throw new PacketExceptionFileFormat("ERROR: no data value found in [FITS key] section.");
    notes[6] = config->output_header_key[9];     /* DATATYPE */
    if(strlen(notes[6]) == 0)
      throw new PacketExceptionFileFormat("ERROR: no data value found in [FITS key] section.");
    notes[7] = integerToCharArray(i[0]);         /* RUNID */
    notes[8] = integerToCharArray(i[1]);         /* APID */
    notes[9] = integerToCharArray(i[2]);         /* TRIGGERS = "Total number of triggers" */
    notes[10] = integerToCharArray(i[3]);        /* DISCARD = "Total number of discarded..." */
    notes[11] = dummyComment;                    /* COMMENT */
    telescope[1] = config->output_header_key[1]; /* TELESCOP */
    if(strlen(telescope[1]) == 0)
      throw new PacketExceptionFileFormat("ERROR: no data value found in [FITS key] section.");
    instrument[1] = config->instrument;          /* INSTRUME */
    history[1] = c[0];                           /* ORIGIN */
    startDate = c[1];
    startTime = c[2];
    endDate = c[3];
    endTime = c[4];
  }
  else
    throw new PacketExceptionIO("ERROR: no configuration file set.");

  /* with the data above gathered one can now fill all the appropriate fields of the XML output */

  if ((status = FITSMLoutDataTagClose()))
    printerror( status );

  if ((status = FITSMLoutArrayNotesTag(notesNumber,noteMarks,notes)))
    printerror( status );

  if ((status = FITSMLoutArrayObservationTag(telescope,instrument,history)))
    printerror( status );

  if ((status = FITSMLoutArrayTagClose()))
    printerror( status );

  if ((status = FITSMLoutRootObservationTag(startDate,endDate,startTime,endTime)))
    printerror( status );

  if ((status = FITSMLoutTailClose()))                      /* FITSML file really closed ONLY here!! */
    printerror( status );

  printf("\nFITSML: ALL written!\n\n");

//char* basefilename;
  char* htmlfilename = new char[strlen(basefilename)+1+5];
  memcpy(htmlfilename,basefilename,strlen(basefilename)+1);
  strcat(htmlfilename,".html");
//char* FITSMLxsltHref = "....";   ---already defined by the class constructor
//char* XMLfilesRoot = "....";   ---already defined by the class constructor
  char* xsltfilename;
  strcpy(xsltfilename,XMLfilesRoot);
  strcat(xsltfilename,FITSMLxsltHref);

  printf("XALAN running on %s, with XSLT = %s, and out sent on %s\n",filename,xsltfilename,htmlfilename);
  //NB- Actually, the def of "xsltfilename" is redundant: a simple call without it can be used instead!
  transformViaXSLT(filename,xsltfilename,htmlfilename,false);
  printf("\nDONE: FITSML converted to HTML!\n\n");  // no validation (here not so useful => overhead!!)

  return status;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutCreateHead(char* FITSMLnameAttr)
{
//File FITSMLoutFile;
  int st = 0;

  char* FITSMLxmlHeader = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n";
  char* FITSMLxsltPrefix = "<?xml-stylesheet type=\"text/xsl\" href=\"";
//char* FITSMLxsltHref = "....";   --- defined by the class costructor
  char* FITSMLxsltPostfix = "\" title=\"AllTableSheet\"?>\n";
  char* FITSMLxmlnsRef = "<FITSML xmlns=\"http://xml.gsfc.nasa.gov/XDF\" \n";
  char* FITSMLxsiRef = "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" \n";
  char* FITSMLschemaLocation = "xsi:schemaLocation=\"http://xml.gsfc.nasa.gov/XDF FITSML_004_SI.xsd\" \n";
  char* FITSMLdescriptionAttr = "Packet Processor Output";
  char* FITSMLtypeAttr = "HK";

  try {
    FITSMLoutFile.open(filename, "w");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " opening FITSML file in FITSMLoutCreateHead()." << endl;
  }

  try {
    FITSMLoutFile.writeString(FITSMLxmlHeader);
    FITSMLoutFile.writeString(FITSMLxsltPrefix);
    FITSMLoutFile.writeString(FITSMLxsltHref);
    FITSMLoutFile.writeString(FITSMLxsltPostfix);
    FITSMLoutFile.writeString(FITSMLxmlnsRef);
    FITSMLoutFile.writeString(FITSMLxsiRef);
    FITSMLoutFile.writeString(FITSMLschemaLocation);
    FITSMLoutFile.writeString("name=\"");
    FITSMLoutFile.writeString(FITSMLnameAttr);
    FITSMLoutFile.writeString("\" description=\"");
    FITSMLoutFile.writeString(FITSMLdescriptionAttr);
    FITSMLoutFile.writeString("\" type=\"");
    FITSMLoutFile.writeString(FITSMLtypeAttr);
    FITSMLoutFile.writeString("\">\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutCreateHead()." << endl;
  }

  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutArrayTagOpen(char* FITSMLarrayNameAttr)
{
//File FITSMLoutFile;
  int st = 0;

  try {
    FITSMLoutFile.writeString("  <array name=\"");
    FITSMLoutFile.writeString(FITSMLarrayNameAttr);
    FITSMLoutFile.writeString("\">\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutArrayTagOpen()." << endl;
  }

  return st;
}
```

```
//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutArrayTagClose()
{
//File FITSMLoutFile;
  int st = 0;

  char* mainComment1 = "    XML archive (with not yet calibrated data) in output";
  char* mainComment2 = " from the Processor, arranged following FISTML v0.04.";

  try {
    FITSMLoutFile.writeString("  </array>\n");
    FITSMLoutFile.writeString("  <note mark=\"COMMENT\">\n");
    FITSMLoutFile.writeString(mainComment1);
    FITSMLoutFile.writeString(mainComment2);
    FITSMLoutFile.writeString("\n  </note>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutArrayTagClose()." << endl;
  }

  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutFieldAxisAndReadTags(char** fName, char** units, char** dForm,
                                       char** almLo, char** wrnLo, char** wrnHi, char** almHi)
{
//File FITSMLoutFile;
  int st = 0;

//int ncols = 1 + 308;      --defined into HKCALDFETEProcessor.h
  int pp = 0;
  char* FITSMLfieldAxisId = "extendedParameterRecord";
  char* FITSMLalmLoNoteTagOpen = "<note mark=\"alarmLow\">";
  char* FITSMLwrnLoNoteTagOpen = "<note mark=\"warningLow\">";
  char* FITSMLwrnHiNoteTagOpen = "<note mark=\"warningHigh\">";
  char* FITSMLalmHiNoteTagOpen = "<note mark=\"alarmHigh\">";

  try {
    FITSMLoutFile.writeString("    <fieldAxis axisId=\"");
    FITSMLoutFile.writeString(FITSMLfieldAxisId);
    FITSMLoutFile.writeString("\">\n");

    FITSMLoutFile.writeString("      <field name=\"");
    FITSMLoutFile.writeString(fName[pp]);
    FITSMLoutFile.writeString("\">\n");
    FITSMLoutFile.writeString("        <units>\n");
    FITSMLoutFile.writeString("          ");
    FITSMLoutFile.writeString(units[pp]);
    FITSMLoutFile.writeString("\n        </units>\n");
    FITSMLoutFile.writeString("        <dataFormat>\n");
    FITSMLoutFile.writeString("          ");
    FITSMLoutFile.writeString(dForm[pp]);
    FITSMLoutFile.writeString("\n        </dataFormat>\n");
    FITSMLoutFile.writeString("      </field>\n");

    for(pp=1; pp < ncols; pp++)
    {
      FITSMLoutFile.writeString("      <field name=\"");
      FITSMLoutFile.writeString(  fName[pp]);
      FITSMLoutFile.writeString("\">\n");
      FITSMLoutFile.writeString("        <units>\n");
      FITSMLoutFile.writeString("          ");
      FITSMLoutFile.writeString(units[pp]);
      FITSMLoutFile.writeString("\n        </units>\n");
      FITSMLoutFile.writeString("        <dataFormat>\n");
      FITSMLoutFile.writeString("          ");
      FITSMLoutFile.writeString(dForm[pp]);
      FITSMLoutFile.writeString("\n        </dataFormat>\n");

      FITSMLoutFile.writeString("        ");
      FITSMLoutFile.writeString(FITSMLalmLoNoteTagOpen);
      FITSMLoutFile.writeString(almLo[pp]);
      FITSMLoutFile.writeString("</note>\n");
      FITSMLoutFile.writeString("        ");
      FITSMLoutFile.writeString(FITSMLwrnLoNoteTagOpen);
      FITSMLoutFile.writeString(wrnLo[pp]);
      FITSMLoutFile.writeString("</note>\n");
      FITSMLoutFile.writeString("        ");
      FITSMLoutFile.writeString(FITSMLwrnHiNoteTagOpen);
      FITSMLoutFile.writeString(wrnHi[pp]);
      FITSMLoutFile.writeString("</note>\n");
      FITSMLoutFile.writeString("        ");
      FITSMLoutFile.writeString(FITSMLalmHiNoteTagOpen);
      FITSMLoutFile.writeString(almHi[pp]);
      FITSMLoutFile.writeString("</note>\n");

      FITSMLoutFile.writeString("      </field>\n");
    }

    FITSMLoutFile.writeString("    </fieldAxis>\n");
    FITSMLoutFile.writeString("    <read>\n");
    FITSMLoutFile.writeString("      <tagToAxis tag=\"d0\" axisIdRef=\"");
    FITSMLoutFile.writeString(FITSMLfieldAxisId);
    FITSMLoutFile.writeString("\"/>\n");
    FITSMLoutFile.writeString("    </read>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutFieldAxisAndReadTags()." << endl;
  }

  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutDataTagOpen()
{
//File FITSMLoutFile;
  int st = 0;

  try {
    FITSMLoutFile.writeString("    <data>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutDataTagOpen()." << endl;
  }

  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutDataTagClose()
{
//File FITSMLoutFile;
  int st = 0;

  try {
    FITSMLoutFile.writeString("    </data>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutDataTagClose()." << endl;
  }

  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutRowOpen(long int time32bitValue)
{
//File FITSMLoutFile;
  int st = 0;

  try {
    FITSMLoutFile.writeString("      <d0>\n");
    FITSMLoutFile.writeString("      ");     // line NEEDED for COMPATIBILITY with XSLT-sheet v3'1 !!
    FITSMLoutFile.writeString(longToDateAndTimeString(time32bitValue));
    FITSMLoutFile.writeString("\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutRowOpen()." << endl;
  }

  return st;
}
```

```
//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutRowClose()
{
//File FITSMLoutFile;
  int st = 0;

  try {
    FITSMLoutFile.writeString("      </d0>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutRowClose()." << endl;
  }
  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutTimeColumn(double timeValue)
{
//File FITSMLoutFile;
  int st = 0;

  try {
    FITSMLoutFile.writeString("        <d1>");
    FITSMLoutFile.writeString(doubleToCharArray(timeValue));
    FITSMLoutFile.writeString("</d1>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutTimeColumn()." << endl;
  }
  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutColumn(word paramValue)
{
//File FITSMLoutFile;
  int st = 0;

  try {
    FITSMLoutFile.writeString("        <d1>");
    FITSMLoutFile.writeString(integerToCharArray(paramValue));
    FITSMLoutFile.writeString("</d1>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutColumn()." << endl;
  }
  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutArrayNotesTag(ushort ntot, char** label, char** nota)
{
//File FITSMLoutFile;
  int st = 0;
  ushort nn;

  try {
    FITSMLoutFile.writeString("    <notes>\n");
    for(nn=0; nn < ntot; nn++)
    {
      FITSMLoutFile.writeString("      <note mark=\"");
      FITSMLoutFile.writeString(label[nn]);
      FITSMLoutFile.writeString("\">");
      FITSMLoutFile.writeString(nota[nn]);
      FITSMLoutFile.writeString("</note>\n");
    }
    FITSMLoutFile.writeString("    </notes>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutArrayNotesTag()." << endl;
  }
  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§
```

```
int HKCALDFETE_FITSML::FITSMLoutArrayObservationTag(char** tele, char** instr, char** origin)
{
//File FITSMLoutFile;
  int st = 0;
  const unsigned char label = 0;
  const unsigned char content = 1;

  try {
    FITSMLoutFile.writeString("    <observation>\n");

    FITSMLoutFile.writeString("      <telescope keyword=\"");
    FITSMLoutFile.writeString(tele[label]);
    FITSMLoutFile.writeString("\" description=\"Data acquisition telescope\">");
    FITSMLoutFile.writeString(tele[content]);
    FITSMLoutFile.writeString("</telescope>\n");

    FITSMLoutFile.writeString("      <instrument keyword=\"");
    FITSMLoutFile.writeString(instr[label]);
    FITSMLoutFile.writeString("\" description=\"Data acquisition instrument\">");
    FITSMLoutFile.writeString(instr[content]);
    FITSMLoutFile.writeString("</instrument>\n");

    FITSMLoutFile.writeString("      <history description=\"History of how the data contained in the array
were processed.\">\n");
    FITSMLoutFile.writeString("        <origin keyword=\"");
    FITSMLoutFile.writeString(origin[label]);
    FITSMLoutFile.writeString("\" description=\"Installation where the FITS file is being written\">");
    FITSMLoutFile.writeString(origin[content]);
    FITSMLoutFile.writeString("</origin>\n");
    FITSMLoutFile.writeString("      </history>\n");

    FITSMLoutFile.writeString("    </observation>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutArrayObservationTag()." << endl;
  }
  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutRootObservationTag(char* startDate, char* endDate,
                                                   char* startTime, char* endTime)
{
//File FITSMLoutFile;
  int st = 0;

  try {
    FITSMLoutFile.writeString("  <observation>\n");
    FITSMLoutFile.writeString("    <datesAndTimes>\n");

    FITSMLoutFile.writeString("      <observationDate keyword=\"DATE-OBS\" description=\"Date of observation
(UT recommended);(\'yyyy-mm-dd\').\">");
    FITSMLoutFile.writeString(startDate);
    FITSMLoutFile.writeString("</observationDate>\n");

    FITSMLoutFile.writeString("      <observationDate keyword=\"DATE-END\" description=\"Date of observation
(UT recommended);(\'yyyy-mm-dd\').\">");
    FITSMLoutFile.writeString(endDate);
    FITSMLoutFile.writeString("</observationDate>\n");

    FITSMLoutFile.writeString("      <observationTime keyword=\"TIME-OBS\" description=\"Time of observation
(UT recommended);(\'hh:mm:ss\').\" type=\"start\">");
    FITSMLoutFile.writeString(startTime);
    FITSMLoutFile.writeString("</observationTime>\n");

    FITSMLoutFile.writeString("      <observationTime keyword=\"TIME-OBS\" description=\"Time of observation
(UT recommended);(\'hh:mm:ss\').\" type=\"end\">");
    FITSMLoutFile.writeString(endTime);
    FITSMLoutFile.writeString("</observationTime>\n");

    FITSMLoutFile.writeString("    </datesAndTimes>\n");
    FITSMLoutFile.writeString("  </observation>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutRootObservationTag()." << endl;
  }
  return st;
}
```

```
//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::FITSMLoutTailClose()
{
//File FITSMLoutFile;
  int st = 0;

  try {
    FITSMLoutFile.writeString("</FITSML>\n");
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " writing on FITSML file in FITSMLoutTailClose()." << endl;
  }

  try {
    FITSMLoutFile.close();
  }
  catch(PacketExceptionIO* e) {
    st = atoi(e->geterror());
    cout << "Error code = " << st << " closing FITSML file in FITSMLoutTailClose()." << endl;
  }

  return st;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

char* HKCALDFETE_FITSML::longToDateAndTimeString(long int absoluteSeconds)
{                                                                          // needs: #include <time.h>
  char* ctimeOut;                //01234567890123456789012   (23 bytes, \0 included)
  char myDateAndTimeString[23] = "dd Mmm yyyy - hh:mm:ss";             //layout of the wanted format
                                       //01234567890123456789 0123-45   (26 bytes, \0 included)
  ctimeOut = ctime(&absoluteSeconds); //"Www Mmm dd hh:mm:ss yyyy\n" adjusted to local timezone
  memcpy(myDateAndTimeString+ 0,ctimeOut+ 8,2);   //day mapping considered the format wanted
  memcpy(myDateAndTimeString+ 3,ctimeOut+ 4,3);   //month mapping considered the format wanted
  memcpy(myDateAndTimeString+ 7,ctimeOut+20,4);   //year mapping considered the format wanted
  memcpy(myDateAndTimeString+14,ctimeOut+11,8);   //hh:mm:ss mapping considered the format wanted

  return strcpy(ctimeOut,myDateAndTimeString);  //IMP: else it returns the pointer to a local variable!!
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

char* HKCALDFETE_FITSML::integerToCharArray(int n)
{
    char* c = new char[1];
    sprintf(c, "%d", n);
    return c;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

char* HKCALDFETE_FITSML::doubleToCharArray(double x)
{
    char* c = new char[1];
    sprintf(c, "%.0f", x);
    return c;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::evalXPathExprInXMLFileContextAs(char* strResult, const char* xml_filename, const char*
context, const char* xpathExpr)
{
  int codeRes = 0;   // contains the searched string, given XMLFilePath, Context and XPathExpression
  try         // A better optimization could be obtained... trying to execute the parsing only ONCE...
  {
    //XMLPlatformUtils::Initialize();  --already done at constructor level
    XPathEvaluator::initialize();
    {
      // Initializes the XalanSourceTree subsystem
      XalanSourceTreeInit theSourceTreeInit;
      // Creates the instances and the link between the objects for the subsequent parsing
      XalanSourceTreeDOMSupport theDOMSupport;
      XalanSourceTreeParserLiaison theLiaison(theDOMSupport);
      theDOMSupport.setParserLiaison(&theLiaison);

      // Associates the XML file with an input source
      const LocalFileInputSource fonteXML(c_wstr(XalanDOMString(xml_filename)));
      // Parses the XML document
      XalanDocument* const docXML = theLiaison.parseXMLStream(fonteXML);
      assert(docXML != 0);
      // Istances the engine to evaluate the XPath expressions
      XPathEvaluator theEvaluator;
      // Looks for the desired context
      XalanNode* const theContextNode = theEvaluator.selectSingleNode(theDOMSupport, docXML,
                                                                XalanDOMString(context).c_str(),
                                                                docXML->getDocumentElement());
      if (theContextNode != 0)
      {
        // The searched context EXIST and thus the XPath expression can be evaluated
        const XObjectPtr objResult(theEvaluator.evaluate(theDOMSupport, theContextNode,
                                                    XalanDOMString(xpathExpr).c_str(),
                                                    docXML->getDocumentElement()));
        assert(objResult.null() == false);
        ostringstream dummyOss;
        dummyOss << objResult->str();
        const char* dummyStr = (dummyOss.str()).c_str();
        memcpy(strResult,dummyStr,strlen(dummyStr)+1);
        //cout << "The string value of the result is:" << endl << objResult->str() << endl << endl;
      }
      else   // ATTN: the specified context does NOT exist!!
      {
        printf("WARNING: WRONG CONTEXT for the XPath expression!!!!\n%s\n\n",context);
        codeRes = -1;
      }
    }
    XPathEvaluator::terminate();
    //XMLPlatformUtils::Terminate();  --already done at constructor level
  }
  catch(...)
  {
    printf("WARNING: an exception occurred in the XPath evaluator module!!!\n");
    codeRes = -1;
  }

  return codeRes;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::eval4XPathExprOnTheSameContext(
                                       char* SR0, char* SR1, char* SR2, char* SR3,
                                       const char* xml_filename, const char* context,
                                       const char* XP0, const char* XP1, const char* XP2, const char* XP3)
{  //Simple reiteration of the above procedure (data structures replicated for just memory mapping issues)
  int codeRes = 0;   // Gives 4 strings at the same time so to verify the context only when really neeeded
  try                // In the future it could be tried to parse the whole XML document only once...
  {                                                             // (*maybe* using a XalanParsedSource)
    //XMLPlatformUtils::Initialize();  --already done at constructor level
    XPathEvaluator::initialize();
    {
      // Initializes the XalanSourceTree subsystem
      XalanSourceTreeInit theSourceTreeInit;
      // Creates the instances and the link between the objects for the subsequent parsing
      XalanSourceTreeDOMSupport theDOMSupport;
      XalanSourceTreeParserLiaison theLiaison(theDOMSupport);
      theDOMSupport.setParserLiaison(&theLiaison);
      // Associates the XML file with an input source
      const LocalFileInputSource fonteXML(c_wstr(XalanDOMString(xml_filename)));
      // Parses the XML document
      XalanDocument* const docXML = theLiaison.parseXMLStream(fonteXML);
      assert(docXML != 0);
      // Istances the engine to evaluate the XPath expressions
      XPathEvaluator theEvaluator;
      // Looks for the desired context
      XalanNode* const theContextNode = theEvaluator.selectSingleNode(theDOMSupport, docXML,
                                                                XalanDOMString(context).c_str(),
                                                                docXML->getDocumentElement());
      if (theContextNode != 0)
      {
        // The searched context EXIST and thus the *FOUR* XPath expressions can be evaluated
        const char* dummyStr;  // And the only way to do this is a mere reiteration of the source code!¡!
        const XObjectPtr objResult0(theEvaluator.evaluate(theDOMSupport, theContextNode,
                                                     XalanDOMString(XP0).c_str(),
                                                     docXML->getDocumentElement()));
        assert(objResult0.null() == false);
        ostringstream dummyOss0;
        dummyOss0 << objResult0->str();
        dummyStr = (dummyOss0.str()).c_str();
        memcpy(SR0,dummyStr,strlen(dummyStr)+1);
        const XObjectPtr objResult1(theEvaluator.evaluate(theDOMSupport, theContextNode,
                                                     XalanDOMString(XP1).c_str(),
                                                     docXML->getDocumentElement()));
        assert(objResult1.null() == false);
        ostringstream dummyOss1;
        dummyOss1 << objResult1->str();
        dummyStr = (dummyOss1.str()).c_str();
        memcpy(SR1,dummyStr,strlen(dummyStr)+1);
        const XObjectPtr objResult2(theEvaluator.evaluate(theDOMSupport, theContextNode,
                                                          XalanDOMString(XP2).c_str(),
                                                          docXML->getDocumentElement()));
        assert(objResult2.null() == false);
        ostringstream dummyOss2;
        dummyOss2 << objResult2->str();
        dummyStr = (dummyOss2.str()).c_str();
        memcpy(SR2,dummyStr,strlen(dummyStr)+1);
        const XObjectPtr objResult3(theEvaluator.evaluate(theDOMSupport, theContextNode,
                                                          XalanDOMString(XP3).c_str(),
                                                          docXML->getDocumentElement()));
        assert(objResult3.null() == false);
        ostringstream dummyOss3;
        dummyOss3 << objResult3->str();
        dummyStr = (dummyOss3.str()).c_str();
        memcpy(SR3,dummyStr,strlen(dummyStr)+1);
      }
      else   // ATTN: the specified context does NOT exist!!
      {
        printf("WARNING: WRONG CONTEXT for the XPath expressions!!!!\n%s\n\n",context);
        codeRes = -1;
      }
    }
    XPathEvaluator::terminate();
    //XMLPlatformUtils::Terminate();  --already done at constructor level
  }
  catch(...)
  {
    printf("WARNING: an exception occurred in the XPath evaluator module!!!\n");
    codeRes = -1;
  }

  return codeRes;
}

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

int HKCALDFETE_FITSML::transformViaXSLT(const char* xml_file, const char* xsltfile, const char* xoutfile,
                                        bool validation)
{
  int resCode = -1;

  try
  {
    XalanTransformer XSLT_engine; // = (XalanTransformer) new XalanTransformer();
    XSLT_engine.setUseValidation(validation);
    if (XSLT_engine.getUseValidation())
      printf("Validation status is now = TRUE\n");
    else
      printf("Validation status is now = FALSE\n");
    printf("Proceding to XSLTransform...\n");
    if (xsltfile != "")
      resCode = XSLT_engine.transform(xml_file,xsltfile,xoutfile);
    else
      resCode = XSLT_engine.transform(xml_file,xoutfile);
    printf("...XSLTransform DONE!\n");

    if(resCode != 0)
      printf("Xalan (XSLT transformation) - Error: %s\n",XSLT_engine.getLastError());
  }
  catch(...)
  {
    printf("Xalan (XSLT transformation): UNKNOWN error!\n");
  }

  return resCode;
}

#########################################################################################


/***************************************************************************
                          main.cpp  -  description
                             -------------------
    begin                : Fri Mar  8 11:43:52 CET 2002
    copyright            : (C) 2002 by Andrea Bulgarelli
 ***************************************************************************/
```

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

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

clock_t start;
clock_t end;
time_t timevar1;
time_t timevar2;

char* c1;

void report()
{
  char* c2 = new char[1];
  printf("heap peak: %p\n", c2-c1);
  printf("Time0: %6.4f\n", (double)(clock() - (double)start) / CLOCKS_PER_SEC);
  time(&timevar2);
  cout << "Time: " << timevar2-timevar1 << endl;
  cout << "Number of ByteStream internal created: " << ByteStream::count_object2
       << " mem allocated: " << ByteStream::count_object2 * sizeof(ByteStream) << endl;
  cout << "Number of ByteStream internal deleted: " << ByteStream::count_object_deleted2
       << " mem free: " << ByteStream::count_object_deleted2 * (long)sizeof(ByteStream) << endl;
  cout << "Number of ByteStream external created (memory allocated): " << ByteStream::count_object
       << " mem allocated: " << ByteStream::count_object * sizeof(ByteStream) << endl;
  cout << "Number of ByteStream external deleted (memory allocated): " << ByteStream::count_object_deleted
       << " mem free: " << (long)(ByteStream::count_object_deleted * (long)sizeof(ByteStream)) << endl;
  cout << "Number of ByteStream wrong: " << ByteStream::count_object_wrong <<  endl;
  cout << "Number of byte read by File class: " << File::byte_read <<  endl;
  cout << "Number of char read by File class: " << File::char_read <<  endl;
}

int main(int argc, char *argv[])
{
  try
  {
    c1 = new char[1];
    struct tm* tm_int;
    time_t timevar1;
    time_t timevar2;
    time(&timevar1);
    start = clock();
    HKCALDFETEProcessor* gp = (HKCALDFETEProcessor*) new HKCALDFETEProcessor();
    gp->loadConfiguration("./conf/HK-CAL-DFE-TE_Socket.processor");
    //gp->loadConfiguration(argv[1]);
    gp->startMeasurementSession();

    time(&timevar2);
    cout << "Time: " << timevar2-timevar1 << endl;
    cout << "Media: " << gp->getTot_nrows() /   ((timevar2-timevar1)?(timevar2-timevar1):1) << endl;
    end = clock();
    report();
    return 0;

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

Appendix B A sample ".processor" configuration file (the one used for the demo processor)

```
-- # PacketLib version 1.1.1
[Processor]
-- Configuration file for PacketLib
HK.stream
-- output file flag
true
-- campaign ID
tbd
-- instrument
MCAL
-- test_level
DFE
-- packetID
1
-- packet ID of start telecommand
2
-- packet ID of stop telecommand
3
-- packet ID instrument configuration
none
-- packet ID measurement log
none
-- extra parameters ----------------------------
[Provider]
-- 0 DISCOS, 1 SingleInputFile, 2 SingleInputSocket
2
-- acquisition type (chain) 0 = ACQ_OLD
0
-- directory with log file
none
-- only for playback mode (acq_type=ACQ_OLD) acq_type of current playback mode
tbd
-- 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: playback mode, read from filename
0
-- 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
-- port number
30000
[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
MODEL
April 2002
```

```
--key 3
DETNAM
MCAL-DFE-TE
--key 4
HOSTCOMP
IASF T.E.
--key 5
DATATYPE
Diagnostic
```

Appendix C Excerpts from a sample FITSML file generated by the demo processor

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="proMozilla-v3'1.xslt" title="AllTableSheet"?>
<FITSML xmlns="http://xml.gsfc.nasa.gov/XDF"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xml.gsfc.nasa.gov/XDF FITSML_004_SI.xsd"
name="MCAL-DFE-TE" description="Packet Processor Output" type="HK">
  <array name="AGILE_Binary">
    <fieldAxis axisId="extendedParameterRecord">
      <field name="timestamp">
        <units>
          <unit>second</unit>
        </units>
        <dataFormat>
          <binaryInteger bits="64" signed="no"/>
        </dataFormat>
      </field>
      <field name="Commanded Threshold #01">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">3</note>
        <note mark="warningLow">1</note>
        <note mark="warningHigh">2</note>
        <note mark="alarmHigh">4</note>
      </field>
      <field name="Commanded Threshold #02">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">5</note>
        <note mark="warningLow">7</note>
        <note mark="warningHigh">7</note>
        <note mark="alarmHigh">8</note>
      </field>
      <field name="Commanded Threshold #03">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">10</note>
        <note mark="warningLow">10</note>
        <note mark="warningHigh">11</note>
        <note mark="alarmHigh">12</note>
      </field>
      <field name="Commanded Threshold #04">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #05">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #06">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #07">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #08">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #09">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #10">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #11">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
```

```
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #12">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #13">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #14">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
      <field name="Commanded Threshold #15">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">20</note>
        <note mark="warningLow">30</note>
        <note mark="warningHigh">280</note>
        <note mark="alarmHigh">290</note>
      </field>
[...]
      <field name="Spare word #34">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">1225</note>
        <note mark="warningLow">1226</note>
        <note mark="warningHigh">1227</note>
        <note mark="alarmHigh">1228</note>
      </field>
      <field name="Spare word #35">
        <units>
          <unitless/>
        </units>
        <dataFormat>
          <integer type="decimal" width="5" signed="no"/>
        </dataFormat>
        <note mark="alarmLow">1229</note>
        <note mark="warningLow">1230</note>
        <note mark="warningHigh">1231</note>
        <note mark="alarmHigh">1232</note>
      </field>
    </fieldAxis>
    <read>
      <tagToAxis tag="d0" axisIdRef="extendedParameterRecord"/>
    </read>
    <data>
```

```
      <d0>
      14 Jul 1970 - 22:15:29
        <d1>16834529</d1>
        <d1>0</d1>
        <d1>1</d1>
        <d1>2</d1>
        <d1>3</d1>
        <d1>4</d1>
        <d1>5</d1>
        <d1>6</d1>
        <d1>7</d1>
        <d1>8</d1>
        <d1>9</d1>
        <d1>10</d1>
        <d1>11</d1>
        <d1>12</d1>
        <d1>13</d1>
        <d1>14</d1>
        <d1>15</d1>
        <d1>16</d1>
        <d1>17</d1>
        <d1>18</d1>
        <d1>19</d1>
[...]
        <d1>300</d1>
        <d1>301</d1>
        <d1>302</d1>
        <d1>303</d1>
        <d1>304</d1>
        <d1>305</d1>
        <d1>306</d1>
        <d1>307</d1>
      </d0>
    </data>
    <notes>
      <note mark="CAMPAIGN">tbd</note>
      <note mark="TESTLEVE">DFE</note>
      <note mark="FILENAME">fits/tbd_MCAL_hkmc_000000_021202_200823_s.fits.xml</note>
      <note mark="MODEL">April 2002</note>
      <note mark="DETNAM">MCAL-DFE-TE</note>
      <note mark="HOSTCOMP">IASF T.E.</note>
      <note mark="DATATYPE">Diagnostic</note>
      <note mark="RUNID">0</note>
      <note mark="APID">1294</note>
      <note mark="TRIGGERS">1</note>
      <note mark="DISCARD">0</note>
      <note mark="COMMENT">Any comment here!</note>
    </notes>
    <observation>
      <telescope keyword="TELESCOP" description="Data acquisition telescope">Agile</telescope>
      <instrument keyword="INSTRUME" description="Data acquisition instrument">MCAL</instrument>
      <history description="History of how the data contained in the array were processed.">
        <origin keyword="ORIGIN" description="Installation where the FITS file is being written">IASF
Bologna SC HK_CALDFETE_Processor v. 1.0.0</origin>
      </history>
    </observation>
  </array>
  <note mark="COMMENT">
    XML archive (with not yet calibrated data) in output from the Processor, arranged following FISTML
v0.04.
  </note>
  <observation>
    <datesAndTimes>
      <observationDate keyword="DATE-OBS" description="Date of observation  (UT recommended);('yyyy-mm-
dd').">NULL</observationDate>
      <observationDate keyword="DATE-END" description="Date of observation  (UT recommended);('yyyy-mm-
dd').">NULL</observationDate>
      <observationTime keyword="TIME-OBS" description="Time of observation  (UT
recommended);('hh:mm:ss')." type="start">NULL</observationTime>
      <observationTime keyword="TIME-OBS" description="Time of observation  (UT
recommended);('hh:mm:ss')." type="end">NULL</observationTime>
    </datesAndTimes>
  </observation>
</FITSML>
```