TITLE: # Planck LFI – Scientific Telemetry Decompression Code URD

DOC. TYPE: URD

PROJECT REF.: PL-LFI-OAT-UR–006 **PAGE:** I of IV, 00

ISSUE/REV.: 0 **DATE:** November 2001

Issued by	**M. Maris** **LFI DPC Development Team**	**Date:** **Signature:**	**November 20, 2001** _____
Agreed by	**F. PASIAN** **LFI Program Manager**	**Date:** **Signature:**	**December 12, 2001** _____
Approved by	**R.C. BUTLER** **LFI Program Manager**	**Date:** **Signature:**	 _____
Approved by	**N. MANDOLESI** **LFI Principal Investigator**	**Date:** **Signature:**	 _____

DISTRIBUTION LIST

Recipient	Company / Institute	E-mail address	Sent

CHANGE RECORD

Issue	Date	Sheet	Description of Change	Release

TABLE OF CONTENTS

1 SCOPE

This documents reports the user requirements for the scientific telemetry decompression code for the LFI pipeline to be operated at DPC.

It relies mainly with the definition of the user interface to allow a fast and simple integration in the DPC pipeline.

1.1 LIMITS OF APPLICABILITY

It is assumed that the decompression code will be executed in the framework of the Planck/LFI pipeline operated by the DPC.

2 APPLICABLE/REFERENCE DOCUMENTS

2.1 APPLICABLE DOCUMENTS

[1] *Requirements on the LFI On-Board Compression*
M. Maris, S. Fogliani
2000-11-07, PL-LFI-OAT-TN-15

[2] *ADAPTIVE ARITHMETIC COMPRESSOR ADAPTATION TO SPU SOFTWARE ARCHITECTURE, DSP & VIRTUOSO ENVIRONMENT*
Thomas Fenal
PL-LFI-IAC-TN-005

2.2 REFERENCE DOCUMENTS

[3] *SELECTION OF ADEQUATE PARAMETERS AND MODIFICATION OF THE ADAPTIVE ARITHMETIC CODER IN ORDER TO COMPLY WITH LFI COMPRESSION REQUIREMENTS*
Thomas Fenal
PL-LFI-IAC-TN-004

2.3 ACRONYMS LIST

DEM	DEcompression Module
DMM	River Main Module
DPC	Data Processing Center
ESM	Environment Service Module
LFI	Low Frequency Instrument
URD	User Requirements Document

3 DECOMPRESSOR SOFTWARE ARCHITECTURE REQUIREMENTS

This section describes the application of a general-purpose architecture, which allows to separate the work of two teams: the development team and the integration team, minimising the amount of documentation flowing between the two teams. In addition, this architecture allows to integrate barely packages in the pipeline maintaining an environment as more as possible similar to the testing environment, thus minimising integration failures and allowing the development team to move in a more "*familiar*" environment when suggesting actions to the integration team.

This architecture has been proven successful for other small sized projects developed for the level S of the Planck LFI DPC as the *quantization* simulations module.

In this framework the architecture of the decompression module is composed of three main parts:
i) the decompression module (DEM);
ii) an environment service module (ESM);
iii) a driver main module (DMM);
described in the next subsections.

3.1 DEM BASIC REQUIREMENTS

Operations are carried out by the decompression module, which may be composed of more submodules, organised as separated files.

It has to satisfy the following requirements:

DCM-URD-DEM-G.1. DEM shall be an independent module.

When the DEM is included it shall be ready to operate its tasks, apart from the inclusion of standard libraries or objects introduced in the ESM.

DCM-URD-DEM-G.2. DEM shall contains all (and only) the functions required to perform the decompression.

3.2 DMM BASIC REQUIREMENTS

The driver main module is the module which send data to the decompressor extracting them to packets, runs the decompressor and recovers the decompressed data storing them where required, it manages communications and messaging.

In its simplest form DMM shall be a command-line main program produced by the team which will produce the DEM. And it will constitute the example of use from which the integration team will start to integrate DEM in the DPC pipeline. There are two requirements for the DMM:

DCM-URD-ESM-G.1. The use of DMM shall not be mandatory to integrate and operate the DEM, leaving the integration team free to develop their own DMM.

The reason for this is the fact that the DEM may be introduced in a complex environment with definitions conflicting with those already included in the DMM.

DCM-URD-ESM-G.1. The implementation of the DMM shall be done making use of the ESM.

In this way the service module shall document the use of the ESM.

3.3 ESM BASIC REQUIREMENTS

The environment service module (ESM) is an aid to the work of the integration team produced by the development team.

Its basic requirements are:

DCM-URD-ESM-G.1. ESM shall be a separate file containing definitions of variables used by the DMM to run the DEM, plus a single function to initialise them at start-up (if any operation is required for this).

As an example of definition which shall be introduced in the DMM there is the memory areas where to save the data chunk to be decompressed, or the definition of variables where to put messages produced by the DEM.

DCM-URD-ESM-G.2.The use of ESM shall not be mandatory to integrate and operate the DEM, leaving the integration team free to develop their own ESM.

The reason for this is the fact that the ESM may be introduced in a complex environment with definitions conflicting with those already included in the ESM.

4 CODE REQUIREMENTS

4.1 CONSTRAINTS

This section describes specifies further where the code is operated, the operation scenario, the services that the code is expected to receive, and the services the code is expected to produce.

DCM-URD-OPE.1. At production time, the decompression code is operated in the framework of the Planck/LFI DPC Pipeline that constitute its operation environment.

DCM-URD-OPE.2. The code is assumed to be linked to the other parts of the pipeline, while data are expected to be passed through the main computer memory.

DCM-URD-OPE.3. Two memory area shall be defined in the operation environment:

i) the area containing the data chunk to be decompressed;
ii) the area containing the decompressed data chunk.

DCM-URD-OPE.4.The packet structure as the collection of decompression parameters shall be managed by the environment program which drives the decompressor.

4.2 COMMAND LINE INTERFACE REQUIREMENTS

DCM-URD-CLI.1. At development time and in order to test the code before integration, the decompression code shall be operated by a command line interface which will constitute its DMM.

DCM-URD-CLI.2. Command Line Interface shall allow to operate both on Windows/NT and on UNIX machines.

DCM-URD-CLI.3. All the parameters to perform decompression shall be passed through the command line.

DCM-URD-CLI.4. All the messages sent in output shall be put on the standard output.

DCM-URD-CLI.5. Command Line Interface shall allow to read a chuck of data, of a length specified through the command line interface, to be decompressed from a file on a disk.

DCM-URD-CLI.6. The data in the chunk are expected to be written in *big endian* format (most significant bits first).

DCM-URD-CLI.7. Decompressed data shall be written on a file on the disk.

DCM-URD-CLI.8. Decompressed data shall be written in *big endian* format (most significant bits first) both on UNIX and Windows/NT machines.

4.3 DECOMPRESSION MODULE REQUIREMENTS

DCM-URD-DEM.1. The code shall decompress data chunks compressed by the code specified by [1] and [2] operated according to [3].

DCM-URD-DEM.2. The code shall decompress a chunk of data at a time.

DCM-URD-DEM.3. The decompressor module shall not get rid of the packet data structure.

> The elaboration of the packet data structure (as an example the extraction of the data chunk from a packet, or the identification of relevant fields in the packet headers is not a task of the decompression module, but of the modules calling the decompression module).

DCM-URD-DEM.4. The packet structure shall be managed by the calling code program.

DCM-URD-DEM.5. A chunk of data is the full string of data carried by a single packet. No packet breaking or merging is allowed.

DCM-URD-DEM.6. The decompressor shall take in account the presence of the padding bits.

DCM-URD-DEM.7. The chunk of data to be decompressed is stored in the computer memory.

DCM-URD-DEM.8. The decompressed chunk of data is stored in the computer memory.

DCM-URD-DEM.9. The decompression action shall not destroy or damage the compressed data chunk.

DCM-URD-DEM.10. The compressed chunk and the corresponding decompressed data will be stored in different, not overlapping memory locations.

DCM-URD-DEM.11. It is not a task of the DEM to recover data from disk, to fill the memory area containing the compressed data, to define it or to initialize it.

DCM-URD-DEM.12. It is not a task of the decompressor module to save data to disk, to empty the memory area containing the decompressed data, or to define it.

DCM-URD-DEM.13. At call time the decompressor shall receive in input:

1. a pointer with the address of the chunk of data to be decompressed;
2. an integer with its length in octets;
3. an integer with the number of zero padding bits queued to the data chunk;
4. other parameters required by the decompressor to be operated (if any);
5. a pointer to the location where to put the decompressed data;
6. the maximum size of the area holding the decompressed data.

DCM-URD-DEM.14. At call time the decompressor shall give in output:

1. the size of the (safely) decompressed data chunk (integer);
2. the code for the error status (integer);
3. the code for failure level (integer);
4. the message for the failure reason (string);
5. the amount of time in milliseconds required to perform the decompression (float or integer);

4.4 SERVICES REQUIREMENTS

DCM-URD-SRV.1. The code shall give a message stating the success or the failure of the operation. Whenever possible the code shall gives also: the failure level, the reason (i.e. the failure point in the code) and the amount of data recovered.

DCM-URD-SRV.2. No generic error messages shall be allowed.

DCM-URD-SRV.3. Messages shall be specified by the development team and documented.

4.5 FAILURE RECOVERY

DCM-URD-FLR.1.In case of failure the code shall give in output the portion of data which have been safely decompressed.

DCM-URD-FLR.2. The decompressor shall check if the amount of space reserved for the decompressed data is too small to store all the uncompressed data.

DCM-URD-FLR.3. If the area to store the decompressed data is too small, decompression will proceed till the full data chunk will be decompressed or a decompression error will occur, but the area will be filled till its maximum capacity and then the filling will be stopped. A specific error message will be issued as specified in DCM-URD-FLR.4.

The reason for this requirement is to assure that the full data chunk may be properly decompressed regardless of any configuration problem in the pipeline.

DCM-URD-FLR.4. The error message issued in the case contemplated by DCM-URD-FLR.3. shall specify why the filling has been stopped and if decompression has been successful or not.

This requirement is motivated by the need to recognise if a failure has been due to a decompression failure or a memory configuration error.

DCM-URD-FLR.5. In case of failures due to internal exceptions the code shall stop nicely issuing a specific error message without to cause an interruption of the pipeline operations.

Example: floating point exceptions shall be trapped and reported avoiding to cause a core dump.

4.6 CODING CONSTRAINTS

DCM-URD-OCO.1. The code shall be written in standard ANSI C.