Earth Observation
Mission CFI Software

GENERAL SOFTWARE USER MANUAL

Code:	EO-MA-DMS-GS-0002
Issue:	4.28
Date:	13/12/2024

	Name	Function	Signature
Prepared by:	Davide Aiello	Project Engineer	
Checked by:	Davide Aiello	Project Engineer	
Approved by:	Davide Aiello	Project Engineer	

DEIMOS Space S.L.U.
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:	4000102614/1O/NL/FF/ef	Internal	
		Public	
Contract Issuer:	ESA / ESTEC	Industry	X
		Confidential	

External Distribution		
Name	Organisation	Copies

Electronic handling	
Word Processor:	LibreOffice 5.2.3.3
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,	31/01/11	

	GOCE and CRYOSAT2 • 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		
4.3	Maintenance release Specific new features for the libraries		
4.5	Maintenance release • Support for GEO orbits • Support for DEM memory cache		
4.5	Maintenance release • Support for longitude of fields in XML files larger than 512 characters. • New function to configure position/attitude interpolator according to user needs. • New function to transform between heights relative to the ellipsoid and the geoid. • New functions to get extra results of lists of targets. • New DEM algorithm using maximum heights. • Number of harmonics used in DEM geoid computations are configurable.	01/03/13	
4.6	Maintenance release • Support for new Attitude Definition File. • Implemented SDP4 TLE propagator • Executable to generate TLE files. • Fitting method to compute TE in xo_osv_to_tle function. • New function xp_attitude_define • Internal Improvements for runtime performance in DEM computations. • Support for new swath ID. • New functions: xv_zonevistime_compute, xv_stationvistime_compute, xv_swathpos_compute, xv_timesegments_xxx_compute	03/10/13	
4.7	Mantenace release.	03/28/14	
4.8	Mantenace release. • Added support for DEM GETASSE v3.0 • Added support for dataset GDEM v2 • New function to add stylesheet to files: xd_xslt_add • New reference frame: Earth Fixed non rotating (intermediate step to the Greenwich reference frame) • New Sun model to take into account Sun light travel time. • New function for quaternions	29/10/2014	

	interpolation: xl_quaternions_interpol • Added support for Earth Fixed input in initialization of satellite nominal attitude with harmonics.		
4.9	Mantenace release. New feature: • Support for Orbit Ephemeris Message files	23/04/2015	
4.10	Maintenance release New features: • Support for DEM ACE2 30 secs • New diagnostic function for orbit files with state vectors: xd_orbit_file_diagnostics • Change of interface in functions xd_read_oem and xd_read_sp3 • Target functions: possibility of considering light travel time in target computation • Run-time improvements in target functions	29/10/15	
4.11	Maintenance release New features: • Support for DEM ACE2 3 secs • Support for BIOMASS, SENTINEL-5, and SAOCOM-CS satellites • New functions xp_gen_attitude_data and xp_gen_attitude_file	15/04/2016	
4.12	Maintenance release New features: • Support for File Format Standard 3 • Extrapolation algorithm for quaternions • New functions xo_osv_check and xv_set_sc_visibility_step	03/11/2016	
4.13	Maintenance release	05/04/2017	
4.14	Maintenance release New features: • Added support for Jason-CS Doris • New functions for CUC time managing: xl_time_cuc_to_processing xl_time_processing_to_cuc • Support for FLEX satellite • Support for MetOp-SG attitude law • New function xp_free_target_id_data • Support for swath defined by incidence angles • Updated generation of SCF	16/11/2017	
4.15	Maintenance release New features: • Refactored code in libraries explorer_file_handling, explorer_data_handling, explorer_lib	20/04/2018	
4.16	Maintenance release New features:	09/11/2018	

	• New customized EOCFI version for IOS. • Refactored code in explorer_orbit library • TLE support for SENTINEL-5P, SENTINEL-3B, Aeolus • Support for ACE2 DEM 5 min • Improved runtime in time conversions. • New functions: xo_orbit_id_check, xv_celestial_body_vistime, xv_compute_aoi		
4.17	Maintenance release New features: • DEM configurable raster • New functions xp_get_dem_cell_value and xp_get_dem_cell_geod	10/05/2019	
4.18	Maintenance release New features: • Support for TanDEM-X 90m DEM • Support for new IERS bulletin B format	08/11/2019	
4.19	Maintenance release New features: • Support for ASTER GDEM V3 • Support Reference_Frame tag in Attitude Angles files • Added support for XML Orbit Ephemeris Message Files	29/05/2020	
4.20	Maintenance release New features: • Added support for CCSDS AEM TXT/XML files • Added support for LEO satellites with SP3 files • Added support for new missions: Sentinel 6, CIMR, ROSEL, CHIME, CRISTAL, CO2M, LSTM, FORUM • Modernized generation of Swath Control Files	30/11/2020	
4.21	Maintenance release New features: • Enabled use and customization of satellite SP3 identifiers • Optimisation of EOCFI library size	23/06/2021	
4.22	Maintenance release New feature: • Enabled support for AEM with custom REF_FRAME used by EUM	22/11/2021	
4.23	Added support for new mission: TRUTHS	23/06/2022	
4.24	Maintenance release New features: • Added support for Windows 10	29/11/2022	

4.25	Maintenance release New features: - Orbit initialization with new Orbit Scenario files with ANX longitude drift parameters	10/05/2023	
4.26	Maintenance release New features: - API support for longitude drift	31/10/2023	
4.27	Maintenance release New features: - Use NORAG catalog number to identy TLE records instead of name	07/06/2024	
4.28	Maintenance release	13/12/2024	

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 4.28 [LIB_SUM]
EO_ORBIT Software User Manual, Issue 4.28 [ORB_SUM]
EO_POINTING Software User Manual, Issue 4.28 [PNT_SUM]
EO_VISIBILITY Software User Manual, Issue 4.28 [VIS_SUM]
EO_FILE_HANDLING Software User Manual, Issue 4.28 [F_H_SUM]
EO_DATA_HANDLING Software User Manual, Issue 4.28 [D_H_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.

[F_H_SUM] Earth Observation Mission CFI Software. EO_FILE_HANDLING Software User Manual. EO-MA-DMS-GS-0008.

[D_H_SUM] Earth Observation Mission CFI Software. EO_DATA_HANDLING Software User Manual. EO-MA-DMS-GS-007.

[LIB_SUM] Earth Observation Mission CFI Software. EO_LIB Software User Manual. EO-MA-DMS-GS-003.

[ORB_SUM] Earth Observation Mission CFI Software. EO_ORBIT Software User Manual. EO-MA-DMS-GS-004.

[PNT_SUM] Earth Observation Mission CFI Software. EO_POINTING Software User Manual. EO-MA-DMS-GS-005.

[VIS_SUM] Earth Observation Mission CFI Software. EO_VISIBILITY Software User Manual. EO-MA-DMS-GS-006.

The latest applicable version of [MCD], [F_H_SUM], [D_H_SUM], [LIB_SUM], [ORB_SUM], [PNT_SUM], [VIS_SUM] is v4.28 and can be found at: http://eop-cfi.esa.int/REPO/PUBLIC/DOCUMENTATION/CFI/EOCFI/BRANCH_4X/

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], [PNT_SUM] and [VIS_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.

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 functions
  xd_xml_validate
- Configuration of interpolators (decimation)
  xd_attitude_file_decimate
  xd_orbit_file_decimate
- Addition of stylesheets
  xd_xslt_add
- Orbit data diagnostics:
  xd_orbit_file_diagnostics
- Selection of File Format Standard:
  xd_set_file_format_standard_version

EO_LIB
- Basic computations
- Time functions
- Reference frames functions
- Model configuration
- Celestial body ephemeris

EO_ORBIT
- Initialisations
  xo_orbit_cart_init(_precise)
  xo_orbit_id_init
  xo_orbit_init_def_2
  xo_orbit_init_file(_precise)
  xo_orbit_init_geo
- Propagation/interpolation
  xo_osv_compute
- Ancillary results
  xo_osv_compute_extra
- Memory clean-up
  xo_orbit_close
- Orbit-related computations
  xo_check_oef
  xo_check_osf
  xo_orbit_abs_from_phase
  xo_orbit_abs_from_rel
  xo_orbit_data_filter
  xo_orbit_id_change
  xo_orbit_info
  xo_orbit_rel_from_abs
  xo_orbit_to_time
  xo_osv_check
  xo_osv_to_tle
  xo_position_on_orbit_to_time
  xo_time_to_orbit
- File generation
  xo_gen_dnf
  xo_gen_osf_<method>
  xo_gen_pof
  xo_gen_rof(_prototype)
  xo_gen_tle

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

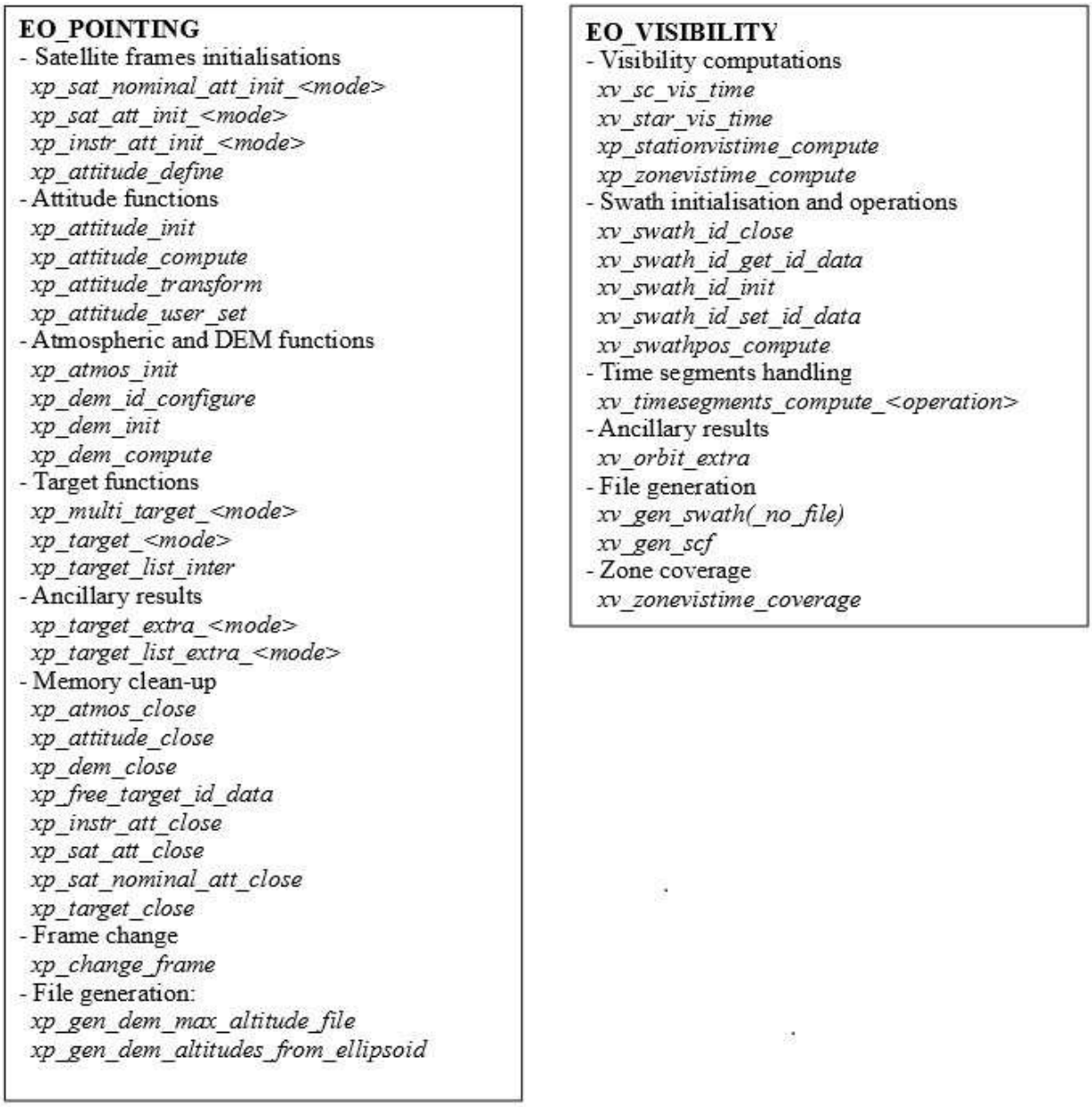


Figure 1: Earth Observation CFI Software libraries

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 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.

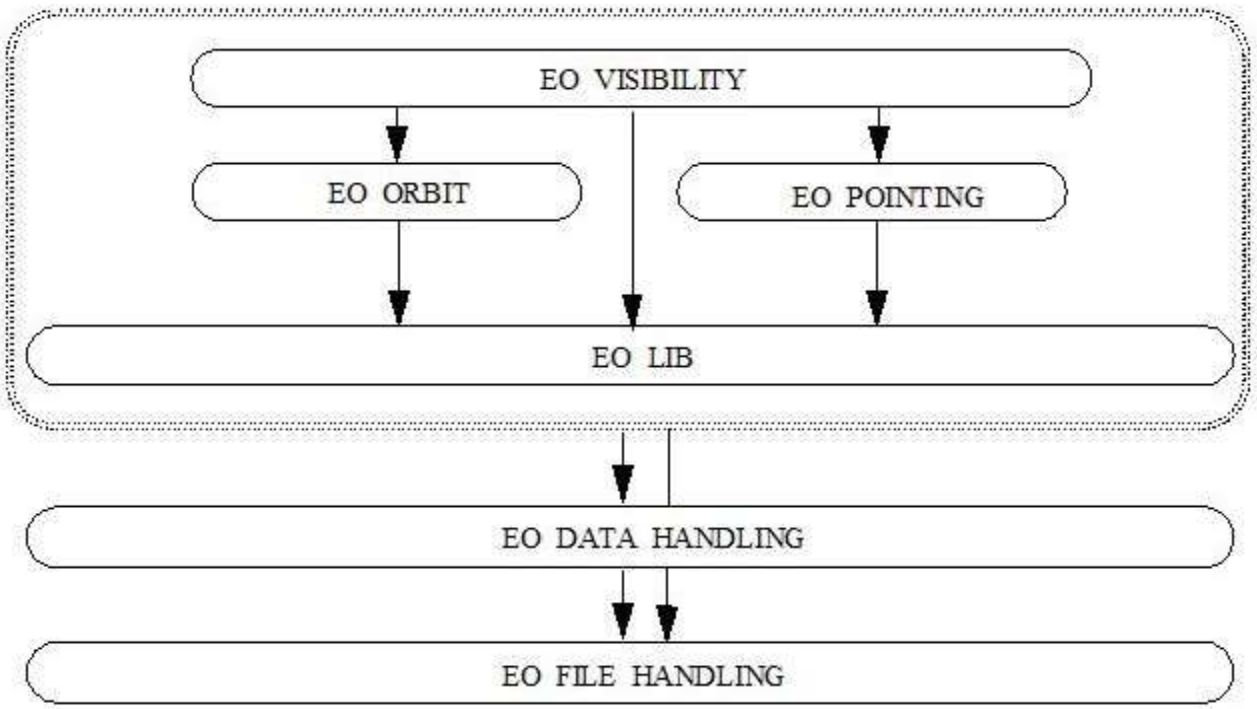


Figure 2: Earth Observation CFI Software libraries dependencies

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

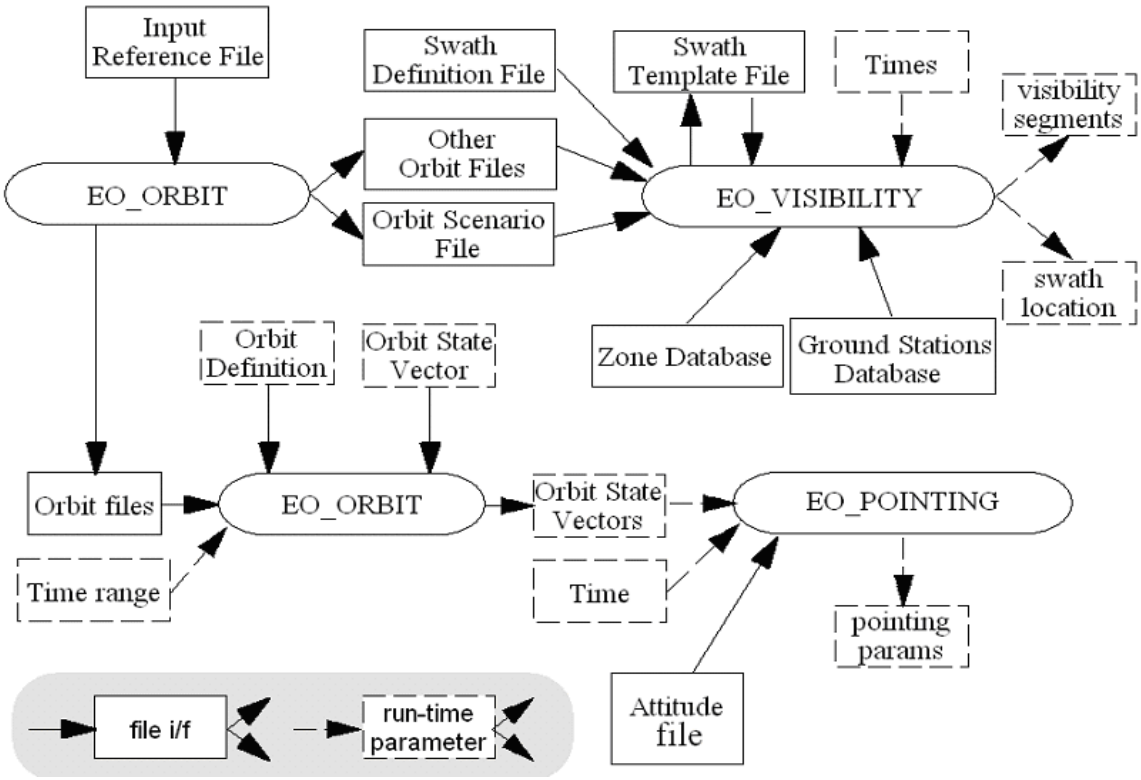


Figure 3: EO_ORBIT, POINTING and VISIBILITY data flow

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

6.1 Usage Requirements

Each CFI software library is distributed as an object code callable from C and C++. The object code is completely system dependent. In this sense, different computer platforms are supported:
PC under Windows 7 or 10
PC under Linux (64-bits platforms)
Macintosh with Intel processor under MacOS X (64-bits platform)
Macintosh with ARM processor under MacOS X (64-bits platform)

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
Windows 64-bits	X	X	X	X	X	X
Linux 64-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
```

Note: Fortran is no longer officially supported. Fortran is not tested.

6.1.1 PC under Windows 64-Bits

The source code has been compiled on a x64 (64-bit) based PC under Microsoft Windows 7 and 10 and using the software *Microsoft Visual Studio*.

In summary, the software requirements for C users are:
Microsoft Windows 7 or 10 Operating System.
Microsoft Visual Studio 2017/2020 for linking the software to a C application.

- **Microsoft Windows SDK for Windows and .NET Framework 4 (link https://www.microsoft.com/en-us/download/confirmation.aspx?id=8279)**
- **Visual Studio 2017/2022 SP1**
- pthread.lib: POSIX thread library

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)

6.1.2 PC under Linux 64-Bits

The source code has been compiled on a x86_64 (64-bit) based PC under Linux 4.10.0 and using the free software *gcc* compiler in two different configurations:

In summary, the software requirements for C users are:
***gcc* compiler version 6.3.0 (for linking the software to a C application)**
glibc 2.24

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 64-Bits (LEGACY)

The source code has been compiled on a x86_64 (64-bit) based PC under Linux 2.6.24 and using the free software *gcc* compiler in two different configurations:

In summary, the software requirements for C users are:
***gcc* compiler version 4.5.x (for linking the software to a C application)**
glibc 2.12

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 Macintosh with Intel processor under MacOS X 64-Bits

The source code has been compiled on a Macintosh (MacMini) with Intel processors under MacOS X and using the gcc 4.2 compiler provided with Xcode 5.1, that is a front-end for clang compiler. The version of the operating system is 10.10, but the libraries have been compiled with compatibility from version 10.12 onwards.

In summary, the software requirements for C users are:
Mac OS X version 10.12 or later
gcc provided with Xcode 9.2 or later

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)

6.1.5 Macintosh with ARM processor under MacOS X 64-Bits

The source code has been compiled on a Macintosh with ARM processors under MacOS X and using the clang 14.0.3 compiler provided with Xcode 14.3, that is a front-end for clang compiler. The version of the operating system is 13.5, but the libraries have been compiled with compatibility from version 13.5.2 onwards.

In summary, the software requirements for C users are:
Mac OS X version 13.5 or later
clang provided with Xcode 14.3 or later

The hardware requirements are:
Macintosh with ARM 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 (registration is required):

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

- http://eop-cfi.esa.int/index.php/mission-cfi-software/eocfi-software/branch-4-x (download page for branch 4.x)

The Earth Observation CFI Software (EOCFI SW) and additional third-party support libraries are provided under the terms and conditions stated in the file TERMS_AND_CONDITIONS.TXT include in the distribution packages.

6.3 How to install the Software

The CFI libraries can be installed by expanding the installation package in any directory.
For specific hints related to the usage of the libraries, please consult Section **Error! Reference source not found.** "CFI LIBRARIES INSTALLATION" of the General SUM and Section 6 "LIBRARY USAGE" of each Library User Manual.

6.4 Overview of Files and Directory Structure

Upon completion the installation procedure, the following directory structure will be created:
***OS* can take the following values depending on the installation package:**
LINUX64_LEGACY
LINUX64
MACIN64 (MAC OSX in machine with Intel processor 64bits)
MACARM64 (MAC OSX in machine with ARM processor 64bits)
WINDOWS64
xxx stands for the cfi library name. It can be any of the following values: *file_handling, data_handling, lib, orbit, pointing, visibility*

Installation_Directory
- **explorer_xxx**
  - **example**
    - **data:** ***data files***
    - **c: Makefiles**
      - **explorer_xxx_c.c**
      - **[explorer_xxx_run_c.c]**
    - **validation: Makefiles**
      - **explorer_xxx_valid.c**
- **files**
  - **models: model files for numeric propagator**
  - **example_files: EO CFI example files**
  - **schemas: EO CFI schema files**
  - **xslt: EO CFI style sheets files**
- **include: explorer_*xxx*.h**
- **lib *OS*: *libraries***
- **bin *OS: executables***
- **terms and conditions: TERMS_AND_CONDITIONS.TXT**
- **scripts to run the validation and example programs**
- **cfi_tools (only WINDOWS): pthread and xerces libraries**

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: Release Notes detailing the changes and/or corrections introduced in the new release.
Quick Start Guide: User manual with usage cases.

Documents are delivered as separate packages not contained in the installation package.

6.4.2 Installation directory

This is the directory where the CFI will be installed. The directory has to be created by the user. It contains scripts to compile and run the validation and example programs. The name of the scripts can be:

- validation.sh: script to compile and run the validation programs in LINUX and Mac OS platforms
- example.sh: script to compile and run the example programs in LINUX and Mac OS platforms

or

- validation.bat: script to compile and run the validation programs in Windows platforms
- example.bat: script to compile and run the example programs in Windows platforms

6.4.3 Directory: `lib`

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

6.4.4 Directory: `include`

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

6.4.5 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. Windows XP, 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.

6.4.6 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. Windows XP, Linux and MacOS.
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 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.**

6.4.8 Directory: cfi_tools

This directory contains third-party libraries and programs used by CFI functions and which are not usually installed in WINDOWS systems, in particular:

- POSIX Thread library: library used to support multithreading.

6.5 Validation Procedure

This procedure should be run to verify the proper installation of the CFI library. Two different procedures can be followed:

A) Run the validation programs for every 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.

B) Run all the validation programs with the validation script:

1. Go to installation directory (**`Installation_Directory`**).
2. Run the validation program using the script

   `validate.xxx`

   (xxx = sh for Linux and Mac Os, xxx= bat for Windows)

The validation programs for all libraries will be created and executed. At the end of the execution, a log file validation_*OS*_log.txt is created (at the level of Installation_Directory) with the result of the validation for every library.

6.6 Examples

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

Note that two examples are provided. The first one is called explorer_*xxx*_c.c and the second one explorer_*xxx*_run_c.c (for explorer_file_handling and explorer_data_handling only the first example is provided). 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:
Linux/MacOS: adding to the LD_LIBRARY_PATH environment variable the locations of all dynamic libraries needed.
Windows XP: 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.

Alternatively all the example programs can be compiled and executed using a single script:

1. Go to installation directory (**`Installation_Directory`**).
2. Run the example program using the script

   `example.xxx`

   (xxx = sh for Linux and Mac Os, xxx= bat for Windows)

The example programs for all libraries will be created and executed. At the end of the execution, a log file example_*OS*_log.txt is created (at the level of Installation_Directory) with the result of the example execution for every library.

6.7 Problems Reporting

For any problems or questions, please send an e-mail to: *cfi@eopp.esa.int*

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

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], [PNT_SUM] and [VIS_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
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

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]).

Input	Description	Enumeration value	Long
	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
	Jason-CSA	XX_SAT_JASON_CSA	132
	Jason-CSB	XX_SAT_JASON_CSB	133
	Metop-SG-A1	XX_SAT_METOP_SG_A1	134
	Metop-SG-A2	XX_SAT_METOP_SG_A2	135
	Metop-SG-A3	XX_SAT_METOP_SG_A3	136
	Metop-SG-B1	XX_SAT_METOP_SG_B1	137
	Metop-SG-B2	XX_SAT_METOP_SG_B2	138
	Metop-SG-B3	XX_SAT_METOP_SG_B3	139
	Sentinel-5P	XX_SAT_SENTINEL_5P	140
	Biomass	XX_SAT_BIOMASS	141
	Sentinel5	XX_SAT_SENTINEL_5	142
	SaocomCS	XX_SAT_SAOCOM_CS	143
	FLEX	XX_SAT_FLEX	144
	Sentinel-6A	XX_SAT_SENTINEL_6A	145
	Sentinel-6B	XX_SAT_SENTINEL_6B	146
	CIMR	XX_SAT_CIMR	147
	ROSE-L	XX_SAT_ROSEL	148
	CHIME	XX_SAT_CHIME	149
	CRISTAL	XX_SAT_CRISTAL	150
	CO2M	XX_SAT_CO2M	151
	LSTM	XX_SAT_LSTM	152
	FORUM	XX_SAT_FORUM	153
	TRUTHS	XX_SAT_TRUTHS	154
	Generic Satellite	XX_SAT_GENERIC	200
	Geostationary Generic Satellite	XX_SAT_GENERIC_GEO	300
	Geostationary Meteorological Satellite	XX_SAT_MTG	301
	Medium orbit satellite	XX_SAT_GENERIC_MEO	400
Time reference	Undefined	XX_TIME_UNDEF	-1
	TAI	XX_TIME_TAI	0

Input	Description	Enumeration value	Long
	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
	Zero-Doppler YSM	XX_AOCS_ZDOPPLER	3
Satellite Nominal Atti tude Model	Generic model	XX_MODEL_GENERIC	0
	Envisat model	XX_MODEL_ENVISAT	1
	Cryosat model	XX_MODEL_CRYOSAT	2
	ADM model	XX_MODEL_ADM	3
	SENTINEL 1 model	XX_MODEL_SENTINEL1	4
	SENTINEL 2 model	XX_MODEL_SENTINEL2	5
	Geostationary satellite model	XX_MODEL_GEO	6
	MetOp-SG	XX_MODEL_METOPSG	7
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***.

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:

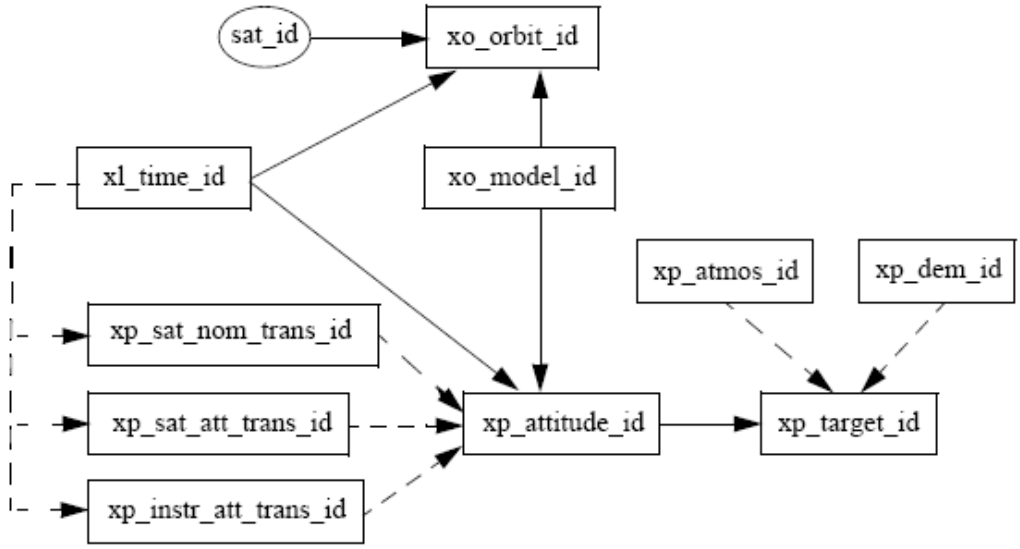
Table 3: CFI identifiers

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
xv_swath_id	EO_VISIBILITY	It stores the data of the swath, used for visibility computations
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



shows the hierarchy of the IDs in the CFI.

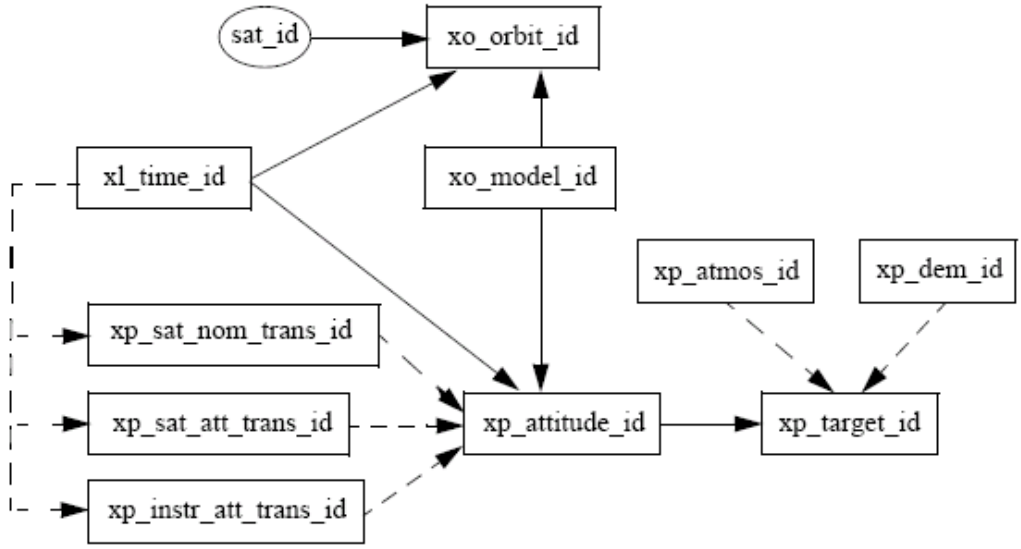


Figure 4: Hierarchical structure of the initialisation variables in the CFI

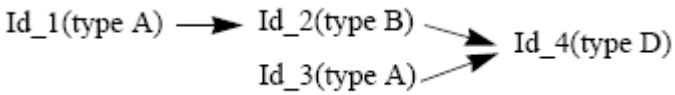
In the figure, the plain lines mean the there is a mandatory link between the ids, while the dashed line indicates the second ID can use or not the first ID.

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):

LINUX64_LEGACY: Intel(R) Xeon(R) CPU E5-2609 v4 @ 1.70GHz (8 cores)
LINUX64: Intel(R) Xeon(R) CPU E5-2470 0 @ 2.30GHz (16 cores)
WINDOWS: Intel(R) Core(TM)2 Duo CPU T8100 @ 2.10 GHz
MACIN64: Intel Core i7 4 cores @2,6 GHz, 16 GB RAM, MACOSX 10.12
MACARM64: Apple M2 Max 12 cores, 64GB RAM, macOS 13.5.2

7.5 Parallel Processing: OpenMP Parameters Customization

The EOCFI user can define the behaviour of the OPENMP parallel regions using environment variables:

- `OMP_DYNAMIC` environment variable enables or disables dynamic adjustment of the number of threads available for the execution of parallel regions. If `OMP_DYNAMIC` is set to `TRUE`, the number of threads that are used for executing parallel regions may be adjusted by the runtime environment to best use system resources. If `OMP_DYNAMIC` is set to `FALSE`, dynamic adjustment is disabled. The default condition is implementation-defined (default).
- `OMP_NUM_THREADS` environment variable sets the default number of threads to use during execution. The value of the `OMP_NUM_THREADS` environment variable must be a positive integer.

The number of threads to use is implementation-defined (i.e, default values) if:

- the `OMP_NUM_THREADS` environment variable isn't specified,
- the value specified isn't a positive integer, or
- the value is greater than the maximum number of threads that the system can support.

More info: How the number of threads are set in OPEMP: https://learn.microsoft.com/en-us/cpp/parallel/openmp/2-directives?view=msvc-170#23-parallel-construct

These variables can be ignored if the programmer uses the OMP API functions:

- omp_set_dynamic(true/false)
- omp_set_num_threads(number_of_threads)

that overrides that environment variables.

7.6 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:

7.6.1 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_version();
}
```

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)

7.6.2 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.

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 = 4.17
Current version of explorer_visibility is compatible with explorer_file_handling
v4.17
Current version of explorer_visibility is compatible with explorer_data_handling
v4.17
Current version of explorer_visibility is compatible with explorer_lib v4.17
Current version of explorer_visibility is compatible with explorer_orbit v4.17
Current version of explorer_visibility is compatible with explorer_pointing v4.17
```

B)

expcfi_check_libs -**dir** ./expcfi_dir

The output would be:

```
Current Earth Explorer CFI libraries:
-------------------------------------
explorer_file_handling version = 4.17
explorer_data_handling version = 4.17
explorer_lib version = 4.17
explorer_orbit version = 4.17
explorer_pointing version = 4.17
explorer_visibility version = 4.17
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

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

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