Alenia
SPAZIO
A FINMECCANICA COMPANY

HERSCHEL PLANCK

PLAN

TITLE: AIT SOFTWARE MANAGEMENT PLAN

DRL Item or D.R.D. No: V-9

SIGNATURE AND APPROVALS ON ORIGINAL

PREPARED: S. PROCCHIO

CHECKED: S. PROCCHIO

APPROVED: G. ANDRINA / E. FERRARI

AUTHORIZED: P. GIORDANO

APPROVALS:

CONFIGURATION CONTROL: R. DROETTO

PRODUCT ASSURANCE: M. BIANCO

SYSTEM ENGINEER: M. SIAS

PROGRAM MANAGER: P. MUSI

DATA MANAGEMENT: 06/12/2002

DOCUMENT CHANGE RECORD

ISSUE	DATE	REASON FOR CHANGE	AFFECTED PARAGRAPHS
1	17/JUN/02	First issue	All
2	05/DEC/02	Upgraded version to align the document to current CCS configuration and development plus changes introduced to close System PDR RID's:	All
		AIV-21	(4.1)
		AIV-22	(4.1, 4.4, 4.5, 5.2.1, 5.3, 5.4)
		AIV-24	(1.1)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 SCOPE

Scope of this document is to specify the requirements for the development of the Test S/W to be used during Integration and Test activities on both HERSCHEL and PLANCK Service Modules (SVM) and their composing on-board Subsystems.

This Test Software Management Plan document shall be considered as a guideline to be followed when S/W elements for the different AIV activities are to be developed.

The Software Management Plan will establish requirements for the overall S/W architecture and will contain rules for the S/W design and development of the individual elements as the TM/TC definition and use from the Herschel/Planck Satellites Database, the Test Sequences and the Synoptic Pictures.

The Instruments Test Software running on the Instrument Stations for scientific data processing is out of the scope of this document.

2. DOCUMENTS

2.1 APPLICABLE DOCUMENTS

The document to be considered as applicable for the EGSE users are listed here below.
They should be also considered as an help for the not yet expert operator asked to produce whichever kind of Test Software on the Herschel/Planck CCS.

AD [1]	CCS Requirement Specification	H-P-SP-AI-0010
AD [2]	Herschel/Planck System Database Specification	H-P-1-ASPI-SP-0082
AD [3]	Herschel/Planck Naming Convention Specification	H-P-1-ASPI-IS-0141
AD [4]	Herschel Planck Central Checkout System System User Manual	H-P-4-TE-MA-0010

2.2 REFERENCE DOCUMENTS

The listed document can be helpful in order to achieve a general understanding of the general EGSE architecture and of the general architectural design of the CCS.

RD [1]	EGSE Requirements Specification	HP-ASPI-1-SP-0045
RD [2]	EGSE INTERFACE Requirements Specification	H-P-1-ASPI-IS-0121
RD [3]	Herschel/Planck Packet Structure Interface Control Document (PS-ICD)	SCI-PT-IF-07527
RD [4]	Herschel Planck Central Checkout System System Design Document – Hardware	H-P-4-TE-DD-2020
RD [5]	Herschel Planck Central Checkout System System Design Document – Software	H-P-4-TE-DD-2010

2.3 ACRONYMS

APID	Application Process Identifier
CCS	Central Checkout System
CDMU	Control and Data Management Unit
DFE	Data Front End
DS	Data Server
EGSE	Electrical Ground Support Equipment
HK	HouseKeeping
LAN	Local Area Network
MTP	Master Test processor
PFM	ProtoFlight Model
PLM	PlayLoad Module
SCOE	Special Check-Out Equipment
S/C	Spacecraft
S/L	Satellite
SVM	SerVice Module
TC	Telecommand
TCC	Test Conductor Consoles
TM	Telemetry

3. CCS DESCRIPTION

3.1 CSS HARDWARE ARCHITECTURE

The Central Checkout System (CCS) that will be used both for AVM/SVM and for System EGSE configurations will consist of two distinct Servers, one dedicated to run the on-line test environment (MTP) and the other mainly devoted to Off-line activities, i.e.Test software preparation and test results analysis (DS).

Several Test Conductor Consoles (TCC) will be used by the EGSE operator during on-line operations, but they will be also used for off-line preparation of the test software, i.e. Test Sequences/Procedure, Synoptic Pictures and to manage the Data Base (Mirror Site).

For a full detailed description of the CORE EGSE from the hardware point of view, refer to RD [4] , Herschel Planck Central Checkout System / System Design Document – Hardware. The general architecture of the Herschel/Planck CSS is show here below in Figure 3-1.

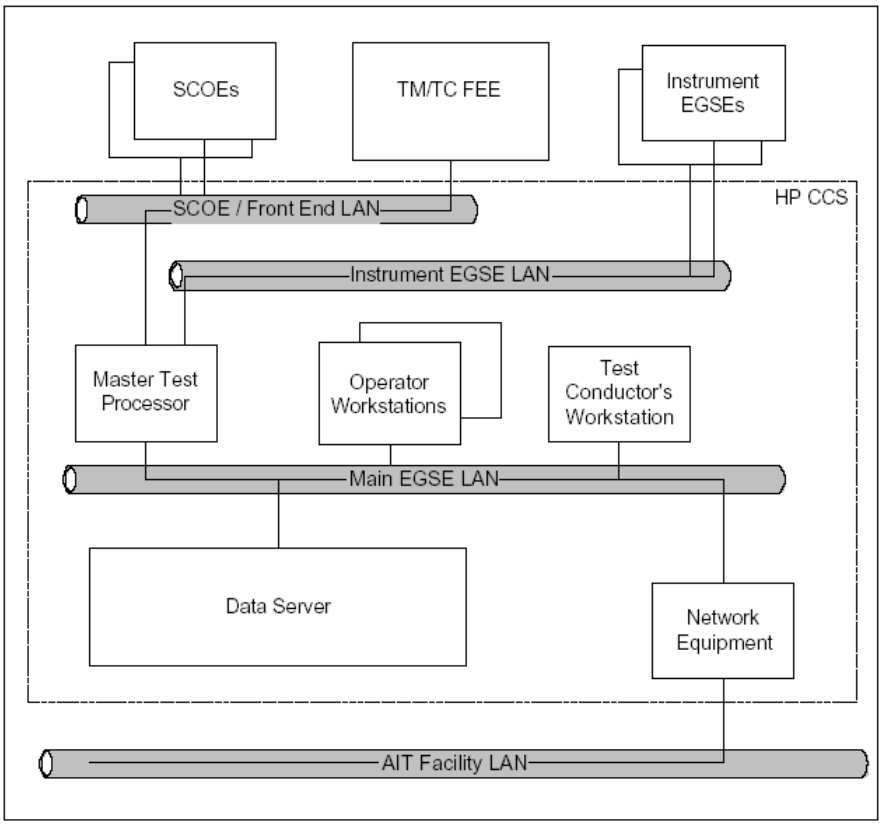

Figure 3-1 CCS hardware architecture overview

It has however to be pointed out that two different version of CCS are foreseen; the only difference between the two CCS is in the number of servers and in the number of workstations. More in details:

- Nominal CCS to be used during AVM/SVM and System tests:
  - 2 servers (MTP & DS)
  - 5 Single screen workstations
  - 3 Double screen workstations

- CCS-Lite to be used for all testing activities on PLM:
  - 1 servers (MTP)
  - 3 Single screen workstations
  - 2 Double screen workstations.

All CCS hardware has been chosen between DELL product line; in particular the currently foreseen main hardware components are listed in the following table.

SERVERS	
CPU	Power Edge 6650 - 4 x PIII Xeon 1400MHz/2MB, 0 Terminators/Redundant Power
Memory	4GB PC133 ECC SDRAM Memory, (8X512MB)
Hard Disk	3 x 73GB SCSI Ultra3 (10,000 rpm) 80 pin Internal Hard Drive managed in RAID-1 configuration
RAID Controller	PERC 3/DC dual channel RAID card with 128MB cache, 1 int and 1 ext channels
Floppy	3.5in 1.44MB Floppy Drive (Kit)
Operating System	Red Hat Linux 7.2 with backup media & documents
Archive System	24x IDE CD-ROM Drive (plus DVD ROM 16X DVD-ROM Drive (No MPEG Support) making part of the Network PC)

Single screen workstations	
CPU	WS Precision 530 MT - Single Xeon 1.8GHz Base Unit, integrated NIC, integrated floppy
Memory	1,024MB PC800 ECC RDRAM Memory, (2X512MB)
Hard Disk	2 x 18GB SCSI U/160M (10,000rpm) Hard Drive
CD-ROM	48X IDE CD-ROM Drive
Monitor	European - 17in Flat Panel Midnight Grey Analog/Digital (1702FP) Monitor
Operating system	Redhat Linux 7.2

Double screen workstations	
CPU	WS Precision 530 MT - Dual Xeon 1.8GHz Base Unit, integrated NIC, integrated floppy
Memory	2,048MB PC800 ECC RDRAM Memory, (4X512MB)
Hard Disk	2 x 18GB SCSI U/160M (10,000rpm) Hard Drive
CD-ROM	48X IDE CD-ROM Drive
Monitor	2 x - 17in Flat Panel Midnight Grey Analog/Digital (1702FP) Monitor
Operating system	Redhat Linux 7.2

3.2 CCS SOFTWARE ARCHITECTURE

The Herschel/Planck Central Checkout System is built around the SCOS2000 EMCS Kernel.
Several modifications to the original SCOS2000 plus a number of new software modules/tasks are required in order to fulfil CCS requirements and provide all the capabilities (as described in AD [1]) needed to allow the Integration and Test activities on both the Herschel/Planck Subsystem's and on the fully integrated Spacecraft's

3.3 CCS SOFTWARE TOOLS

The Software tools available for the CCS operators can be classified into 4 different categories depending on the function they are aimed to; in particular:

- Test Software preparation :
  - Data base operations on Mirror Site
  - Test Sequences preparation
  - Synoptic Pictures preparation
- Test environment preparation
- Configuration Management
- Test result Evaluation.

The scope of this document is mainly to describe the rules to be followed during Test Sequences and Synoptic Pictures preparation, so only these subjects will be deepened in the next paragraphs.

For all the other software tools available on the CCS refer to the specific User Manual (AD [4]).

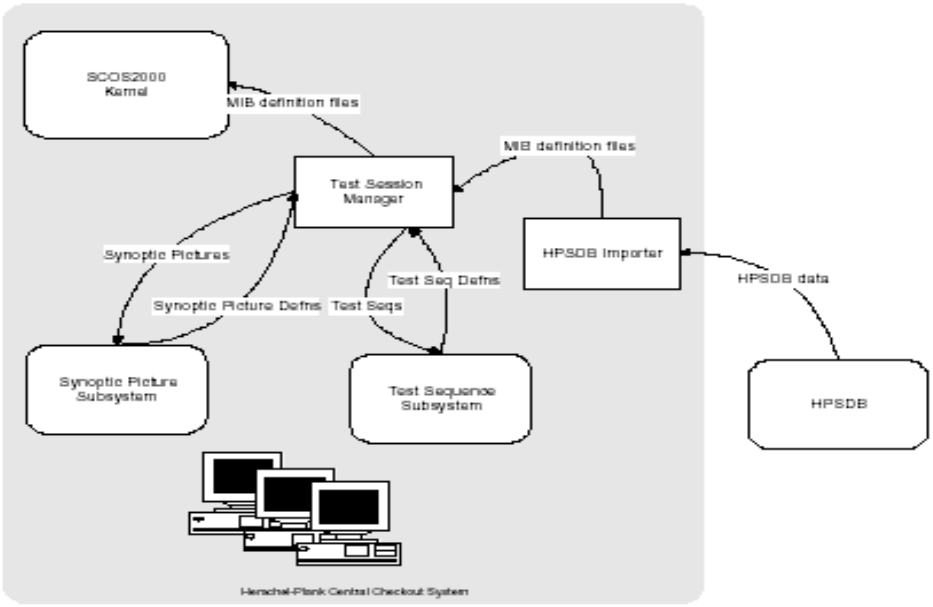

Figure 3.3-1: Overall Test Preparation software Architecture

3.3.1 Test Sequence preparation

In order to write a Test Sequence the operator can use the Test Sequence Editor utility provided as part of the CCS software.
The programming language foreseen on CCS is TCL/TK: it is an interpreted language (refer to AD[4] for details on available statement and relevant syntax/use).
Started in the frame of SESS (CCS Session manager utility), it allows the user to write the code in a driven way, since the coloured annotation, indentation of language keywords and on-line help are provided. The same editor tool can also be started during On-line operation in order to view or eventually edit the active Test Sequence.

Syntax check is available in order to reduce the number of errors before running the Test Sequence, in particular a check will be done in order to verify if all the parameters referenced inside the Test Procedure are defined inside the Data base.

Finally, at run time, the user can choose if the Test Sequence will be run with or without a Debugger running in parallel.
For full details about Test Sequence Editor capabilities and provided functions refer to relevant section of User Manual AD [4] already made available by CCS Supplier.

Moreover, since Test procedure are standard ASCII files, they can be prepared also on other systems with whichever normal text editor; For this reason the CCS will also provide a standard editor tool like NEDIT .

3.3.2 Synoptic Pictures Preparation

Synoptic Pictures are available on CCS in order to provide to the user a pictorial display to organise the monitored data coming from the Spacecraft, the connected SCOE's/DFE plus system parameters relevant to CCS behaviour and health checks.

Synoptic Pictures are managed in the frame of SCOS2000 "MON" component.

Inside SCOS2000 documentation they are referred as "MIMIC displays" and they are defined as animated displays containing mimic elements, which can be animated through telemetry. MIMIC display are prepared using ILOG views studio editor .

For full details about Synoptic Picture Preparation and management capabilities refer to relevant section of CCS User Manual (AD [4]) already made available by CCS Supplier.

An example of MIMIC editor window as implemented in SCOS2000 is shown in Figure 3.3.2-1.

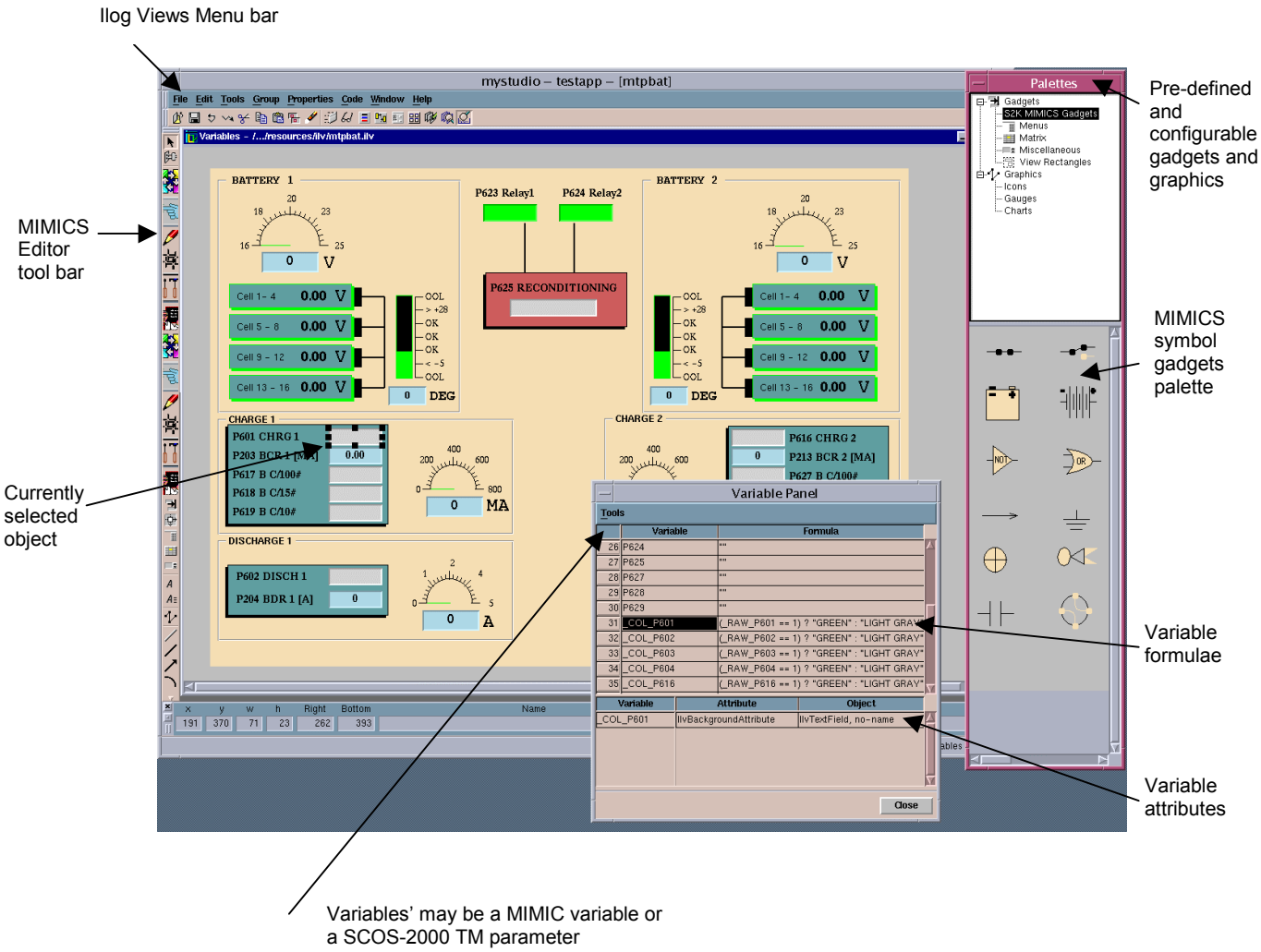

Figure 3.3.2-1. MIMIC Editor window

3.3.3 CCS directories organisation for test software

Directories organisation will be predefined by the CCS administrator and it has to be maintained during the Test Sequences development.
This means that all the Test Sequences have to be coded inside the proper area depending on the main purpose of the test itself or the Subsystem they are testing.

SUBSYSTEM/UNIT TO BE TESTED	DEVELOPMENT AREA
SPACECRAFT	TBD
ACMS	TBD
POWER	TBD
CDMU	TBD
TT&C	TBD
EXPERIMENTS	TBD
….	…

If further directories will be needed in the future they will be added but without changing the tree structure created by the CCS administrator.

On the contrary all the synoptic pictures that will be prepared will be stored together on the same area and properly managed also at HPSDB level.

All the Test Sequences can be prepared or updated in local "DEVELOPMENT" areas on each single workstation. Once they have been configured in the Test Environment repository, they can be however be retrieved back to local areas for updating and modification.
All modification will be kept under configuration control by dedicated Software installed on CCS (CVS tool).

4. FUNCTIONAL REQUIREMENTS

4.1 TEST SEQUENCE TYPES

Each Test Sequence/Procedure must belong to one of the following categories:

- MASTER
  A master Test Sequence is in charge of the high level execution of a complete test starting from configuration A and ending with configuration B, with A=B when necessary. Sequences of this kind can only be executed from keyboard command in order to generate a dedicated output window on the TCC.
- MODULE
  A module Test Sequence is in charge of the execution of a portion of the test, ranging from changes in Spacecraft or subsystem configuration to the verification of an entire functionality. In principle, they are independent from the calling master sequence and the only return code is "sequence completed" or "sequence aborted".
  In other words, MODULE definition can be assimilated (and from here on will be referenced as) to the standard concept of SUBROUTINE.
- LIBRARY
  Library Sequences are in charge for routinary operations on the Satellite subsystems, like command sending/verification, sensor calibration, telemetry processing, SCOE control, unit switch on/off, unit configuration, and all those functions that are common to many test cases. Most of these sequences require some input parameters from the calling program, and return output data in addition to the usual "sequence completed" or "sequence aborted" information.
  In other words, LIBRARY sequences are pieces of code belonging to a common pool of "utility software" (as per standard LIBRARY concept) that can be included inside Test Sequences by all test engineers using the CSS.

Test Sequences of the "MODULE" or "LIBRARY" type are intended for execution from a calling Test Sequence.
Every action performed by a Test Sequence which results in a change of the on-board configuration must be verified in Telemetry for proper execution before proceeding with the test.
Every action performed by a Test Sequence which results in a change of the configuration of SCOE interfaces with the Spacecraft must be verified in the SCOE monitors and/or in the Telemetry for proper execution before to continue with the test.

4.2 TRY AND ERROR STRATEGY

Whenever a fault condition is detected by the running Test Sequence, corrective actions must be taken by the software according to the seriousness of the problem, as follows:

- **Failure**:
  The reported condition can jeopardise on-board or test equipment or is hazardous to the

personnel. The test software must immediately initiate a recovery procedure to set the system in a safe configuration. At the same time the Test Conductor must be notified of

- the failure,
- the start of an emergency procedure
- The result of the recovery action.

- **Malfunction**:
  The reported anomaly cannot cause damage to on-board or test equipment, nor to personnel working on the Satellite. The running test shall be suspended and the Test Conductor must be notified of the anomaly. Corrective actions or troubleshooting, if necessary, will be organised by the Test Conductor.
  Whenever the consequence of an action performed by a Test Sequence is not verified, the operator shall be prompted to choose among the following possibilities:
  - abort the test
  - retry the same action
  - Skip the verification and continue.

4.3 TEST SOFTWARE REUSE

Reuse of Test Software, and in particular of the Sequences, is strongly encouraged as far as the sequence objectives, test approach and success criteria can be acceptable.
This means that the maximum possible effort shall be done in order to reuse for HERSCHEL/PLANCK FM SVM the already validated software used at AVM level trying to keep to a minimum the modifications required to adapt.
The same philosophy can be applied with respect to System level testing.

4.4 NAMING CONVENTIONS

Restrictions on the length of the Test Sequence names or on the set of allowed characters are those determined by the Test Language available on CCS and by the MTP implementation; in particular TCL/TK Language allows the user to use up to 255 characters for Test Sequence names.

In general, the names used to identify Test Sequences should be meaningful of the sequence purposes. Whenever the target of the Test Sequence operations is restricted to a specific unit or subsystem, this should be reflected in the name.

Examples:

- Test sequences performing unit functional checks or configurations:

CDMUSWAP	Changes the active CDMU
ACCTEST1	Performs functional test #1 on ACC
ACCDUMP	Performs a dump of the ACC memory
RWLEND	Performs the end to end test of the reaction wheels

- Test Sequences performing subsystem tests or configurations:

EARTHRATE Perform the gyro scaling factor calibration
AOCSCL01 Perform AOCS Closed Loop test #1

- Test Sequences performing system test or configurations:
  ORBIT Perform the Orbit Simulation test.

Keeping into account also the subsystems identified for the definition of TM/TC parameters inside the HPSDB (refer also to AD [3]) it can be possible to define at least the following "keywords" to be inserted as a prefix in the Test Sequences names:

AAD	**HIFI**	**SAS-P**
ACC	**LFI**	**SOLAR_AR_H**
ACC_SW_H	**LGA**	**SOLAR_AR_P**
ACC-SW_P	**MGA**	**SPIRE**
BATTERY	**PACS**	**SRP_COOLER**
CDMU	**PCDU**	**STR**
CDMU_SW_H	**QRS**	**STR_MAPPER**
CDMU_SW_P	**RAD_MON**	**STR_SW**
CRYO_ELEC	**RCS_H**	**SYSTEM**
CRYOSTAT	**RCS_P**	**THERMAL_H**
FRAME_STR	**RFDN**	**THERMAL_P**
FSS	**RW**	**TRSP**
GYRO	**RWE**	**TWTA**
HFI	**SAS-H**	**VISUAL_MON**

4.5 CONFIGURATION CONTROL

Rules established by the Configuration Control Tool (CVS) of the CCS and relevant to the Test Sequences shall apply.

4.6 ARCHITECTURE OF TEST SOFTWARE

Test Software will be organised in such a way to avoid as much as possible duplication of software: this means for example that there will be only one sequence aimed to power ON and power OFF the SVM. These sequences will be organised with a menu structure allowing the user to select the configuration he needs to reach before starting the tests.

Similar philosophy will be also adopted for all the other running Test Sequence: for example a Master test sequence for each phase of test (IST, SIT, SVT, …) will be prepared, possibly one for each Subsystem under test, in order to group/organise the software in a simple and safe way for the Test Engineers.
Obviously all Test Software will be keep under Configuration Control with the CCS provided tool; and also with a correct use and management of the Test Environment and Test session concept (see also AD [4]).

4.7 INPUT / OUTPUT CONVENTION

The CCS implementation allows up to 20 Test Sequences running in parallel during on-line operations. After each Test Sequence starting (it doesn't matter if started by keyboard command or by a father Test Sequence), the CCS software automatically manages/displays all input/output messages for this Test Sequence.

In order to standardise in some way the windows' utilisation, a set of elementary rules must be established to organise the Test Sequence output on the output window.

Test Sequence messages displayed, as a consequence of user statements, will have the following format:

<TS_name><:><text expression>

where <test expression> is a normal text string.

Each Test Sequence will be monitored with at least the following kinds of messages/information provided by the CCS software itself:

- Test Sequence status (started, hold, prompt, …)
- Messages informing that a secondary (modules) Test Sequences will be executed/performed upon Test Language statement
- Runtime updating information
- Error messages
- Prompt messages due request for input or anomalies.

Sequence polling for the occurrence of some event must notify its status by periodically displaying messages in the Test Sequence window.

4.8 SEQUENCE TO SEQUENCE INTERFACE DEFINITION

The interface between a master and a slave Test Sequence shall be implemented by using shared variables or ad hoc defined parameters to pass input parameters and results.

Definition, use and management of shared variables can be found in relevant section of CCS User Manual (AD [4]) already made available by CCS Supplier.

As a minimum, the user shall define the following variables, shared by all the Test Sequences

- Interactive/Batch execution mode for the slave Test Sequence. Sequences executed interactively will prompt the operator for the input parameters; Sequences running batch will use data from the pool of global variables.
- Successful/Unsuccessful Test sequence completion: the slave Test Sequence will inform the master for the successful/unsuccessful execution of the operations. The code to be returned to the master sequence in case the slave execution is aborted shall be "Unsuccessful".
- Time-out error in operation execution.

- Others as needed.

4.9 LOGGING AND ARCHIVING

Generally, all information necessary to demonstrate the proper execution of the test and the correctness of the achieved results must be included into the Test Sequence logfile.

The following classes of events, as a minimum, some of them automatically generated by CCS software and some other coming from user defined output messages, will be logged:

- Start of the Test Sequence execution, specifying the release of the software and whether the execution has been requested by keyboard command or by another Test Sequence
- Test Sequence termination, specifying whether it was due to nominal completion or test aborted by operator request
- Entering a commanded HOLD status
- Resuming the operation from a commanded HOLD status
- Entering the PROMPT status, including the prompt reason
- Resuming the operation from a PROMPT status, including the operator reply
- Start of test phases, specifying phase objective and major test parameters
- Conclusion of test phases, reporting the achieved result
- Commanding configuration changes to SCOE
- Commanding the execution of child Test Sequences
- Resuming operations returning in control after child Test Sequence completion, specifying whether the other Test Sequence reported any error condition
- Issuing Telecommands on-board, specifying the TC content or the TC data when the command is built by the sequence itself.

5. CODING RULES

5.1 VARIABLE DECLARATION

The rules listed here below should be followed for variables declaration:

- Variables used in the Test Sequence must be declared (even if this is not mandatory in TCL/TK lasnguage) before any executable statement, in the dedicated code section.
- Variable names should be meaningful and readable; avoid obscure abbreviations
- Variable names should be in English
- For basic variables, the type must precede the variable name, and the declaration must be completed with a comment to explain the variable purpose.
- Whenever not trivial, only one variable should be declared per line.

Example:

```
REAL  BATVOLTAGE          ; Voltage of the Simulated Battery
REAL  BUFFER (20)         ; Temporary storage for telemetry data
INT   I, J, K             ; The usual set of counters
```

5.2 TEST SEQUENCE LAYOUT

The Test Sequence code listings shall be organised in the following parts:

- Test Sequence header
- Test Sequence body
- Test Sequence Trailer

5.2.1 Test Sequence Header and template

The header of a Test Sequence will consist in a series of "remark" lines in which all information concerning the Test Sequence objectives, the system resources that are required to run properly and all the information that could be necessary to explain how it works. The typical set of information that should be included in the Test Sequence header are the following:

- Test sequence release data, including:
  - Test Sequence Name
  - Test Sequence Version
  - Test Sequence purpose
  - Relevant Subsystem(s)
  - Test Sequence type (MASTER/SUBROUTINE/LIBRARY)
  - Author of the program
  - Creation Date
  - Last revision Date
- Test Sequence Description
- Test Sequence Constraints
- Allocated Resources
- Returned Data, including Error Management
- Development history.

A predefined standard Test Sequence header will be present on the CCS; a possible template format for Test Sequences running on Herschel / Planck CCS is shown here below.

```
# ****************************************************************

# Test Sequence name <TS_name>
#
# ****************************************************************
# Project:                HERSCHEL/PLANCK
# Satellite under test:   <S/L><Model>
# SubSystem under test:   <S/S><Model>
# Language:               TOPE - TCL
# User:                   <User name>
# Author:                 <OperatorName>
# File name:              Source: <directory_name><file_name.TCL>
# Type:                   < MASTER >
# Version:                <Revision_Number>
# Description:
#
#
#       <Test Sequence description showing the main purposes/goal of the test>
#
#
# Subroutines:
#
```

```
#
#       <List of Subroutines/Modules called by the Test sequence>
#
#
# External Subroutines/Libraries:
#
#
#       <List of External Subroutine/Libraries called by the Test sequence>
#
#
# Batch/Interactive:
#
#       <Running modality>
#
#
# Input/Output parameters:
#
#
#       <List of Input/Output parameters>
#
#
# ********************************************************************
#
# <Configuration control information as managed by the CCS "CSV" tool>
#
#
# ********************************************************************
#
#       <List of used Variable including
                <Variable_name>
                <Variable_type (even if automatically managed by the language)>
                <Variable_short_description>
                <Variable_usage (local var., global var, ..)
                <Variable_default_value>
                >
#
#
# ********************************************************************


                Test Sequence Body


# ********************************************************************
# End of <TS_name> Test Sequence
# ********************************************************************
```

5.3 SUBROUTINE LAYOUT

Each subroutine that will be included into a Test Sequence shall be preceded by a comment heading consisting of documentation that includes the following blocks:

- Banner: consists in a line of repeated characters (asterisk, "-", "=") across the page
- Name: one line containing the subroutine name
- Description: a complete description of the routine task and the instructions for usage
- Parameters: meaning of the input parameters
- Returns: the word "RETURNS:" followed by a description of the possible return values. If the subroutine may return an error status, the list of the possible errors and under what condition they are returned must be included. If no output parameters are returned, "N/A" shall be indicated.
- End Banner: terminate the description section of the subroutine, and consists in a line of repeated characters (asterisk, "-", "=") across the page

A predefined standard Subroutine header will be present on the CCS; a possible template format for Subroutines (and Libraries) is shown here below.

```
# ****************************************************************
#
# Subroutine name <Subroutine_name>
#
# ****************************************************************
# Project:                HERSCHEL/PLANCK
# Satellite under test:   <S/L><Model>
# SubSystem under test:   <S/S><Model>
# Language:               TOPE - TCL
# User:                   <User name>
# Author:                 <OperatorName>
# Master Test Sequence    < Master_Test_Sequence_name>
# File name:              Source: <directory_name><file_name (only for external subroutines or
#                          Libraries)>
# Type:                   < SUBROUTINE or LIBRARY >
# Version:                <Revision_number (only for external subroutines or Libraries)>
# Description:
#
#
#       <Subroutine description showing the main purposes/goal of the test>
#
#
# Input/Output parameters:
#
#
#       <List of Input/Output parameters including "returned" variables/values>
#
#
# ****************************************************************
```

```
#
#       <Configuration control information as managed by the CCS "CSV" tool
#       (only for external subroutines or Libraries)>
#
#
# ********************************************************************
#
#       <List of used Variable including
                <Variable_name>
                <Variable_type (even if automatically managed by the language)>
                <Variable_short_description>
                <Variable_usage (local var., global var, ..)
                <Variable_default_value>
                >
#
#
# ********************************************************************


                Subroutine Body


# ********************************************************************
# End of <Subroutine_name> Subroutine
# ********************************************************************
```

5.4 LIBRARY

The same approach foreseen for Subroutines can be applied also to Library.

5.5 TELECOMMAND REFERENCES

Telecommands will be referenced inside the Test Sequences with the same name as defined inside the HPSDB.
Refer to AD [3] where naming convention to be used for HPSDB population are defined.
An example of statements in which Telecommands is referenced is the ones used to send a command to on-board units after it has been authorised (see AD [4]):

authorise <TC_packet_name>
……
tcsend <TC_packet_name>
……

5.6 PARAMETERS REFERENCES

Parameters will be usually referenced inside the Test Sequences with the same name as defined inside the HPSDB (refer to AD [3] where naming convention to be used for HPSDB population are defined).
An example of how the operator can have direct access to a Telemetry parameter is given by the fetch command:

fetch <TM_parameter_name>

A different way to have access to parameters, in case not only the instant values are required, but some processing have to be performed on them is depicted here below:

subscribe <TM_parameter_name> **referby** <varName>

In this particular statement attention shall be paid to <TM_parameter_name> and <varName> since the two parameters identifiers cannot be identical.
The convention that will be used by CCS operators will be:

<TM_parameter_name>	is the Parameter identifier as defined in HPSDB
<varName>	Mnemonic name for the parameter – eventually the same name as contained inside HPSDB but written in lowercase -**TBC**

Refer to AD [4] for further details.