Earth Observation
Mission CFI Software

GENERAL SOFTWARE USER MANUAL

Code:	EO-MA-DMS-GS-0002
Issue:	4.2
Date:	30/01/2011

	Name	Function	Signature
Prepared by:	Mariano Sánchez-Nogales	Project Engineer	
	Juan Jose Borrego Bote	Project Engineer	
	Carlos Villanueva	Project Engineer	
	Rubén Castro	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
28760 Tres Cantos (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	Organisation	Copies

Electronic handling	
Word Processor:	OpenOffice 2.0
Archive Code:	P/SUM/DMS/01/026-006
Electronic file name:	eo-ma-dms-gs-002-21

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	
3.1	Release of CFI libraries version 3.1	13/10/04	
3.2	Release of CFI libraries version 3.2	15/11/04	
3.3	Release of CFI libraries version 3.3	11/07/05	
3.4	Release of CFI libraries version 3.4	18/11/05	
3.5	Release of CFI libraries version 3.5. New features: • Library optimization. • Thread-safe library • C99 compliance • First version for 64-bits machines in SOLARIS, LINUX and MACOS	26/05/06	
3.6	Release of CFI libraries version 3.6 New features: • Header files for linking the libraries to fortran applications. Fortran examples are also provides.	24/11/06	
3.7	Maintenance release New features: • Library version for MAC OS X on Intel (32 and 64-bits)	13/07/07	
3.7.2	Maintenance release Specific new features for the libraries	31/07/08	
4.0	Maintenance release New features: • New gcc compiler for UNIX OS: gcc 4.2 or higher. • New CFI Identifier to select different astronomical models • Numerical propagator	19/01/09	
4.1	Maintenance release Specific new features for the libraries	07/05/10	
4,2	Maintenance release New features: • New NORAD TLE designators for SMOS, GOCE and CRYOSAT2	31/01/11	

	• New format for the OSF to support curved MLST • New DEM configuration file • Support for DEM ACE2 9SECS • Support of visibility functions with TLE and precise propagator		

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 SCOPE

The Software User Manual (SUM) of the Earth Observation 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:

- EO_LIB Software User Manual, Issue 4.2 [LIB_SUM]
- EO_ORBIT Software User Manual, Issue 4.2 [ORB_SUM]
- EO_POINTING Software User Manual, Issue 4.2 [PNT_SUM]
- EO_VISIBILITY Software User Manual, Issue 4.2 [VIS_SUM]
- EO_FILE_HANDLING Software User Manual, Issue 4.2 [F_H_SUM]
- EO_DATA_HANDLING Software User Manual, Issue 4.2 [D_H_SUM]
- EXPLORER_GEO_CORRECTIONS Software User Manual, Issue 1.2 [GC_SUM]
- EXPLORER_RETRACKER Software User Manual, Issue 1.1 [RT_SUM]

2 ACRONYMS, NOMENCLATURE AND TERMINOLOGY

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)

2.3 Notes in Terminology

In order to keep compatibility with legacy CFI libraries, the Earth Observation Mission CFI Software makes use of terms that are linked with missions already or soon in the operational phase like the Earth Explorers.

This may be reflected in the rest of the document when examples of Mission CFI Software usage are proposed or description of Mission Files is given.

3 APPLICABLE AND REFERENCE DOCUMENTS

3.1 Reference Documents

[MCD] Earth Observation Mission CFI Software. Conventions Document. EO-MA-DMS-GS-0001. Issue 4.2 31/01/2011.

[MSC] Earth Observation Mission CFI Software. Mission Conventions Document. EEEO-MA- DMS-GS-0018. Issue 4.2 31/01/2011.

[F_H_SUM] Earth Observation Mission CFI Software. EO_FILE_HANDLING Software User Manual. EO-MA-DMS-GS-0008. Issue 4.2 31/01/2011.

[D_H_SUM] Earth Observation Mission CFI Software. EO_DATA_HANDLING Software User Manual. EO-MA-DMS-GS-007. Issue 4.2 31/01/2011.

[LIB_SUM] Earth Observation Mission CFI Software. EO_LIB Software User Manual. EO-MA-DMS-GS-003. Issue 4.2 31/01/2011.

[ORB_SUM] Earth Observation Mission CFI Software. EO_ORBIT Software User Manual. EO-MA-DMS-GS-004. Issue 4.2 31/01/2011.

[PNT_SUM] Earth Observation Mission CFI Software. EO_POINTING Software User Manual. EO-MA-DMS-GS-005. Issue 4.2 31/01/2011.

[VIS_SUM] Earth Observation Mission CFI Software. EO_VISIBILITY Software User Manual. EO-MA-DMS-GS-006. Issue 4.2 31/01/2011.

[GC_SUM] Earth Explorer Mission CFI Software. EXPLORER_GEO _CORRECTIONS Software User Manual. Issue 1.2. 09/08/01

[RT_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 6), 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 an 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 ([F_H_SUM], [D_H_SUM], [LIB_SUM], [ORB_SUM], [VIS_SUM], [PNT_SUM], [GC_SUM] and [RT_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 Observation Mission CFI Software is a collection of software functions performing accurate computations of mission related parameters for Earth Observation 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.

EO_FILE_HANDLING
· Read XML files
· Write XML files
· Verify XML files

EO_DATA_HANDLING
- Reading functions
  - xd_read_<file>
- Writing functions
  - xd_write_<file>
- XML Validation function
  - xd_xml_validate

EO_LIB
· Basic computations
· Time functions
· Reference frames functions
· Model configuration

EO_ORBIT
· Initialisations
  - xo_orbit_init_def
  - xo_orbit_cart_init[_precise]
  - xo_orbit_init_file[_precise]
· Propagation / Interpolation
  - xo_osv_compute
· Ancillary results
  - xo_osv_compute_extra
· Memory clean-up
  - xo_orbit_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
  - xo_osv_to_tle
· File GenerationFunctions
  - xo_gen_osf_<method>
  - xo_gen_pof
  - xo_gen_rof(_prototype)
  - xo_gen_dnf
  - xo_gen_oef
  - xo_gen_scf
  - xo_gen_tle
  - xo_check_osf
  - xo_check_oef

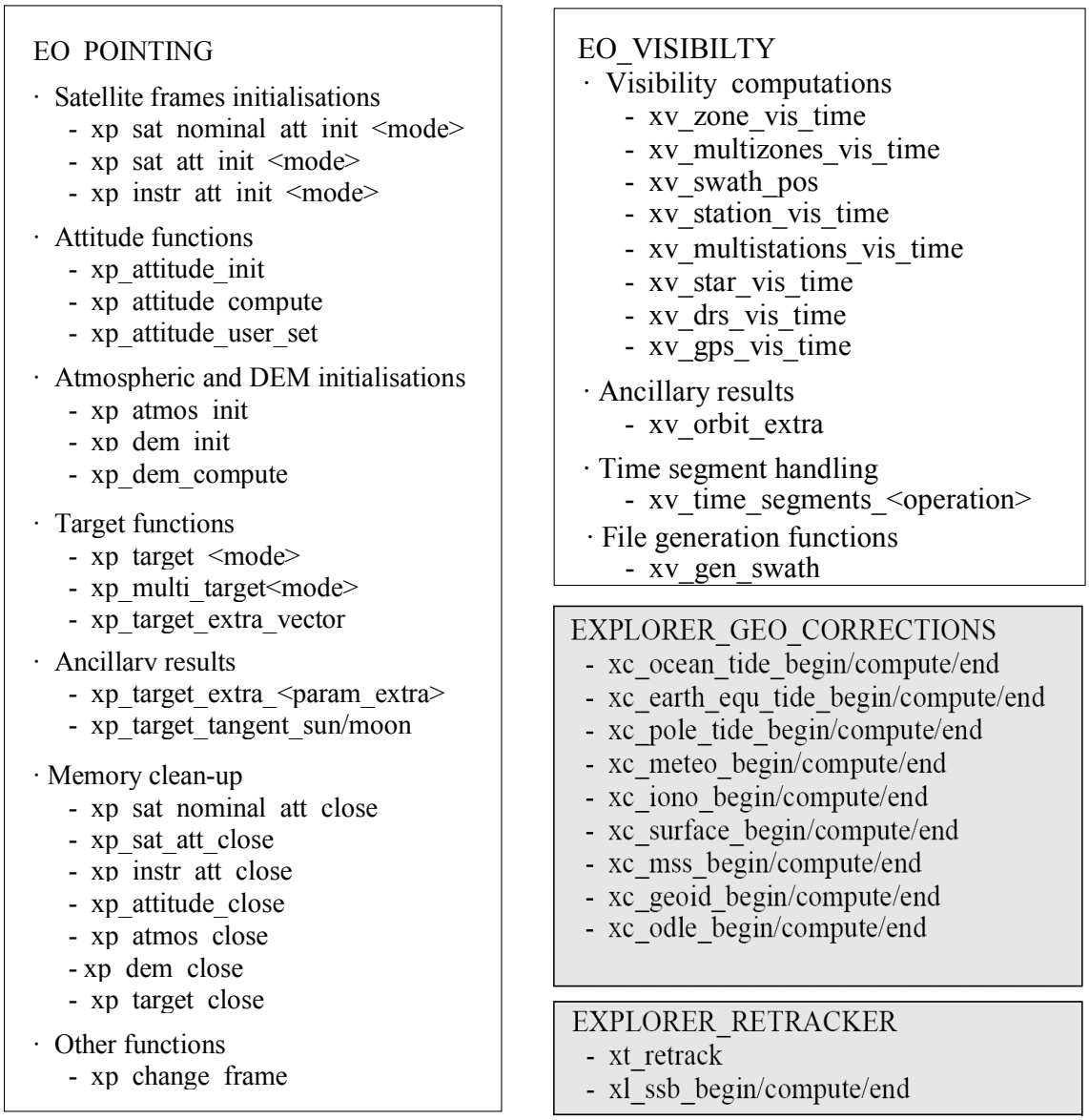


Figure 1: Earth Observation CFI Software libraries

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 4 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.

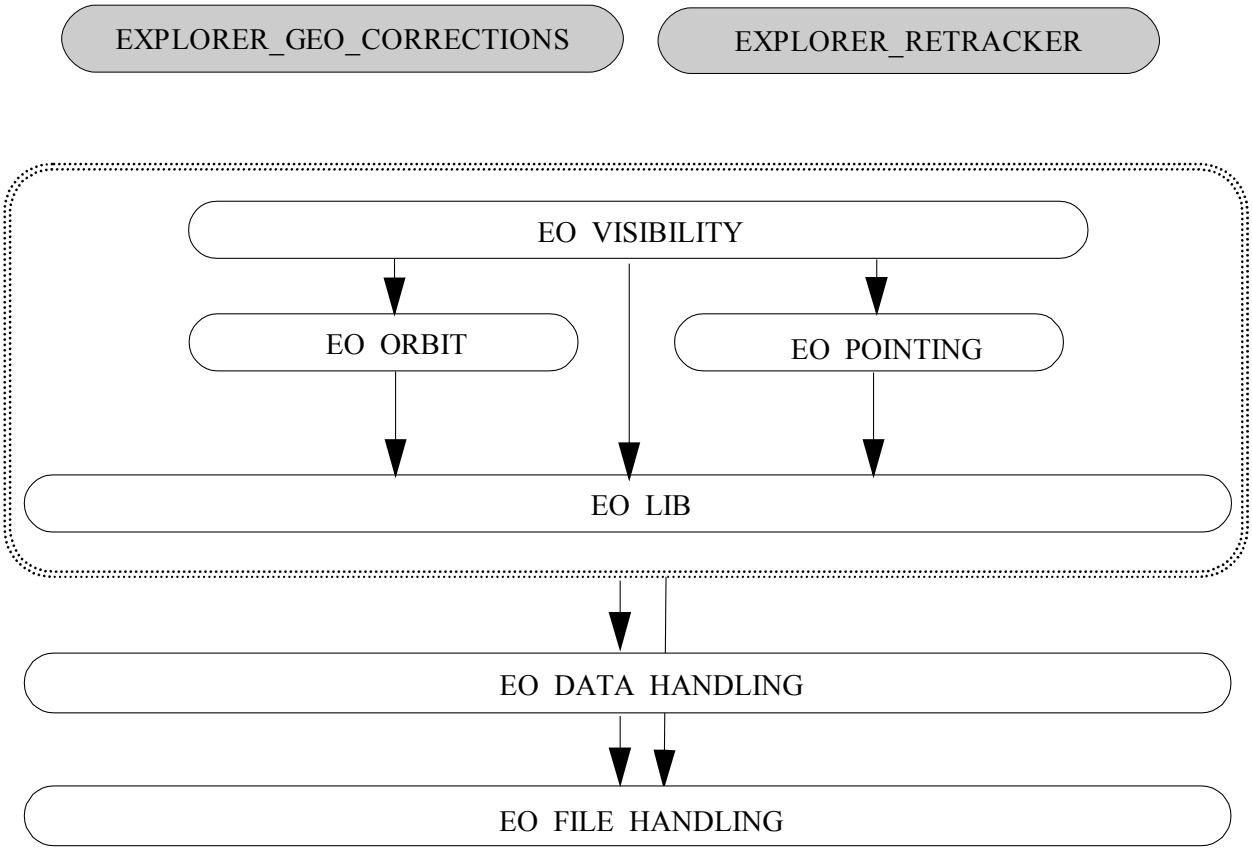


Figure 2: Earth Observation CFI Software libraries dependencies

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

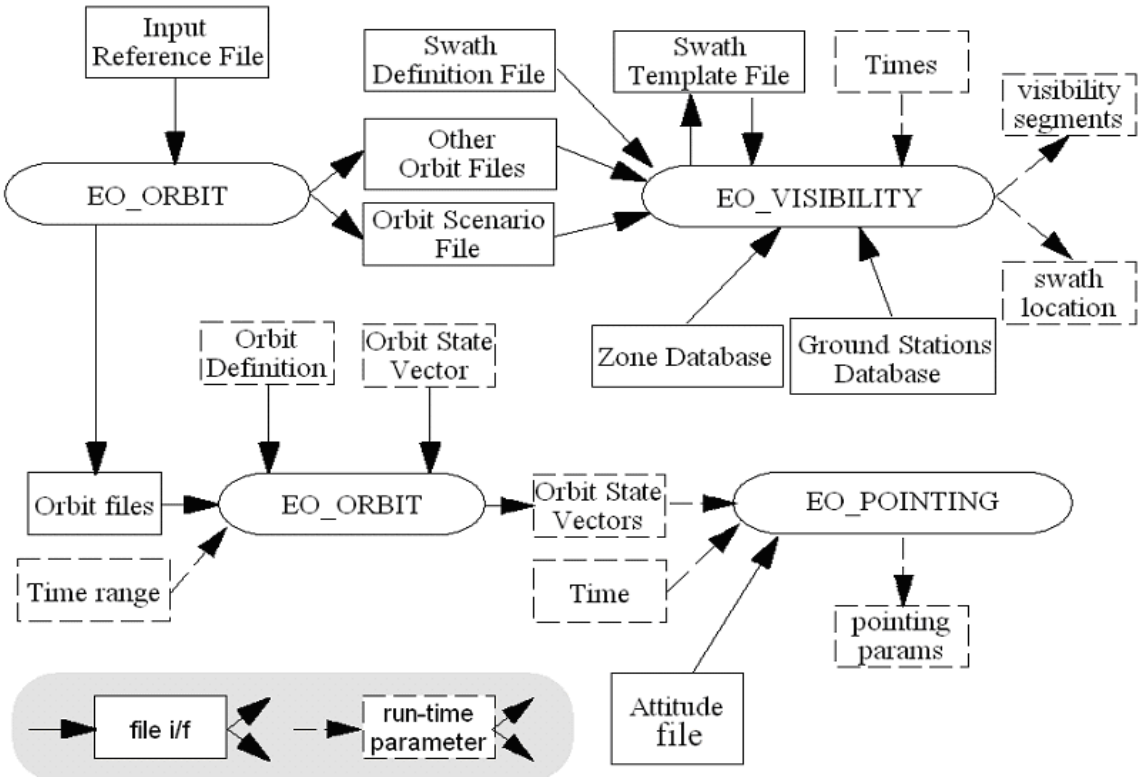


Figure 3: EO_ORBIT, POINTING and VISIBILITY data flow

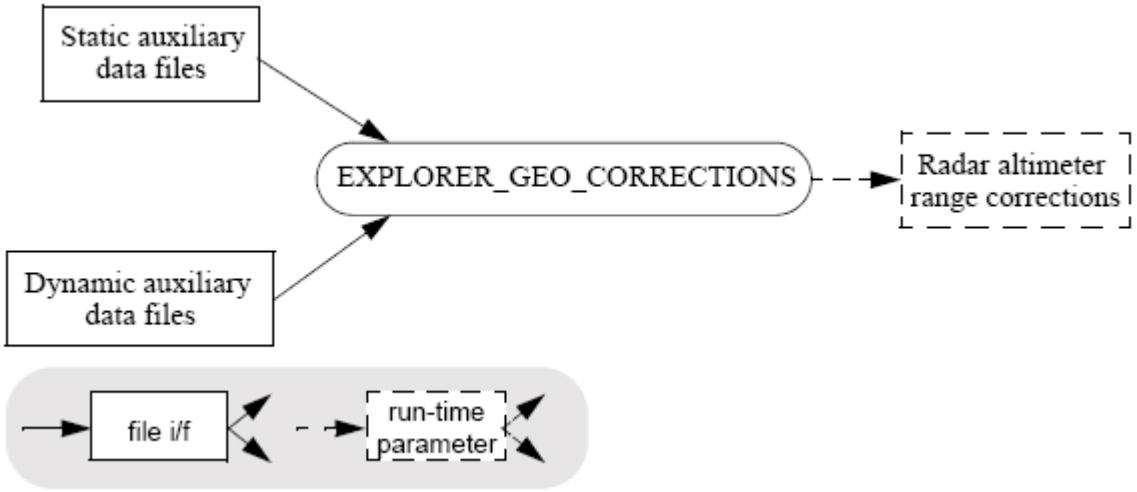


Figure 4: EXPLORER_GEO_CORRECTIONS data flow

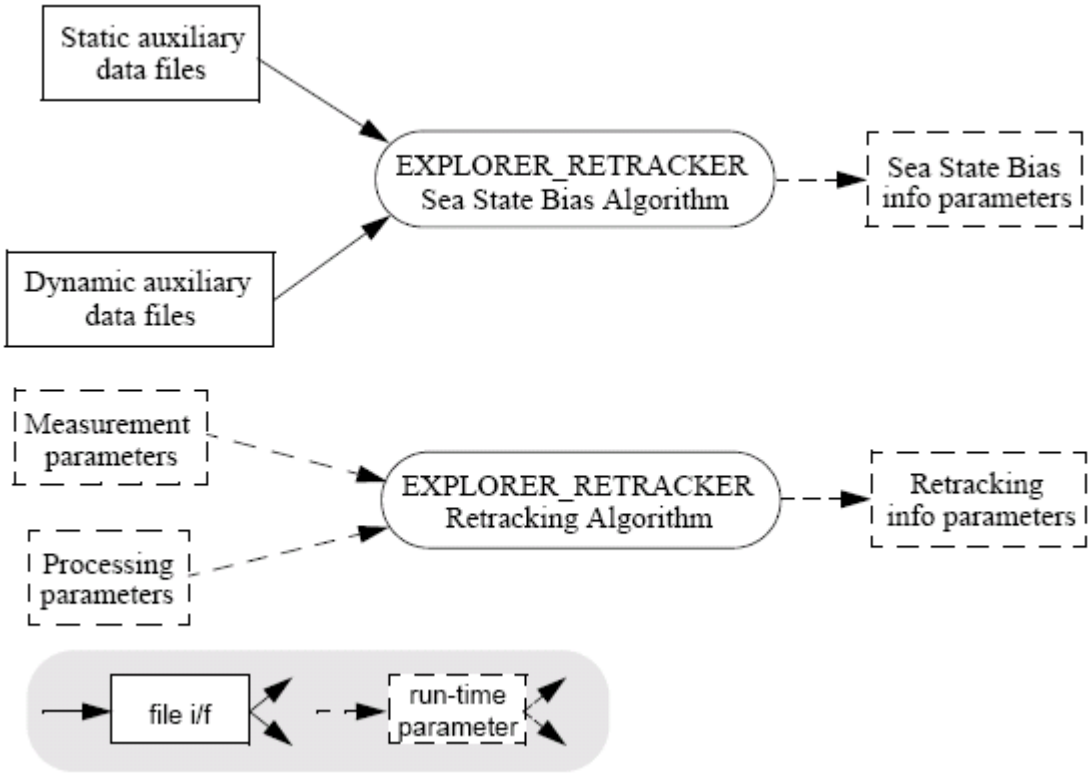


Figure 5: EXPLORER_RETRACKER data flow

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 one of the following names:

- ***explorer_file_handling*** `for EO_FILE_HANDLING library`
- ***explorer_data_handling*** `for EO_DATA_HANDLING library`
- ***explorer_lib*** `for EO_LIB library`
- ***explorer_orbit*** `for EO_ORBIT library`
- ***explorer_pointing*** `for EO_POINTING library`
- ***explorer_visibility*** `for EO_VISIBILITY library`
- ***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 (64-bits platforms)
- PC under Windows 95/98/NT/2000
- PC under Linux (32-bits and 64-bits platforms)
- Macintosh under MacOS X (32-bits)
- Macintosh with Intel processor under MacOS X (32-bits and 64-bits platforms)

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: correspondence between software libraries and operative systems

CFI Libraries	explorer_file_handling	explorer_data_handling	explorer_lib	explorer_orbit	explorer_pointing	explorer_visibility	explorer_geo_corrections	explorer_retracker
Solaris 64-bits	X	X	X	X	X	X	-	-
Windows 32-bits	X	X	X	X	X	X	-	-
Linux 32-bits	X	X	X	X	X	X	X	X
Linux 64-bits	X	X	X	X	X	X	-	-
MacOS X 32-bits	X	X	X	X	X	X	-	-
MacOS X on Intel 32-bits	X	X	X	X	X	X	-	-
MacOS X on Intel 64-bits	X	X	X	X	X	X	-	-

In order to enable the use of the libraries in C++, the following syntax is used within the headers:

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

6.1.1 Sun under Solaris 64-Bits

The source code has been compiled on a Sun SparcStation under Solaris 5.9 and using the free software *gcc* compiler.

The software requirements for C users are:

- Solaris 5.9 (or later) Operating System
- Solaris 5.9 (or later) *libm.a* (or *libm.so*) mathematical library

- *gcc* compiler version 4.2.2 (for linking the software to a C application)
- *libxml2* version 2.4.23 or later
- libpthread.so: POSIX thread library

For Fortran users, the following sofware is needed:

- gfortran compiler version 4.2.0 (for linking the software to a fortran application)
- libgfortran, libgcc_s, libgfortranbegin

The hardware requirements are:

- Sun SparcStation
- TBD Mb free of disk space (for FILE_HANDLING, DATA_HANDLING, LIB, ORBIT, POINTING and VISIBILITY libraries)
- TBD Mb RAM (for FILE_HANDLING, DATA_HANDLING, LIB, ORBIT, POINTING 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++ 8.0* compiler.

In summary, the software requirements for C users are:

- Microsoft Windows 2000 Operating System.
- *Microsoft Visual C++ 9.0* Compiler (Visual Studio 2008) for linking the software to a C application.
- *libxml2* version 2.6.20 or later (including iconv-1.9.1 and zlib-1.2.3)
- pthread.lib: POSIX thread library

For Fortran users, the following sofware is needed:

- gfortran compiler version 4.2.0 (for linking the software to a fortran application)
- libgfortran, libgcc_s, libgfortranbegin

The hardware requirements are:

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

6.1.3 PC under Linux 32-Bits

The source code has been compiled on a PC under Linux 2.6.16 (RedHat) and using the free software *gcc* compiler.

In summary, the software requirements for C users are:

- Linux 2.6.16 (RedHat)

- *gcc* compiler version 4.2.2 (for linking the software to a C application)
- *libxml2* version 2.6.23 or later
- libpthread.so: POSIX thread library
- glibc 2.4

For Fortran users, the following sofware is needed:

- gfortran compiler version 4.2.0 (for linking the software to a fortran application)
- libgfortran, libgcc_s, libgfortranbegin

The hardware requirements are:

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

6.1.4 PC under Linux 64-Bits

The source code has been compiled on a PC under Linux 2.6.16 (RedHat) and using the free software *gcc* compiler.

In summary, the software requirements for C users are:

- Linux 2.6.16 (RedHat)
- *gcc* compiler version 4.2.2(for linking the software to a C application)
- *libxml2* version 2.6.23 or later
- libpthread.so: POSIX thread library
- glibc 2.4

For Fortran users, the following sofware is needed:

- gfortran compiler version 4.2.0 (for linking the software to a fortran application)
- libgfortran, libgcc_s, libgfortranbegin

The hardware requirements are:

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

6.1.5 Macintosh under MacOS X 32-Bits

The source code has been compiled on a Macintosh (Power PC G5) under MacOS X and using gcc compiler.

In summary, the software requirements for C users are:

- Mac OS X version 10.4.6 or later
- *gcc* compiler version 4.2.1 (for linking the software to a C application)
- *libxml2* version 2.6.22. It is not possible to run the executable programs (such as the file generation executables) linking with a *libxml* version different from the version used to build those executables.
- libpthread.so: POSIX thread library

For Fortran users, the following sofware is needed:

- gfortran compiler version 4.2.0 (for linking the software to a fortran application)
- libgfortran, libgcc_s, libgfortranbegin

The hardware requirements are:

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

6.1.6 Macintosh with Intel processor under MacOS X 32-Bits

The source code has been compiled on a Macintosh (MacBook2,1) with an Intel processor under MacOS X and using the gcc compiler.

In summary, the software requirements for C users are:

- Mac OS X version 10.4.11 or later
- *gcc* compiler version 4.2.1 (for linking the software to a C application)
- *libxml2* version 2.6.16. It is not possible to run the executable programs (such as the file generation executables) linking with a *libxml* version different from the version used to build those executables.
- libpthread.so: POSIX thread library

The hardware requirements are:

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

6.1.7 Macintosh with Intel processor under MacOS X 64-Bits

The source code has been compiled on a Macintosh (MacBook2,1) with an Intel processor under MacOS X and using the gcc compiler.

In summary, the software requirements for C users are:

- Mac OS X version 10.4.11 or later

- *gcc* compiler version 4.2.1 (for linking the software to a C application)
- *libxml2* version 2.6.22. It is not possible to run the executable programs (such as the file generation executables) linking with a *libxml* version different from the version used to build those executables.
- libpthread.so: POSIX thread library

The hardware requirements are:

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

6.2 How to get the Software

The CFI software can be downloaded from the ESA EOP System Support Division Web Server:

http://eop-cfi.esa.int (main page)

From there, just follow the links until you reach the Earth Observation CFI page:

http://eop-cfi.esa.int/CFI/eo_cfi_software.html

Follow the instructions given on the page and you will be able to save the distribution file(s) on your local disk.

6.3 How to install the Software

The installation procedure details are described in the Release Note that can be found in the ESA EOP System Support Division Web Server:

http://eop-cfi.esa.int (main page)

6.4 Overview of Files and Directory Structure

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

- For *OS* = SOLARIS | SOLARIS64 |LINUX |LINUX64:

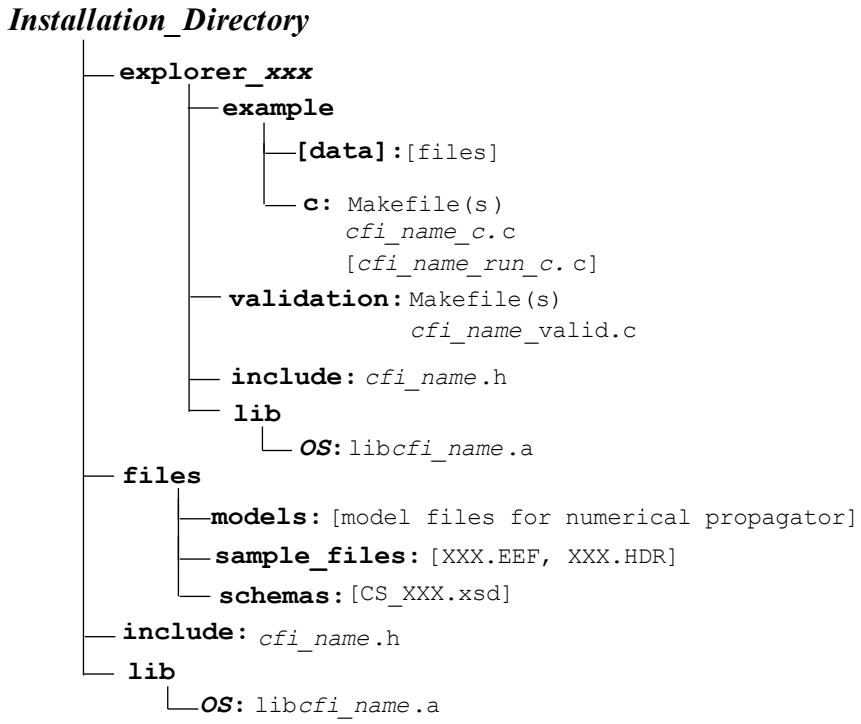


Figure 6: CFI directories structure for LINUX and SOLARIS

- For MACOS| MACOS64 |MACIN |MACIN64:

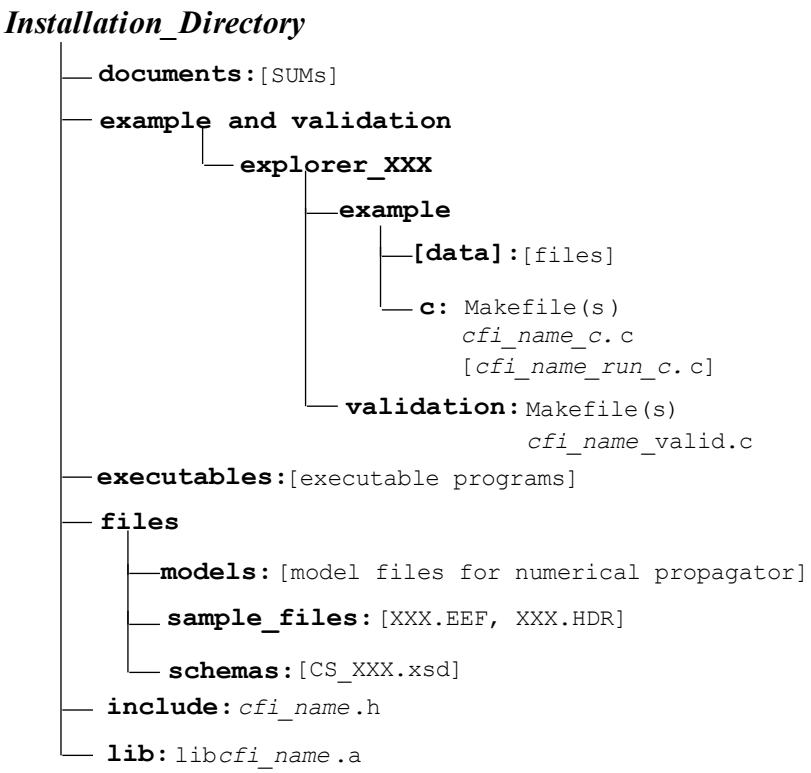


Figure 7: CFI directories structure for MAC OS

- For WINDOWS:

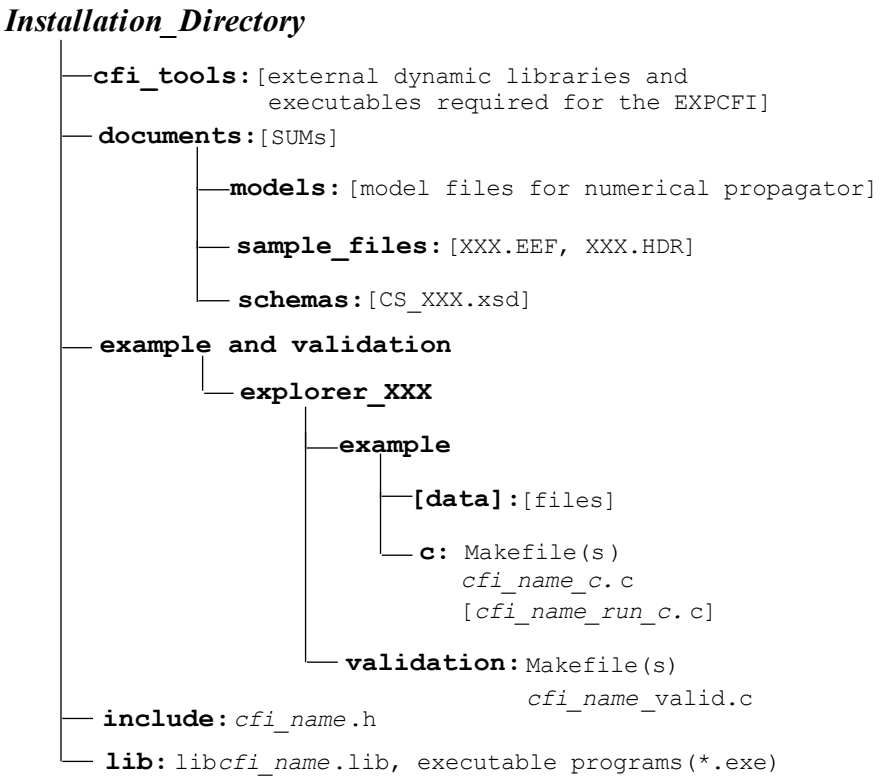


Figure 8: CFI directories structure for WINDOWS

The following subsections describe the content of the directories.

6.4.1 General Documents

The following documents are available:

- Mission Conventions Document.
- Software User Manuals.
- Release_Notes.pdf: Release Notes detailing the changes and/or corrections introduced in the new release.
- Quick Start Guide: User manual with usage cases.

For LINUX and SOLARIS OS's documents are delivered as separate packages not contained in the installation package.

For WINDOWS and MAC OS distributions, SUMS are provided with the installation package.

6.4.2 Directory: `lib`

This directory contains the CFI object libraries and executables for one computer platform.

For MAC OS distributions, the executables are in a different directory (executables)

6.4.3 Directory: `include`

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

6.4.4 Directory: `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 (or *cfi_name*_valid.exe for Windows)
- *cfi_name*_valid.*OS*.out (*OS* stands for the different allowed operative systems)
- other output files depending on the library.

6.4.5 Directory: `example`

This directory contains example programs and associated makefiles. There is 1 file per supported computer platform, each in a separate sub-directory:

- **c**/*cfi_name*_c.c: for C users
- 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.

6.4.6 Directory: *files*

This directory contains files and schemas that can be useful for the user. This directory has 3 subdirectories:

- *example_files*: this directory contains examples of Earth Observation files.
- *models*: this directory contains files that can be used as input for the numerical propagator.
- *schemas*: this directory contains schemas of the Earth Observation files.

6.5 Validation Procedure

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

1. Go to directory **`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.
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.`*`OS`*`.out` (*OS* stands for the different allowed operative systems) 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 EO_LIB
- xo_ and XO_ for EO_ORBIT
- xp_ and XP_ for EO_POINTING
- xv_ and XV_ for EO_VISIBILITY
- xf_ and XF_ for EO_FILE_HANDLING
- xd_ and XD_ for EO_DATA_HANDLING
- xc_ and XC_ 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 ([F_H_SUM], [D_H_SUM], [LIB_SUM], [ORB_SUM], [VIS_SUM], [PNT_SUM], [GC_SUM] and [RT_SUM]).

7.2 General enumerations

It is possible to use enumeration values rather than integer values for some of the input arguments of the EOCFI 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 **EO_LIB**, *XO* for EO_ORBIT, and so on.

Table 2: General purpose Enumerations

Input	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

Input	Description	Enumeration value	Long
Satellite ID[1]	Default Satellite 0	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
	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
	Terrasar	XX_SAT_TERRASAR	81
	EarthCARE	XX_SAT_EARTHCARE	91
	Swarm-A	XX_SAT_SWARM_A	101
	Swarm-B	XX_SAT_SWARM_B	102
	Swarm-C	XX_SAT_SWARM_C	103
	Sentinel-1A	XX_SAT_SENTINEL_1A	110
	Sentinel-1B	XX_SAT_SENTINEL_1B	111
	Sentinel-2	XX_SAT_SENTINEL_2	112
	Sentinel-3	XX_SAT_SENTINEL_3	113
	Seosat	XX_SAT_SEOSAT	120
	Sentinel-1C	XX_SAT_SENTINEL_1C	125
	Sentinel-2A	XX_SAT_SENTINEL_2A	126
	Sentinel-2B	XX_SAT_SENTINEL_2B	127
	Sentinel-2C	XX_SAT_SENTINEL_2C	128
	Sentinel-3A	XX_SAT_SENTINEL_3A	129
	Sentinel-3B	XX_SAT_SENTINEL_3B	130
	Sentinel-3C	XX_SAT_SENTINEL_3C	131
	Generic Satellite	XX_SAT_GENERIC	200

Input	Description	Enumeration value	Long
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	XX_TIME_UNDEF	-1
	TAI	XX_TIME_TAI	0
	UTC	XX_TIME_UTC	1
	UT1	XX_TIME_UT1	2
	GPS	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

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

1 To use a default satellite, it is necessary to initialize the satellite using the EO_LIB CFI function xl_default_sat_init (see [LIB_SUM]).

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	Description
sat_id	-	Satellite identifier
xl_model_ id	EO_LIB	It stores the data about the models to be used for astronomical models
xl_time_id	EO_LIB	It stores the time correlations
xo_orbit_i d	EO_ORBIT	It stores the orbit data needed for orbit calculations

xp_atmos_ id	EO_POINTING	It stores the atmospheric data used in target functions
xp_dem_id	EO_POINTING	It stores the Digital Elevation Model data used in target functions
xp_sat_no m_trans_id	EO_POINTING	It stores the Satellite Nominal Attitude Ref. Frame data used in attitude functions
xp_sat_att _trans_id	EO_POINTING	It stores the Satellite Attitude Ref. Frame data used in attitude functions
xp_instr_trans_id	EO_POINTING	It stores the Instrument Ref. Frame data used in attitude functions
xp_attitude_id	EO_POINTING	It stores the results of the attitude calculation used in target functions
xp_target_ id	EO_POINTING	It stores the results of the target calculation, needed to get ancillary results
run_id	all	It stores a set of Ids.

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 9 shows the hierarchy of the IDs in the CFI.

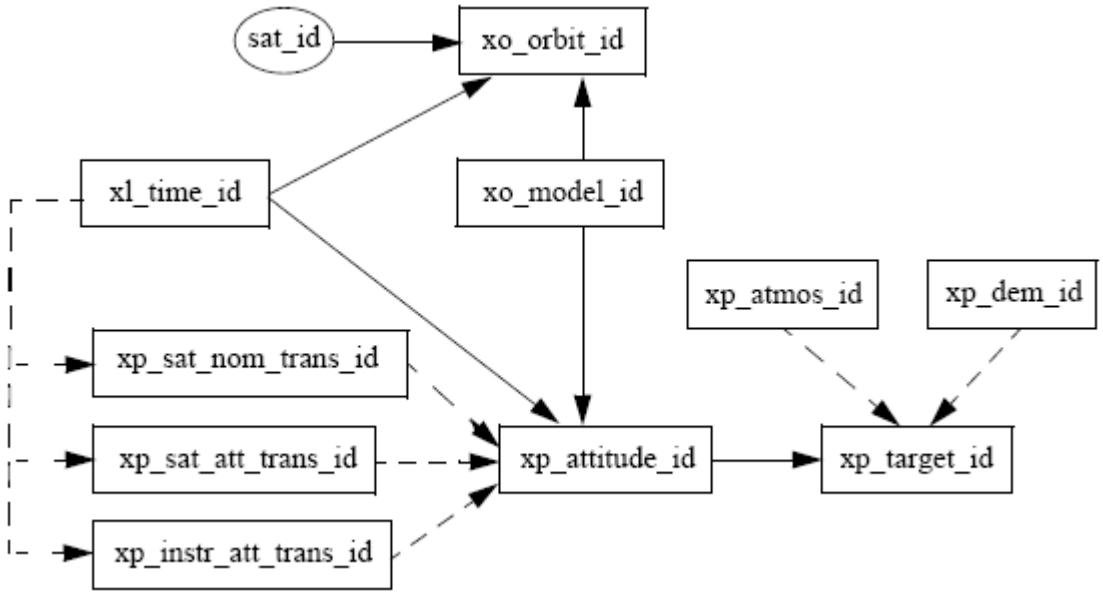


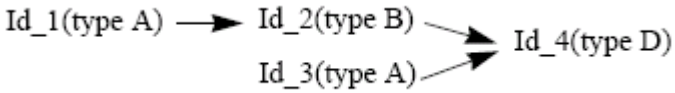
Figure 9: Hierarchical structure of the initialisation variables in the CFI

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 to 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

Each ID have a set of functions for handling the data that it stores. These functions allow the user to access the content of the ID and even to change it. In this case, it has to be into account that the changes also affect the IDs that depend on the changed ID.

Some of the handling functions are similar for all IDs and they are presented in the following sections. Other functions are specific for each ID and its usage has been detailed in the corresponding SUM.

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);
}
```

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

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_sat_id

7.3.2.2.1 Overview

The **xx_<function>_get_sat_id** allows to know the satellite for which the ID is used. Note that some IDs are not initialized for a specific satellite, i.e., the ID does not depend on the satellite. In such a case the returned value is -1.

7.3.2.2.2 Calling Interface

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

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

7.3.2.2.3 Input Parameters

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

Table 6: Input parameters of xx_<function>_get_sat_id

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_sat_id** function are:

Table 7: Output parameters of xx_<function>_get_sat_id

C name	C type	Array element	Description (Reference)	Unit (Format)	Allowed Range
xx_<function>_get_sat_id	long	-	Satellite ID.	-	Returned values according to “Satellite ID” in table 1. -1 if the ID is not linked to any satellite.

7.3.2.3 xx_<function>_get_mode

7.3.2.3.1 Overview

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

7.3.2.3.2 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);
}
```

7.3.2.3.3 Input Parameters

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

Table 8: 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	-	-

7.3.2.3.4 Output Parameters

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

Table 9: Output parameters of xx_<function>_get_mode

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.

7.4 Runtime Performances

The runtime performances depend a lot on the type of machine where the program is run. The performances for the CFI functions have been measured for every function in the following machines(see the other Earth Observation SUMs for details):

- **SOLARIS 32-bits:** Sun Sparc Station Ultra-60. 2 x 450 MHz.
- **SOLARIS 64-bits:** Sun Work Station Sun-Blade-2500. 1280 MHz.
- **Linux 32-bits:** Pentium III 2 x 803 MHz
- **Linux 64-bits**: Dual Core AMD Opteron. 2 x 2193 MHz.

7.5 Checking library integrity

The Earth Observation CFI provides two methods (functions and executable programs) for checking the compatibility of the libraries that are defined in the following subsections:

7.5.1 xx_check_library_version function

Every library in the EOCFI contains a function called ***xx*_check_library_version** that allows to know if a set of EOCFI libraries are compatible between them (***xx*_** stands for the suffix of the library as described in section 7.1).

The function checks the versions of the library to which the function belongs and the libraries that it depends on. For example, ***xl*_check_library_version** will check the compatibility between EO_FILE_HANDLING, EO_DATA_HANDLING and EO_LIB libraries.

Note also that the libraries checked are those that are used for linking the program that contains the function call.

The calling interface of the ***xx*_check_library_version** CFI function is the following:

```
#include <cfi_name.h>
{
     long status;
     status = xx_check_library_init_status();
}
```

The functions do not requiere any input parameters and the only output is the status. This could be:.

- status = 0: if the libraries are compatible.
- status = -1: if the libraries are not compatible.

The function also reports in the standard output the library versions and the found incompatibilities (if any).

Finally an alias has been defined for the name of the function called: **expcfi_check_libs**. This alias can be called in the same way than the function:

```
status = expcfi_check_libs();
```

when doing this, the chosen function is the one at the higher level in the dependency tree (see Figure 3). In other words, if the program is linked with all the EOCFI libraries, then it call **xv_check_library_version**, if the program is linked with EO_FILE_HANDLING, EO_DATA_HANDLING and EO_LIB libraries, then the alias will be a call to **xl_check_library_version**.

The user should be warned that if the program is linked with EO_POINTING and EO_ORBIT libraries, then the alias will call **xp_check_library_version** and then the EO_ORBIT library will not be checked (note that those libraries do not depend on each other)

7.5.2 expcfi_check_libs executable program

This executable program can be called from a UNIX shell as follows:

expcfi_check_libs [{**-dir**| "directory name" (current directory by default)
-lib}] "library name" (to get the library version.
Compatibility between libraries are not checked)

[**-help**]

Note that:

- Order of parameters does not matter.
- Bracketed parameters are not mandatory.
- Options between curly brackets and separated by a vertical bar are mutually exclusive.

The executable program checks the compatibility of the libraries that are located in the directory "dir". For the correct execution of the program it is required that:

- All EOCFI libraries are located in the directory "dir".
- The current directory (from which the program is called) has writing permissions.

Important: For WINDOWS OS, this executable program fails if the system cannot find the tool "strings". This tool is provided freely in a collection of tools called GNU binutils. More information about GNU binutils can be get in: http://www.gnu.org/software/binutils/

Examples: Let's suppose that the libraries are stored in the directory "*./expcfi_dir*":

A)

expcfi_check_libs -**lib** ./expcfi_dir/libexplorer_visibility.a

The output would be:

```
explorer_visibility version = 3.7
Current version of explorer_visibility is compatible with explorer_file_handling
v3.7
Current version of explorer_visibility is compatible with explorer_data_handling
v3.7
Current version of explorer_visibility is compatible with explorer_lib v3.7
Current version of explorer_visibility is compatible with explorer_orbit v3.7
Current version of explorer_visibility is compatible with explorer_pointing v3.7
```

B)

expcfi_check_libs -**dir** ./expcfi_dir

The output would be:

```
Current Earth Explorer CFI libraries:
-------------------------------------
explorer_file_handling version = 3.7
explorer_data_handling version = 3.7
explorer_lib version = 3.7
explorer_orbit version = 3.7
explorer_pointing version = 3.7
explorer_visibility version = 3.7
EXPCFI library integrity check = OK
```

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 EO_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 EO_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 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.

```
<LIBRARY NAME> >>> ERROR in <function name>: <error description>
<LIBRARY NAME> >>> WARNING in <function name>: <error description>
```

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 handling 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 for C 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])

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();
}
```

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 10: 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();
}
```

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 11: 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.

8.7.3 Input Parameters

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

Table 12: Input 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 13: 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

8.8.3 Input Parameters

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

Table 14: 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 15: 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

8.9.3 Input Parameters

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

Table 16: 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 17: 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 18: Known problems list

CFI library	Problem	Work around solution
-	-	-