Earth Explorer
Mission CFI Software

GENERAL SOFTWARE USER MANUAL

Internal Code:	P/SUM/DMS/01/026-006
Project Code:	CS-MA-DMS-GS-0002
Issue:	3.0.1
Date:	13/08/04

	Name	Function	Signature
Prepared by:	Mariano Sánchez-Nogales	Project Engineer	
	Juan José Borrego Bote	Project Engineer	
Checked by:	José Antonio González Abeytua	Project Manager	
Approved by:	José Antonio González Abeytua	Project Manager	

DEIMOS Space S.L.
Ronda de Poniente, 19,
Edificio Fiteni VI, Portal 2, 2ª Planta, Tres Cantos
28760 Madrid, SPAIN
Tel.: +34 91 806 34 50
Fax: +34 91 806 34 51
E-mail: deimos@deimos-space.com

Document Information

Contract Data		Classification	
Contract Number:	15583/01/NL/GS	Internal	
		Public	
Contract Issuer:	ESA / ESTEC	Industry	X
		Confidential	

External Distribution		
Name	Organization	Copies

Electronic handling	
Word Processor:	Framemaker 6.0
Electronic file name:	cs-ma-dms-gs-0002-21

Document Status Log

Issue	Change Description	Date	Approval
1.0	New document	08/11/01	
2.0	Release of CFI libraries version 2.0.	29/11/02	
2.1	Release of CFI libraries version 2.1.	13/05/03	
2.2	Release of CFI libraries version 2.2.	30/09/03	
3.0	Release of CFI libraries version 3.0	21/07/04	

Table of Contents

List of Tables

List of Figures

This page is intentionally left blank

1 SCOPE

The Software User Manual (SUM) of the Earth Explorer Mission CFI Software is composed of:

- General document describing the sections common to all the CFI software libraries.
- Specific document for each of those libraries.

This document is the ***General Software User Manual***. It provides an overview of the CFI libraries and describes the software aspects that are common to all those libraries.

The following specific SUM's are also available:

- EXPLORER_LIB Software User Manual, Issue 3.0 ([XL_SUM])
- EXPLORER_ORBIT Software User Manual, Issue 3.0([XO_SUM])
- EXPLORER_POINTING Software User Manual, Issue 3.0([XP_SUM])
- EXPLORER_FILE_GEN Software User Manual, Issue 3.0([XG_SUM])
- EXPLORER_VISIBILITY Software User Manual, Issue 3.0([XV_SUM])
- EXPLORER_FILE_HANDLING Software User Manual, Issue 3.0([XF_SUM])
- EXPLORER_GEO_CORRECTIONS Software User Manual, Issue 1.2 ([XC_SUM])
- EXPLORER_RETRACKER Software User Manual, Issue 1.1 ([XT_SUM])

2 ACRONYMS AND NOMENCLATURE

2.1 Acronyms

ANX	Ascending Node Crossing
CFI	Customer Furnished Item
ESA	European Space Agency
ESTEC	European Space Technology and Research Centre
RAM	Random Access Memory
SUM	Software User Manual

2.2 Nomenclature

CFI	A group of CFI functions, and related software and documentation. that will be distributed by ESA to the users as an independent unit
CFI function	A single function within a CFI that can be called by the user
Library	A software library containing all the CFI functions included within a CFI plus the supporting functions used by those CFI functions (transparently to the user)

3 APPLICABLE AND REFERENCE DOCUMENTS

3.1 Reference Documents

[MCD] Earth Explorer Mission CFI Software. Mission Conventions Document. CS-MA-DMS-GS-0001. Issue 1.2. 15/04/02.

[XL_SUM] Earth Explorer Mission CFI Software. EXPLORER_LIB Software User Manual. CS-MA-DMS-GS-0003. Issue 3.0. 21/07/04.

[XO_SUM] Earth Explorer Mission CFI Software. EXPLORER_ORBIT Software User Manual. CS-MA-DMS-GS-0004. Issue 3.0. 21/07/04.

[XP_SUM] Earth Explorer Mission CFI Software. EXPLORER_POINTING Software User Manual. CS-MA-DMS-GS-0005. Issue 3.0. 21/07/04.

[XG_SUM] Earth Explorer Mission CFI Software. EXPLORER_FILE_GEN Software User Manual. CS-MA-DMS-GS-0006. Issue 3.0. 21/07/04.

[XV_SUM] Earth Explorer Mission CFI Software. EXPLORER_VISIBILITY Software User Manual. CS-MA-DMS-GS-0007. Issue 3.0. 21/07/04.

[XF_SUM] Earth Explorer Mission CFI Software. EXPLORER_FILE_HANDLING Software User Manual. CS-MA-DMS-GS-0008. Issue 3.0. 21/07/04.

[XC_SUM] Earth Explorer Mission CFI Software. EXPLORER_GEO_CORRECTIONS Software User Manual. Issue 1.2. 09/08/01.

[XT_SUM] Earth Explorer Mission CFI Software. EXPLORER_RETRACKER Software User Manual. Issue 1.1. 04/09/01.

4 INTRODUCTION

This ***General Software User Manual*** consists of the following sections:

- Introduction explaining how to use this document (section 4)
- Overview of the CFI libraries (section 5), indicating the CFI functions available within each of the CFI software libraries, and the data and control flow between those libraries.
- Installation guide (section Figure 4:), explaining how to get, install and validate any of the CFI software libraries, as well as listing the software items provided with the delivery of the related CFI.
- Library usage overview (section 7), describing how to create a user application
- Detailed description of the error handling functions which are delivered with each CFI. This is described in this document because all CFIs use exactly the same error handling mechanism

The ***specific Software User Manual*** of each CFI software library ([XL_SUM], [XO_SUM], [XP_SUM], [XG_SUM], [XV_SUM], [XF_SUM], [XC_SUM] and [XT_SUM]) describes in detail the use of each of the CFI functions included within that library, as well as refine the description regarding how to use that library.

In addition to the general and specific SUM for a CFI library, the user must refer to the ***Mission Conventions Document*** ([MCD]) for details on the time references and formats, reference frames, parameters and models used in all these software user manuals.

5 CFI LIBRARIES OVERVIEW

The Earth Explorer Mission CFI Software is a collection of software functions performing accurate computations of mission related parameters for Earth Explorer missions.

Those functions are delivered in the form of software libraries gathering together the functions that share similar functionalities.

An overview of the complete CFI software collection is presented in figure 1.

EXPLORER_FILE_HANDLING
- Read XML files
- Write XML files
- Verify XML files

EXPLORER_LIB
- Basic computations
- Time functions
- Reference frames functions

EXPLORER_ORBIT
- Initialisations
  - xo_orbit_init_def
  - xo_orbit_cart_init
  - xo_orbit_init_file
  - xo_propag_init
  - xo_interpol_init
- Propagation / Interpolation
  - xo_propag
  - xo_interpol
- Ancillary results
  - xo_propag_extra
  - xo_interpol_extra
- Memory clean-up
  - xo_orbit_close
  - xo_propag_close
  - xo_interpol_close
- Other functions
  - xo_orbit_info
  - xo_orbit_abs_from_rel
  - xo_orbit_abs_from_phase
  - xo_orbit_rel_from_abs
  - xo_time_to_orbit
  - xo_orbit_to_time

EXPLORER_POINTING
- Satellite frames initialisations
  - xp_sat_nominal_att_init_<mode>
  - xp_sat_att_init_<mode>
  - xp_instr_att_init_<mode>
- Attitude functions
  - xp_attitude_init
  - xp_attitude_compute
  - xp_attitude_user_set
- Atmospheric and DEM initialisations
  - xp_atmos_init
  - xp_dem_init
- Target functions
  - xp_target_<mode>
  - xp_multi_target<mode>
  - xp_target_extra_vector
- Ancillary results
  - xp_target_extra_<param_extra>
- Memory clean-up
  - xp_sat_nominal_att_close
  - xp_sat_att_close
  - xp_instr_att_close
  - xp_attitude_close
  - xp_atmos_close
  - xp_dem_close
  - xp_target_close
- Other functions
  - xp_change_frame

Figure 1: Earth Explorer CFI software libraries

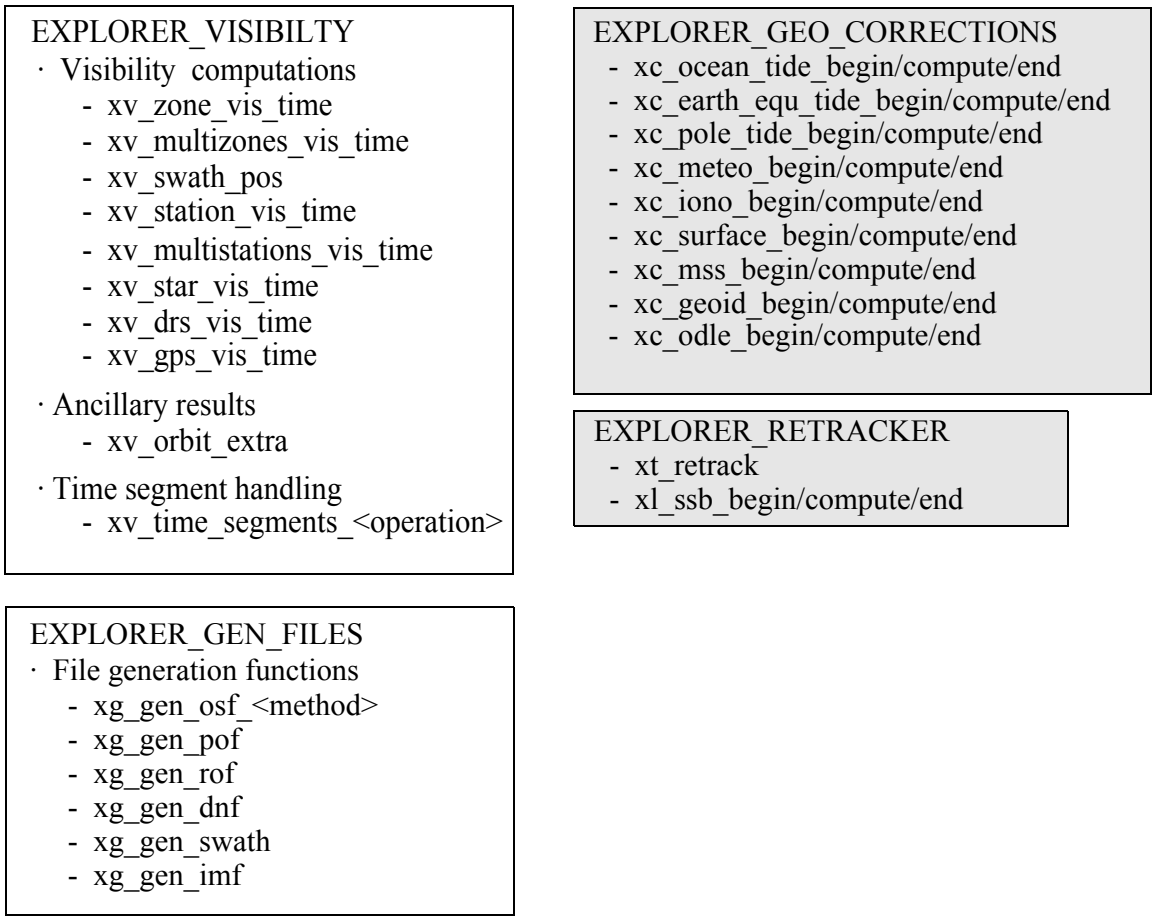


Figure 1: Earth Explorer CFI software libraries

Those libraries aimed to instrument processing appear shadowed in the previous diagram.

The CFI software libraries are to be seen as several layers, each layer being directly accessible to a user's program. Lower layers are more generic functions which are likely to be used by most application software, whereas higher level layers are more specialized functions which are to be used for more specific tasks.

figure 2 shows the software dependencies between the CFI software libraries, where each row between libraries indicates that the higher level library requires the lower level one to operate.

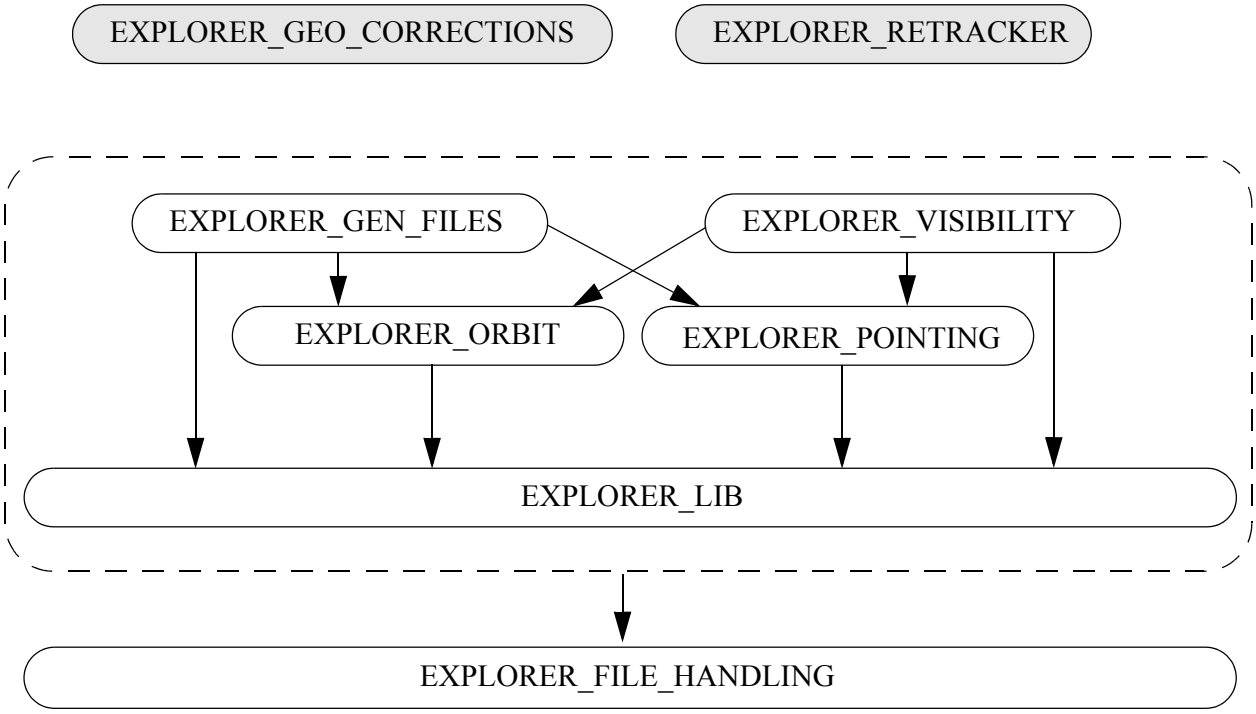


Figure 2: Earth Explorer CFI Software libraries dependencies

Furthermore, the high level data flow between those CFI libraries are shown in figure 3:

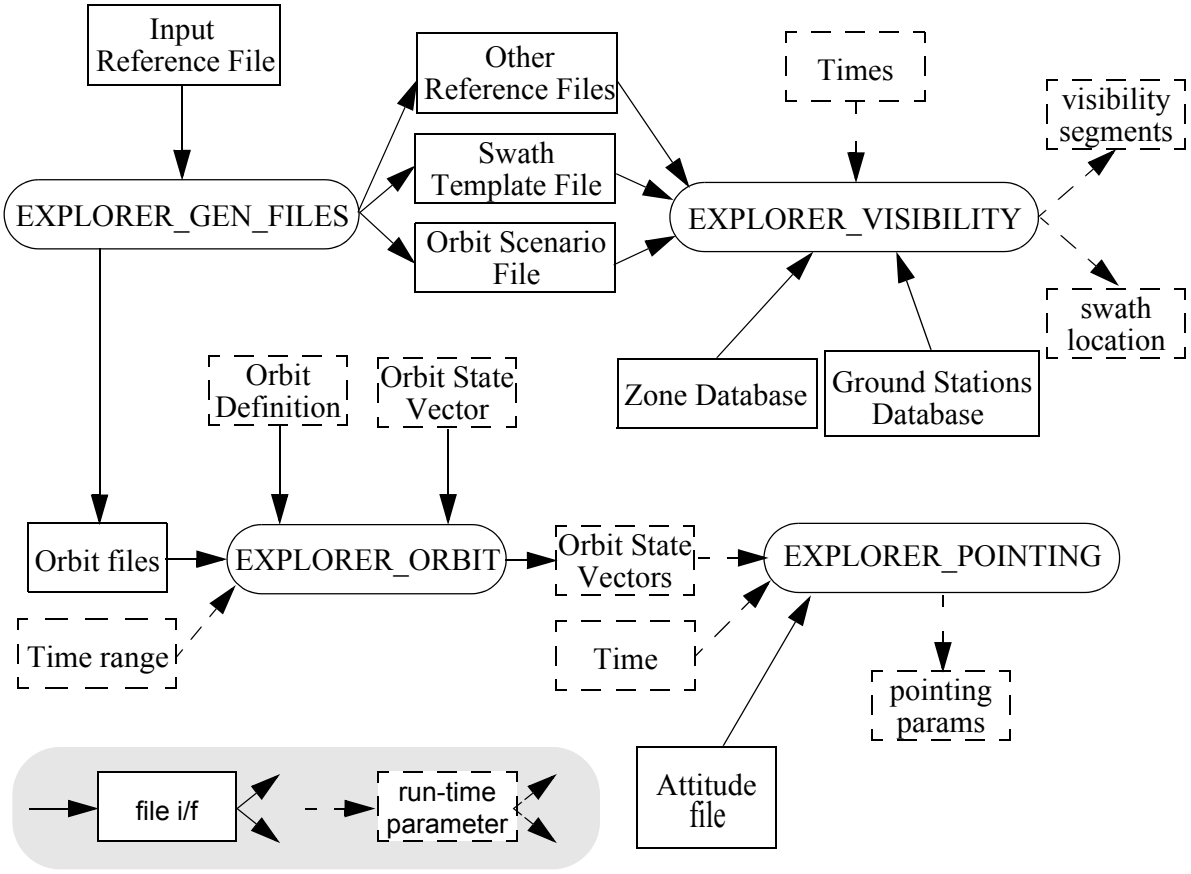


Figure 3: EXPLORER_ORBIT, POINTING, GEN_FILES, VISIBILITY data flow

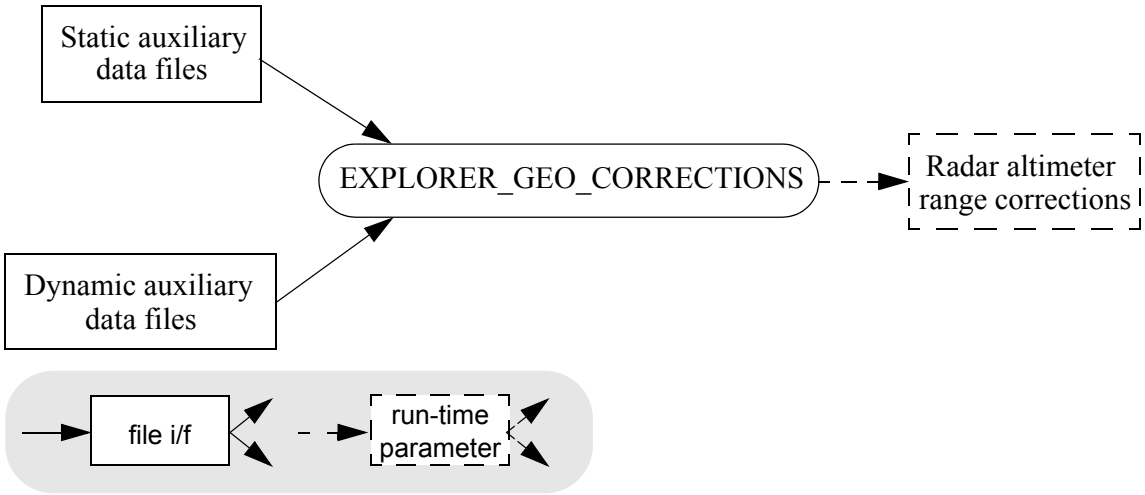


Figure 4: EXPLORER_GEO_CORRECTIONS data flow

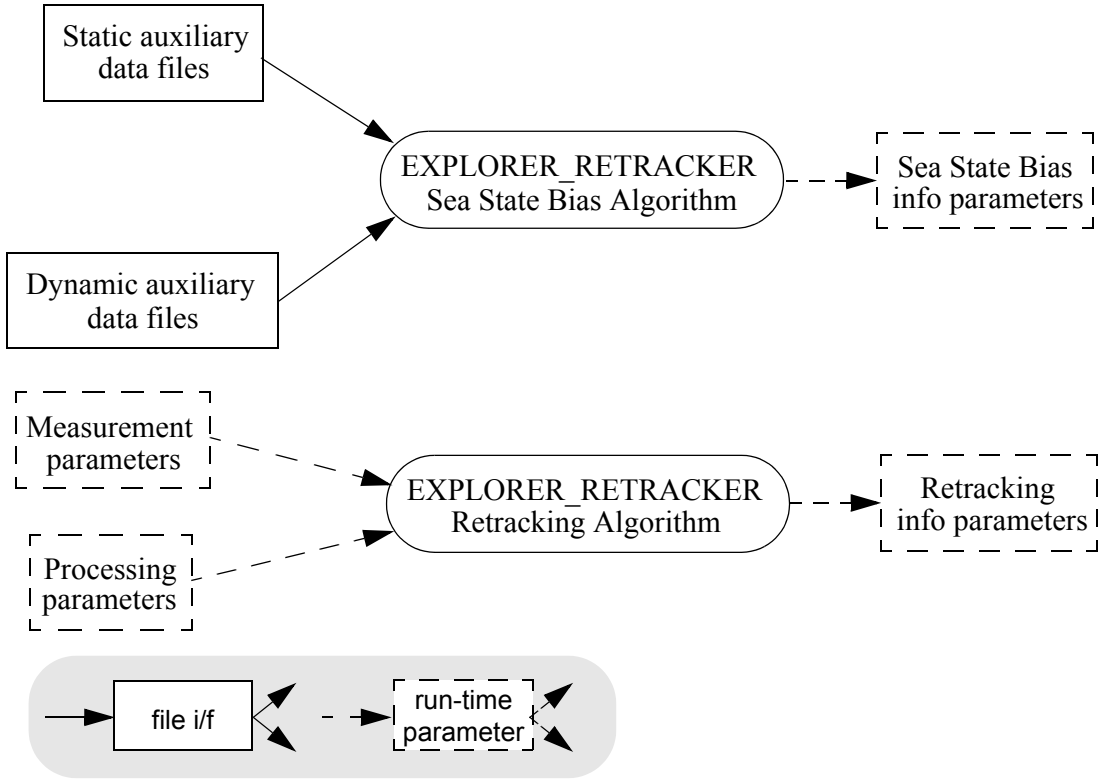


Figure 5: EXPLORER_RETRACKER data flow

6 CFI LIBRARIES INSTALLATION

This section describes the procedures to get, install and validate the installation of a CFI software library. It also describes the directory structure and available files resulting from a successful installation.

These procedures and structures are the same for each of the available CFI software libraries and so, they will be described in this document for a generic CFI, namely *cfi_name*. To perform an actual installation, please follow the procedures while replacing *cfi_name* with the appropriate name, i.e. one of:

- `explorer_file_handling`
- `explorer_lib`
- `explorer_orbit`
- `explorer_pointing`
- `explorer_gen_files`
- `explorer_visibility`
- `explorer_geo_corrections`
- `explorer_retracker`

6.1 Usage Requirements

Each CFI software library is distributed as an object code callable from C, C++ and ForTran, The object code is completely system dependent. In this sense, different computer platforms are supported:

- Sun under Solaris
- PC under Windows 95/98/NT/2000
- PC under Linux
- Macintosh under MacOS X

Some of the libraries are not available for all operative systems. In this sense, the following table presents the correspondence between software libraries and operative systems.

Table 1: CFI libraries availability

CFI Libraries	Solaris	Windows	Linux	MacOS X
explorer_lib	X	X	X	X
explorer_orbit	X	X	X	X
explorer_pointing	X	X	X	X
explorer_gen_files	X	X	X	X
explorer_visibility	X	X	X	X
explorer_file_handling	X	X	X	X
explorer_geo_corrections	X	-	X	-
explorer_retracker	X	-	X	-

6.1.1 Sun under Solaris

The source code has been compiled on a Sun SparcStation under Solaris 2.7 and using the free software *gcc* compiler.

The software requirements are:

- Solaris 2.7 (or later) Operating System
- Solaris 2.7 (or later) *libm.a* (or *libm.so*) mathematical library
- *gcc* compiler version 3.2 (for linking the software to a C application)
- *g77* compiler version 0.5.26 (for linking the software to a ForTran application)
- *libxml2* version 2.5.7 or later

The following syntax is used within the headers, enabling the use of the libraries in C++:

```
#ifdef __cplusplus
  extern "C" {
#endif
   -----prototyping
   -----prototyping
#ifdef __cplusplus
  }
#endif
```

The hardware requirements are:

- Sun SparcStation
- TBD Mb free of disk space (for FILE_HANDLING, LIB, ORBIT, POINTING, GEN_FILES and VISIBILITY libraries)
- TBD Mb RAM (for FILE_HANDLING, LIB, ORBIT, POINTING, GEN_FILES and VISIBILITY libraries)

6.1.2 PC under Windows 95/98/NT/2000

The source code has been compiled on a PC under Microsoft Windows 2000 and using the software *Microsoft Visual C++ 6.0* compiler.

In summary, the software requirements are:

- Microsoft Windows 2000 Operating System.
- *Microsoft Visual C++ 6.0* Compiler (for linking the software to a C application)
- *TBD* (for linking the software to a ForTran application)
- *libxml2* version 2.5.7 or later

The hardware requirements are:

- PC
- TBD Mb free of disk space (for FILE_HANDLING,LIB, ORBIT, POINTING, GEN_FILES and VISIBILITY libraries)
- TBD Mb RAM (for FILE_HANDLING,LIB, ORBIT, POINTING, GEN_FILES and VISIBILITY libraries)

6.1.3 PC under Linux

The source code has been compiled on a PC under Linux 2.4.18 (RedHat 8.0) and using the free software *gcc* compiler version 3.2.

In summary, the software requirements are:

- Linux 2.4.18 (RedHat 8.0)
- *gcc* compiler version 3.2 (for linking the software to a C application)
- *g77* compiler version 0.5.26 (for linking the software to a ForTran application)
- *libxml2* version 2.5.7 or later

The hardware requirements are:

- PC
- TBD Mb free of disk space (for FILE_HANDLING,LIB, ORBIT, POINTING, GEN_FILES and VISIBILITY libraries)
- TBD Mb RAM (for FILE_HANDLING,LIB, ORBIT, POINTING, GEN_FILES and VISIBILITY libraries)

6.1.4 Macintosh under MacOS X

The source code has been compiled on a Macintosh under MacOS X and using gcc 3.0 compiler.

In summary, the software requirements are:

- Mac OS X version 10.2.6 or later
- *gcc* compiler version 3.3 (for linking the software to a C application)
- *g77* compiler version 0.5.26 (for linking the software to a ForTran application)
- *libxml2* version 2.6.2. It is not possible to run the executable programs (such as the xp_converter or the file generation executables) linking with a *libxml* version different from the version used to build those executables.

The hardware requirements are:

- Macintosh
- TBD Mb free of disk space (for FILE_HANDLING,LIB, ORBIT, POINTING and VISIBILITY libraries)
- TBD Mb RAM (for FILE_HANDLING,LIB, ORBIT, POINTING and VISIBILITY libraries)

6.2 How to get the Software

To get the software, please send an e-mail to:

cfi@jw.estec.esa.nl

Retrieve all the files corresponding to desired version number (X.X):

cfi_name_X.X.tar.gz (*cfi_name* might also be replaced by `explorer_cfi`, for all the libraries)

If any patch is delivered, with the following syntax:

cfi_name_X.X.patch<letter>.tar.gz

retrieve it also together with any release note available.

6.3 How to install the Software

To install the software (procedure for UNIX users. Other users should adapt it to their own configuration and tools):

1. Copy the retrieved files in the directory where you want to install the distribution, say *any_dir*
2. Go to directory *any_dir*
3. Read the file `X.X README` if present, and follow the recommendations if any
4. Uncompress the distribution file using:

   gunzip *cfi_name_X.X*.tar.gz
5. Unpack the file using:

   tar xvf *cfi_name_X.X*.tar
6. Remove the distribution file using:

   rm *cfi_name_X.X*.tar
7. If you had not done this yet, uncompress the general documents using:

   gunzip release_notes *X.X*.pdf.gz

   gunzip general_SUM *X.X*.pdf.gz

   gunzip MCD *X.X*.pdf.gz
8. (optional) if patches are available, repeat above steps to overwrite with the updated files.

6.4 Overview of Files and Directory Structure

Upon completion the installation procedure, the following directory structure will be created.

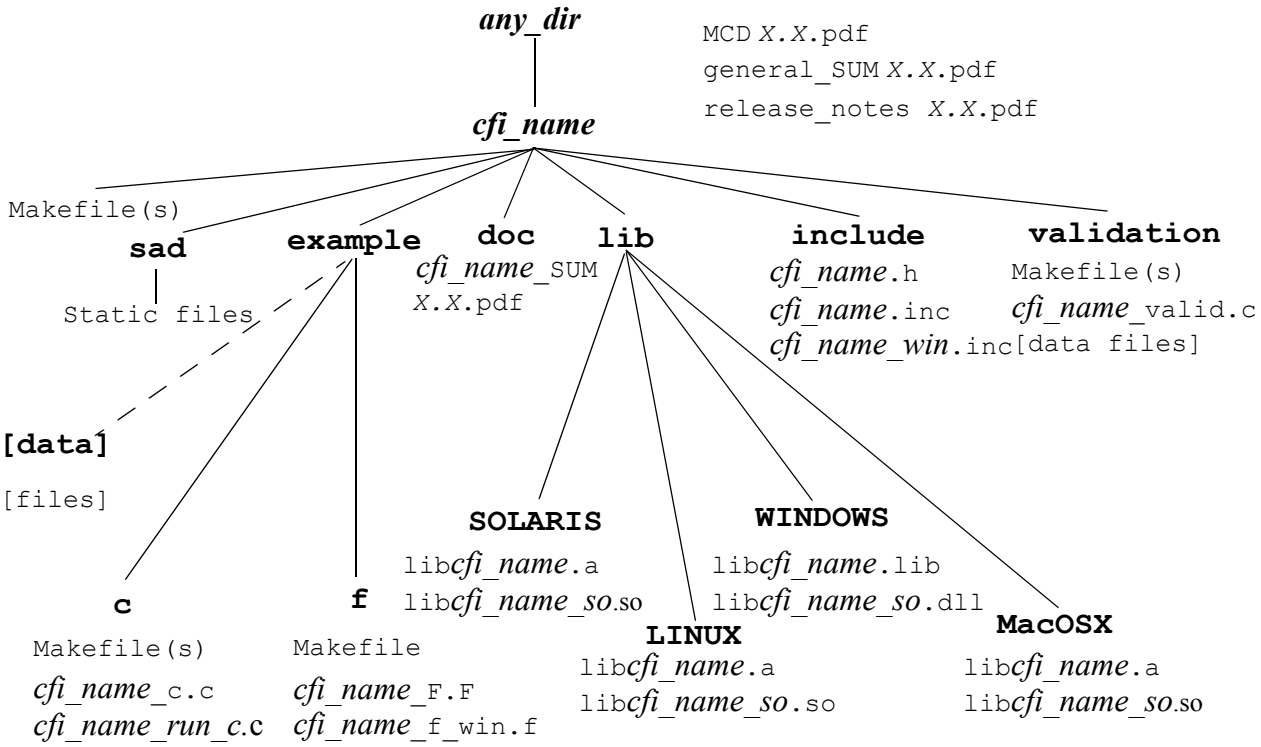


Figure 6: CFI directories structure

Makefile shall be chosen for the appropriate platform.

6.4.1 General Documents

The following PDF files are available:

- `MCD X.X.pdf`: Mission Conventions Document.
- `general_SUM X.X.pdf`: General Software User Manual.
- `release_notes X.X.pdf`: Release Notes detailing the changes and/or corrections introduced in the new release.

These are relevant to all CFIs, and must be moved manually to the directory of your choice.

6.4.2 Directory: cfi_name/doc

This directory contains the PDF files of the following documents:

- *cfi_name*`_SUM X.X.pdf`: Specific CFI Software User Manual

6.4.3 Directory: cfi_name/lib

This directory contains the CFI object libraries. There are two files per supported computer platform (static and dynamic libraries), each in a separate sub-directory:

- **SOLARIS/**`lib`*cfi_name*`.a` and `lib`*cfi_name_so*`.so`: for Sun/SOLARIS users.
- **WINDOWS/**`lib`*cfi_name*`.lib and lib`*cfi_name_so*`.dll`: for PC/Windows users.
- **LINUX/**`lib`*cfi_name*`.a and lib`*cfi_name_so*`.so`: for PC/Linux users.
- **MacOS X/**`lib`*cfi_name*`.a and lib`*cfi_name_so*`.so`: for Macintosh/MacOS X users.

6.4.4 Directory: cfi_name/include

This directory contains the include files with the function declaration for every CFI function, plus the related enumerations:

- *cfi_name*`.h`: for C users.
- *cfi_name*`.inc`: for ForTran users under Solaris platform.

6.4.5 Directory: cfi_name/validation

This directory contains the validation program and associated makefile:

- *cfi_name*`_valid.c`
- makefile files for the different allowed operative systems, i.e. Solaris, Windows 95/98/NT/2000, Linux and MacOS.

Depending on the CFI, input data files used by the validation program may be included. In such a case they can be found in the directory **example/data**. After running the validation procedure (section 6.5), other files appear.

- *cfi_name*`_valid`
- *cfi_name*`_valid.out`
- other output files depending on the library.

6.4.6 Directory: cfi_name/example

This directory contains example programs and associated makefiles. There is one sub-directory per supported language, each in a separate sub-directory:

- **c/***cfi_name*`_c.c`: for C users
- **f/***cfi_name*`_F.F`: for ForTran users under Solaris
- makefile files for the different allowed operative systems, i.e. Solaris, Windows 95/98/NT/2000.
- data files used for the validation and example drivers.

Depending on the CFI, example data files to be used with the CFI may be included in a separate **data** subdirectory.

6.4.7 File: cfi_name/Makefile(s)

These makefiles are used at the end of the installation procedure, to move the CFI files to their final destination (one per supported language).

6.4.8 Static Files: cfi_name/sad

They are operational files to be used within the SW (different from those used in the examples). These operational files belong to different libraries.

6.5 Validation Procedure

This procedure should be run to verify the proper installation of the CFI library:

1. Go to directory ***cfi_name*/validation**
2. Edit the makefile for your platform and configure it to your installation. The configuration parameters are all located at the top of the `Makefile`, with instructions on how to use them.
3. Note in particular that if the CFI requires to link with other CFIs, you will have to specify the location of those other CFI libraries. If, when installing those other CFIs, you always followed the advice given below in section 6.3, this will be easier (as all already installed include files will be grouped in one directory, and all already installed libraries as well).
4. Run the validation program using

   `make -f make.`*`OS`* where *`OS`* stands for the different allowed operative systems.

The validation program is created, executed and a validation status message printed. The message should look like:

cfi_name: ... CFI LIBRARY INSTALLATION = OK

or:

cfi_name: ... CFI LIBRARY INSTALLATION = FAILED !!!

In the latter case, check again your installation, and run the validation program again if necessary. If the message persists, report the problem (see section 6.7).

During the execution of the validation program a log file *cfi_name*`_valid.out` is also created. It can be consulted for a detailed listing of the validation run.

6.6 Examples

Three examples are provided to illustrate how the interface with the CFI functions contained in the CFI software library works with both C and ForTran, and in particular how to handle the returned errors. Proper usage of error handling and enumerations is systematically shown for each function.

Note that for C, two examples are provided. The first one is called explorer_*xxx*_c.c and the second one explorer_*xxx*_run_c.c. Both examples are similar, except for the fact that the "_run" example follows an alternative way for calling to CFI functions that uses the "run_id" variable (see section 7.3 for further details about this alternative method). The makefiles used to compile the two examples are:

- make.OS for running the explorer_*xxx*_c.c
- makerun.OS for running the explorer_*xxx*_run_c.c

where OS stands for the different allowed operative systems.

The examples should be self-explanatory. To use them, use the same procedure as for the validation program.

In a user application, the same conventions to compile and link as in the example makefiles should be followed.

Note that the examples can be used with either the static linking of the dynamic linking version of the library. To select which version, use the configuration of the Makefile (this should be self-explanatory).

Note, in particular, that when using dynamic linking libraries, proper setting of the environment must be performed at run-time. This means:

- SOLARIS/Linux/MacOS: adding to the `LD_LIBRARY_PATH` environment variable the locations of all dynamic libraries needed.

- Windows 95/98/NT/2000: adding to the `PATH` environment variable the locations of all dynamic libraries needed.

It is advised to consult your manuals for proper usage of dynamic linking libraries.

6.7 Problems Reporting

For any problems or questions, please send an e-mail to: *cfi@jw.estec.esa.nl*

7 CFI LIBRARIES USAGE

7.1 Using CFI's in a user application

To use CFIs in an application, the user must:

- include the header files provided with the CFIs (one header file per CFI)
- link the application with the CFI libraries (one library per CFI)

To avoid any naming conflicts with the user application all the software items in the CFI libraries are prefixed either *XX_* or *xx_* . *xx_* stands for the initials of the name of the CFI software library, i.e.:

- `xl_` and `XL_` for EXPLORER_LIB
- `xo_` and `XO_` for EXPLORER_ORBIT
- `xp_` and `XP_` for EXPLORER_POINTING
- `xg_` and `XG_` for EXPLORER_GEN_FILES
- `xv_` and `XV_` for EXPLORER_VISIBILITY
- `xf_` and `XF_` for EXPLORER_FILE_HANDLING
- `xc_` and `XG_` for EXPLORER_GEO_CORRECTIONS
- `xt_` and `XT_` for EXPLORER_RETRACKER

The user should avoid naming software items in the application with any of the above prefixes.

Details can be found in the specific Software User Manuals of each CFI ([XL_SUM], [XO_SUM], [XP_SUM], [XG_SUM], [XV_SUM], [XF_SUM], [XC_SUM] and [XT_SUM]).

7.2 General enumerations

It is possible to use enumeration values rather than integer values for some of the input arguments of the EXPLORER routines, as shown in the table below. The *XX* prefix is generic, that is, it must be replaced by the corresponding library prefix, e. g., *XL* for **explorer_lib**, *XO* for **explorer_orbit**, and so on.

Table 2: General purpose enumerations

Input/Output	Description	Enumeration value	long
Error handling	An error is returned by the CFI	XX_ERR	-1
	Nominal execution of the CFI	XX_OK	0
	A warning is returned by the CFI	XX_WARN	1
Satellite ID[a]	Default Satellite	XX_SAT_DEFAULT	0
	Default Satellite 1	XX_SAT_DEFAULT1	1
	Default Satellite 2	XX_SAT_DEFAULT2	2
	Default Satellite 3	XX_SAT_DEFAULT3	3
	Default Satellite 4	XX_SAT_DEFAULT4	4
	Default Satellite 5	XX_SAT_DEFAULT5	5
	Default Satellite 6	XX_SAT_DEFAULT6	6
	Default Satellite 7	XX_SAT_DEFAULT7	7

Table 2: General purpose enumerations

Input/Output	Description	Enumeration value	long
	Default Satellite 8	XX_SAT_DEFAULT8	8
	Default Satellite 9	XX_SAT_DEFAULT9	9
	ERS-1	XX_SAT_ERS1	11
	ERS-2	XX_SAT_ERS2	12
	Envisat	XX_SAT_ENVISAT	21
	MetOp-1	XX_SAT_METOP1	31
	MetOp-2	XX_SAT_METOP2	32
	MetOp-3	XX_SAT_METOP3	33
	CryoSat	XX_SAT_CRYOSAT	41
	ADM	XX_SAT_ADM	51
	GOCE	XX_SAT_GOCE	61
	SMOS	XX_SAT_SMOS	71
Cartesian coordinates	X-coordinate	XX_CART_X	0
	Y-coordinate	XX_CART_Y	1
	Z-coordinate	XX_CART_Z	2
Keplerian elements	Semi-major axis	XX_KEPL_A	0
	Eccentricity	XX_KEPL_E	1
	Inclination	XX_KEPL_I	2
	Right ascension of ascending node	XX_KEPL_RA	3
	Argument of perigee	XX_KEPL_W	4
	Mean anomaly	XX_KEPL_M	5
Time reference	Undefined time reference	XX_TIME_UNDEF	-1
	TAI time reference	XX_TIME_TAI	0
	UTC time reference	XX_TIME_UTC	1
	UT1 time reference	XX_TIME_UT1	2
	GPS time reference	XX_TIME_GPS	3
AOCS mode	Geocentric pointing	XX_AOCS_GPM	0
	Local normal pointing	XX_AOCS_LNP	1
	Yaw steering + local normal pointing	XX_AOCS_YSM	2
Time window initialisation	Initialisation from file	XX_SEL_FILE	0
	Initialisation from time	XX_SEL_TIME	1
	Initialisation from absolute orbit number	XX_SEL_ORBIT	2
	Used in xo_propag_init	XX_SEL_DEFAULT	3

a. To use a default satellite, it is necessary to initialize the satellite using the **explorer_lib** CFI function **xl_default_sat_init** (see [XL_SUM]).

Whenever available ***it is strongly recommended to use enumeration values rather than integer values***.

7.3 CFI Identifiers

7.3.1 Introduction

In most cases, CFI functions need to make use of a certain amount of internal data that characterize the system. The way to provide this data to the functions is a variable (ID) that makes a reference to the needed internal data. This variable is a structure that always contains a pointer to void, independently of the type of data (the reason why a void pointer is used is to prevent users from accessing the internal data directly).

The logical use of an ID in a program is:

1. Declaration of the ID: When declaring the ID variable it is important to set to NULL the pointer that contains. Not doing this, can make the program crash when calling any function that uses the ID.

   xx_<function>_id ID = {NULL};

   or

   xx_<function>_id ID;

   xx_<function>_id.ee_id = NULL;

2. Initialize the ID: For this issue, there are functions to initialize the internal data of the type

   xx_<function>_init (input_params, ...,
   / Output */*
   &ID);

3. Call to the functions using the ID when needed.

   xx_<function>_<xxx>(&ID, ...)

4. Close the ID when it is not needed any more. It is important to close the ID as this operation frees the allocated dynamic memory. This operation is performed with a function of the type

   xx_<function>_close(&ID, ...)

The following table shows the complete set of IDs that exits in the CFI:

Table 3: CFI Identifiers

ID	Library	Despcription
sat_id	-	Satellite identifier used for accessing generic satellite data.
xl_time_id	explorer_lib	It stores the time correlations.
xo_orbit_id	explorer_orbit	Orbit data needed for orbit calculations.
xo_propag_id	explorer_orbit	Propagation data needed for the propagation function.
xo_interpol_id	explorer_orbit	Interpolation data needed for the interpolation function.
xp_atmos_id	explorer_pointing	Atmospheric data used in target functions
xp_dem_id	explorer_pointing	It stores the Digital Elevation Model data

Table 3: CFI Identifiers

ID	Library	Despcription
xp_sat_nom_trans_id	explorer_pointing	It stores the data relative to the Satellite Nominal Ref. Frame.
xp_sat_att_trans_id	explorer_pointing	It stores the data relative to the Satellite Relative Actual Ref. frame
xp_instr_att_trans_id	explorer_pointing	It stores the data relative to the Instrument Ref. Frame.
xp_attitude_id	explorer_pointing	It stores the the results of the attitude calculation, needed for target calculations.
xp_target_id	explorer_pointing	It stores the results of the target calculation, needed for getting ancillary results

Note that the sat_id is not an ID in the same sense as the others, as it is a long value that indicates the satellite, so there is not need to construct it or destroy it.

The IDs in the CFI libraries follow a hierarchical structure, in the sense that some IDs need another IDs in order to be initialised. The figure 7 shows the hierarchy of the IDs in the CFI.

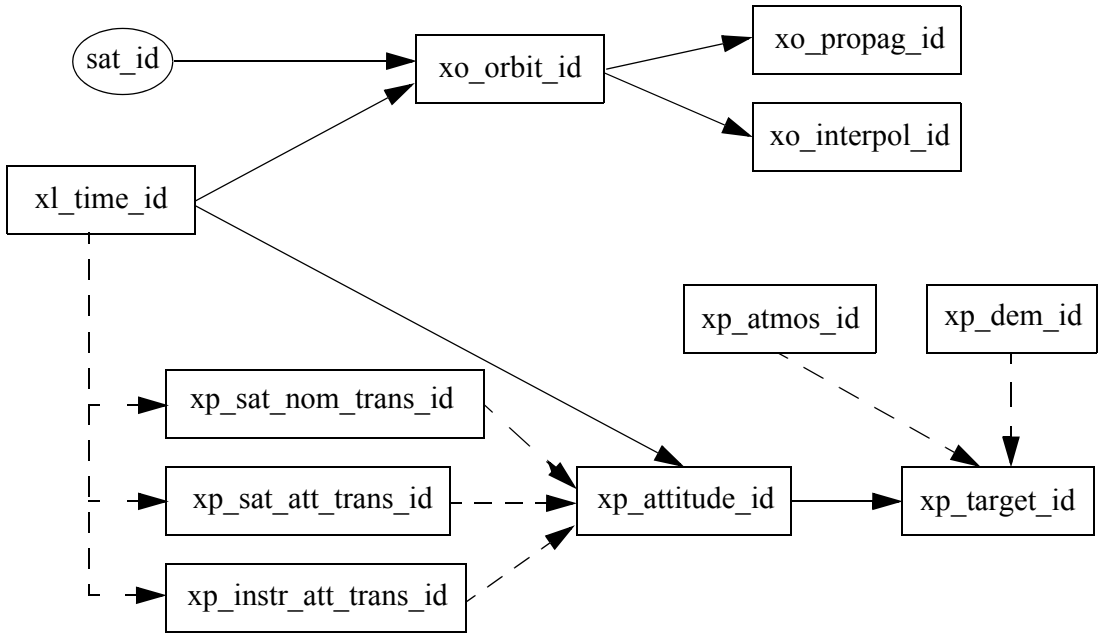
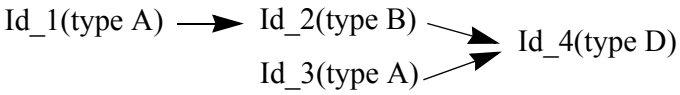


Figure 7: Hierarchical structure of the initialisation variables in the CFI.

In the figure, the plain lines mean the there is a mandatory link between the ids, while the dashed line indicates the second ID can use or not the first ID.

The previous hierarchy must be taken into account when creating and closing IDs. In general the following rules must be kept:

- An ID can be constructed if all the needed IDs are constructed.
- An ID can be closed if it is not being used by another ID. This means that the IDs must be closed in the inverse way in which they were created.
- When creating an ID, the input IDs must be consistent between them. This rule can be clarify with the following schema:



  In this case, the Id_4 can only be constructed if the Id_1 and the Id_3 are the same.

- An ID can be used for initialize several IDs. For example a xl_time_id can be used to initialize two variables of the type xo_orbit_id (let's say orbit_id_1 and orbit_id_2). If now the xl_time_id has to be closed, the orbit_id_1 and the orbit_id_2 should be closed first.

7.3.2 Function Description

The following paragraphs describes generic functions for handling IDs. There exits one function for each of the IDs of the table 3. Usage of the specific **xx_<function>_init** and **xx_<function>_close** functions is detailed in the corresponding SUMs.

7.3.2.1 xx_<function>_init_status

7.3.2.1.1 Overview

The **xx_<function>_init_status** allows to know if a CFI function has been initialized.

7.3.2.1.2 Calling Interface

The calling interface of the **xx_<function>_init_status** CFI function is the following (input parameters are underlined):

```
#include <cfi_name.h>
{
    long status;
    xx_<function>id id = {NULL};
    status = xx_<function>_init_status(&id);
}
```

For ForTran programs the declaration and calling procedure is as follows (input parameters are underlined, note that the C preprocessor must be used because of the presence of the `#include` statement):

```
#include <cfi_name.inc>
    INTEGER*4 STATUS

    STATUS = XX_<FUNCTION>_INIT_STATUS(ID)
```

7.3.2.1.3 Input Parameters

The input parameters of the **xx_<function>_init_status** CFI function are:.

Table 4: Input parameters of xx_<function>_init_status

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
id	xx_<function>_id	-	Initialisation ID	-	-

7.3.2.1.4 Output Parameters

The output parameters of the **xx_<function>_init_status** CFI function are:

Table 5: Output parameters of xx_<function>_init_status function

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
xx_<function>init_status	long	-	• 0 (false) if the input ID is not initialized. • 1 (true) if the input ID is initialized.	-	-

7.3.2.2 xx_<function>_get_mode

7.3.2.2.1 Overview

The **xx_<function>_get_mode** allows to know the mode attribute of the ID.

7.3.2.2.2 Calling Interface

The calling interface of the **xx_<function>get_mode** CFI function is the following (input parameters are underlined):

```
#include <cfi_name.h>
{
    long status;
    xx_<function>id id = {NULL};

    status = xx_<function>_get_mode(&id);
}
```

For ForTran programs the declaration and calling procedure is as follows (input parameters are underlined, note that the C preprocessor must be used because of the presence of the `#include` statement):

```
#include <cfi_name.inc>
    INTEGER*4 STATUS
```

```
STATUS = XX_<FUNCTION>_GET_MODE(ID)
```

7.3.2.2.3 Input Parameters

The input parameters of the **xx_<function>_get_mode** CFI function are:.

Table 6: Input parameters of xx_<function>_get_mode

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
id	xx_<function>_id	-	Initialisation ID	-	-

7.3.2.2.4 Output Parameters

The output parameters of the **xx_<function>_get_mode** function are:

Table 7: Output parameters of xx_<function>_get_mode function

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
xx_<function>_get_mode	long	-	Attribute mode for the initialisation ID.	-	-

7.3.3 Grouping CFI Identifiers

As the complexity of the libraries grows, more IDs are needed for the interfaces of the functions. To avoid this, another alternative for calling the CFI functions has been developed. It consists in creating another identifier (run_id) that groups several IDs. Then, instead of calling a function that uses the set of IDs as input, another one with the same name and ending with the suffix "_run" is called. This new function uses only the run_id.

A run_id is constructed incrementally, that is, firstly a run_id is created using basic IDs, and then other IDs can be added. When closing the run_id, the IDs has to removed firstly in the inverse way to which they were introduced.

Summarizing, the logical sequence for using CFI functions using a run_id is as follow:

1. Construct the needed IDs.
2. Create the run_id.
3. Add the needed IDs. to the run_id
4. Call CFI functions (those ending in "_run") using the run_id.
5. Close the run_id.
6. Close IDs.

Finally, the run_id can be considered as another ID, so the rules for constructing and closing IDs, are also applicable.

8 ERROR HANDLING

Every CFI software library follows the same error handling strategy and have exactly similar error handling functions. For this reason, the detailed description of these error handling can be found below, rather than duplicated in each specific Software User Manual.

In the following description those error handling functions are named with the generic prefix *xx_* (section 7.1).

The common error handling strategy is given below, followed by the detailed description of the error handling functions.

8.1 Functions producing an Output Status Vector

All the CFI functions belonging to the CFI software libraries, except the simpler functions of the EXPLORER_LIB CFI:

- Return a main status flag, named `status` in the code examples below.
- Produce on output a status vector of variable size, named `ierr` in the code examples below, which stores information of the returned errors and warnings.

```
long status, ierr[N];
status = xx_cfi_function(..., ierr);
```

The main status flag can take only the values:

- XX_OK (0) NOMINAL
- XX_WARN (+1) WARNING
- XX_ERR (-1) ERROR

All elements of the status vector may take values:

- Zero if nominal behaviour occurred.
- Positive if one or more warnings and no errors occurred.
- Negative if one or more errors occurred.

8.2 Functions returning an Extended Status Flag

The simpler CFI functions of the EXPLORER_LIB CFI follow a slightly different pattern, returning an extended status flag but not producing a status vector on output, i.e:

```
long ext_status;
ext_status = xx_cfi_function(...);
```

In this case the extended status flag can be:

- Zero if nominal
- Positive if one or more warnings and no errors occurred
- Negative if one or more errors occurred

In other words it is not only 0, +1 or -1.

8.3 Testing the Returned Status

To test the status of a CFI function after calling it, the user application must test for:

- `(status == XX_OK)` to detect nominal execution
- `(status >= XX_WARN)` to detect warnings
- `(status <= XX_ERR)` to detect errors

8.4 Retrieving Errors and Warnings

The errors and warnings are contained in either:

- The status vector for functions which produce it.
- The extended status flag for the simpler functions.

In both cases, the errors and warnings information is coded in an encrypted way. To translate the encrypted data into meaningful information, two error handling functions are provided with each CFI library, i.e:

- *xx*`_get_code`: to transform either the status vector or the extended status flag to a list of integer values, each one referring to a single warning or error.
- *xx*`_get_msg`: to transform either the status vector or the extended status flag to a list of error messages, each one referring to a single warning or error.

The possible error codes and messages for each CFI function are detailed in that CFI function description, in the specific Software User Manuals.

Furthermore, the user can set two error handling modes of operation.

By default, no error messages are printed when an error or a warning occurs (***silent*** mode). But if the ***verbose*** mode is set, whenever an error or warning takes place a related error message is sent automatically to the standard error output (`stderr`).

To set the error handling mode, two functions are provided with each CFI software library:

- *xx*`_silent`: sets the mode to silent for all *xx*`_`-prefixed functions.
- *xx*`_verbose`: sets the mode to verbose for all *xx*`_`-prefixed functions.

The format of an error message returned by the **xx_get_msg** function or printed automatically if the verbose mode is set, is as follows.

It begins with the name of the CFI library containing the function that returned that error or warning (e.g. `EXPLORER_POINTING`) followed by ”>>>”.

Next, depending if an error or a warning occurred, “`ERROR in`” or “`WARNING in`” appears followed by the name of the function and an explicative text associated with the error or warning returned.

Examples of error messages are:

```
EXPLORER_ORBIT >>> ERROR in xo_interpol: Wrong time on input
EXPLORER_POINTING >>> WARNING in xp_target_<mode>: Path from sat-
ellite to target occulted by the Earth
```

Finally, it is also possible for the user to send to the standard error output (`stderr`) the error messages returned by the **xx_get_msg** function, or even to send his own log messages, by calling the last error han-

dling function provided with each CFI software library:

- *xx*`_print_msg`: sends to `stderr` a list of messages

The following sections describe each CFI function.

The calling interfaces are described both for C users and ForTran users.

Input and output parameters of each CFI function are described in tables, where C programming language syntax is used to specify:

- Parameter types (e.g. long, double)
- Array sizes of N elements (e.g. param[N])
- Array element M (e.g. [M])

ForTran users should adapt the tables using ForTran syntax equivalent terms:

- Parameter types (e.g. long <=> INTEGER*4, double <=>REAL*8)
- Array sizes of N elements (e.g. param[N] <=> param (N))
- Array element M (e.g. [M] <=> (M+1))

8.5 xx_silent

8.5.1 Overview

The **xx_silent** CFI error handling function is used to set the error handling mode of the corresponding CFI to silent (i.e. for all *xx*_-prefixed functions). This is the default error handling mode.

8.5.2 Calling Interface

The calling interface of the **xx_silent** CFI error handling function is the following (input parameters are underlined):

```
#include <cfi_name.h>
{
    long status;

    status = xx_silent();
}
```

For ForTran programs the declaration and calling procedure is as follows (input parameters are underlined, note that the C preprocessor must be used because of the presence of the `#include` statement):

```
#include <cfi_name.inc>
    INTEGER*4 STATUS

    STATUS = XX_SILENT()
```

8.5.3 Input Parameters

The **xx_silent** CFI error handling function has no input parameters.

8.5.4 Output Parameters

The output parameters of the **xx_silent** CFI error handling function are:

Table 8: Output parameters of xx_silent function

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
xx_silent	long	-	Status flag	-	-1, 0, +1

8.6 xx_verbose

8.6.1 Overview

The **xx_verbose** CFI error handling function sets the error handling mode of the corresponding CFI library function to verbose (i.e. for all *xx*_-prefixed functions).

Note that when the verbose mode is on, all warnings from low-level supporting functions become visible, whereas they may be of no relevance in the context of the higher-level CFI function calls made by the user application.

This mode should be reserved for trouble-shooting. To expose the CFI functions errors and warnings, use silent mode and the **xx_print_msg** function (section 8.9).

8.6.2 Calling Interface

The calling interface of the **xx_verbose** CFI error handling function is the following (input parameters are underlined):

```
#include <cfi_name.h>
{
     long status;

     status = xx_verbose();
}
```

For ForTran programs the declaration and calling procedure is as follows (input parameters are underlined, note that the C preprocessor must be used because of the presence of the `#include` statement):

```
#include <cfi_name.inc>
     INTEGER*4 STATUS

     STATUS = XX_VERBOSE()
```

8.6.3 Input Parameters

The **xx_verbose** CFI error handling function has no input parameters.

8.6.4 Output Parameters

The output parameters of the **xx_verbose** CFI error handling function are:

Table 9: Output parameters of xx_verbose function

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
xx_verbose	long	-	Status flag	-	-1, 0, +1

8.7 xx_get_code

8.7.1 Overview

The **xx_get_code** CFI error handling function transforms the status vector or the extended status flag returned by a CFI function to an equivalent list of error codes.

This list can be used to take appropriate decisions within the user application. All possible error codes for a given CFI function are detailed with that CFI function description.

8.7.2 Calling Interface

The calling interface of the **xx_get_code** CFI error handling function is the following (input parameters are underlined):

```
#include <cfi_name.h>
{
     long func_id, n;
     long ierr[XX_MAX_ERR_VECTOR_LENGTH], ext_status;
     long vec[XX_MAX_COD], status;

     status = xx_get_code(&func_id, ierr, &n, vec);
     status = xx_get_code(&func_id, &ext_status, &n, vec);
}
```

The parameter length_error_vector must be set in each case to the length of the status vector returned by the corresponding CFI function (or a larger value).

The *XX*_MAX_COD and *XX*_MAX_ERR_VECTOR_LENGTH constants are defined in the file *cfi_name*.h.

For ForTran programs the declaration and calling procedure is as follows (input parameters are underlined, note that the C preprocessor must be used because of the presence of the #include statement):

```
#include <cfi_name.inc>
     INTEGER*4 FUNC_ID, N
     INTEGER*4 IERR(XX_MAX_ERROR_VECTOR_LENGTH), EXT_STATUS
     INTEGER*4 VEC(XX_MAX_COD), STATUS

     STATUS = XX_GET_CODE(FUNC_ID, IERR, N, VEC)
     STATUS = XX_GET_CODE(FUNC_ID, EXT_STATUS, N, VEC)
```

8.7.3 Input Parameters

The **xx_get_code** CFI error handling function has the following input parameters:

Table 10: Output parameters of xx_get_code function

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
func_id	long *	-	Function ID	-	-
ierr ext_status	long *	-	Status vector Extended status flag	-	-

8.7.4 Output Parameters

The output parameters of the **xx_get_code** CFI error handling function are:

Table 11: Output parameters of xx_get_code function

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
xx_get_code	long	-	Status flag	-	-1, 0, +1
n	long *	-	Number of error codes	-	>= 0
vec[*XX*_MAX_COD]	long	all	Error code numbers	-	-

8.8 xx_get_msg

8.8.1 Overview

The **xx_get_msg** CFI error handling function transforms the status vector or the extended status flag returned by a CFI function to an equivalent list of error messages.

This list can be used to print messages using the **xx_print_msg** function (section 8.9).

8.8.2 Calling Interface

The calling interface of the **xx_get_msg** CFI error handling function is the following (input parameters are underlined):

```
#include <cfi_name.h>
{
    long func_id, n;
    char msg[XX_MAX_COD][XX_MAX_STR];
    long ierr[XX_MAX_ERR_VECTOR_LENGTH], ext_status, status;

    status = xx_get_msg(&func_id, &ierr, &n, msg);
    status = xx_get_msg(&func_id, &ext_status, &n, vec);
}
```

The parameter length_error_vector must be set in each case to the length of the status vector returned by the corresponding CFI function (or a larger value)

The *XX*_MAX_COD, *XX*_MAX_STRING and *XX*_MAX_ERR_VECTOR_LENGTH constants are defined in the file *cfi_name*.h

For ForTran programs the declaration and calling procedure is as follows (input parameters are underlined, note that the C preprocessor must be used because of the presence of the `#include` statement):

```
#include <cfi_name.inc>

    INTEGER*4 FUNC_ID, N
    CHARACTER*256 MSG(256)
    INTEGER*4 IERR(XX_MAX_ERR_VECTOR_LENGTH), EXT_STATUS, STATUS

    STATUS = XX_GET_MSG(FUNC_ID, IERR, N, MSG)
    STATUS = XX_GET_MSG(FUNC_ID, EXT_STATUS, N, MSG)
```

8.8.3 Input Parameters

The **xx_get_msg** CFI error handling function has the following input parameters:

Table 12: Input parameters of xx_get_msg function

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
func_id	long *	-	Function ID	-	-
ierr ext_flag	long *	-	Status vector Extended status flag	-	-

8.8.4 Output Parameters

The output parameters of the **xx_get_msg** CFI error handling function are:

Table 13: Output parameters of xx_get_msg function

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
xx_get_msg	long	-	Status flag	-	-1, 0, +1
n	long *	-	Number of error codes	-	>= 0
msg [*XX*_MAX_COD][*XX*_MAX_STR]	char	all	Error code messages	-	-

8.9 xx_print_msg

8.9.1 Overview

The **xx_print_msg** CFI error handling function sends a vector of messages to `stderr`.

8.9.2 Calling Interface

The calling interface of the **xx_print_msg** CFI error handling function is the following (input parameters are underlined):

```
#include <cfi_name.h>
{
    long n;
    char msg[XX_MAX_COD][XX_MAX_STR];
    long status;

    status = xx_print_msg(&n, msg);
}
```

The *XX*_MAX_COD and *XX*_MAX_STR constants are defined in the file *cfi_name*.h

For ForTran programs the declaration and calling procedure is as follows (input parameters are underlined, note that the C preprocessor must be used because of the presence of the `#include` statement):

```
#include <cfi_name.inc>

    INTEGER*4 N
    CHARACTER*256 MSG(256)
    INTEGER*4 STATUS

    STATUS = XX_PRINT_MSG(N, MSG)
```

8.9.3 Input Parameters

The **xx_print_msg** CFI error handling function has the following input parameters:

Table 14: Input parameters of xx_print_msg function

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
n	long *	-	Number of error codes	-	>= 0
msg[*XX*_MAX_COD][*XX*_MAX_STR]	char	all	Error code message	-	-

8.9.4 Output Parameters

The output parameters of the **xx_print_msg** CFI error handling function are:

Table 15: Output parameters of xx_print_msg function

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
xx_print_msg	long	-	Status flag	-	-1, 0, +1

9 KNOWN PROBLEMS

The following precautions shall be taken into account when using the CFI software libraries:

Table 16: Known problems list

CFI library	Problem	Work around solution
	(no known problems)	