RE-ENGINEERING OF MISSION ANALYSIS SOFTWARE FOR ENVISAT-1

GENERAL SOFTWARE USER MANUAL

PO-IS-DMS-GS-0556

Code:	PO-IS-DMS-GS-0556
Issue:	5.9
Date:	30/05/11

	Name	**Function**	**Signature**
Prepared by:	Carlos Villanueva Muñoz	Project Engineer	
Checked by:	Juan José Borrego Bote	Project Engineer	
Approved by:	José Antonio González Abeytua	Project Manager	

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:	Contract number	Internal	
		Public	
Contract Issuer:	ESA / ESTEC	Industry	X
		Confidential	

Internal Distribution		
Name	Unit	Copies

External Distribution		
Name	Organisation	Copies

Archiving	
Word Processor:	Framemaker 6.0
File name:	GeneralSum
Archive Code	P/SUM/DMS/01/043-024

Document Status Log

Issue	Change Description	Date	Approval
4.1	• First Version	15/01/97	
4.2	• pp_genatt removed from the CFI library • `int` variables replaced by `long` in the error handling functions • known problems of release 4.1 solved • updated overall dataflow diagram	30/04/97	
4.3	• Updated according to the release of 4.3 and 2.2 CFI libraries	18/05/98	
4.4	• Updated according to the release of 4.3, 2.2 and 1.1 CFI libraries	19/10/98	
4.5	• Updated according to the release of PPF_LIB, PPF_ORBIT and PPF_POINTING version 4.5 , PPF_GENREF and PPF_VISIBILITY version 2.4, PPF_ENVIPLAN version 1.4 and PPF_INSTPLAN version 1.0 CFI libraries	31/05/99	
4.6	• Unreleased	14/04/00	
4.7	• Unreleased	22/06/01	
4.8	• Unreleased	31/07/01	
4.9	• Unreleased	22/10/01	
5.0	• Unreleased	18/01/02	
5.1	• Unreleased	25/11/02	
5.2	• Document format change. • New document from previous version by: - L. J. Alvarez (GMV) - J. A. Gonzalez (GMV) - M. Sanchez-Nogales (GMV) • See change bars	26/05/03	
5.3	• Maintenance Release.	13/12/04	
5.3.1	• New libraries for LINUX and MAC OS • Maintenance Release	15/02/05	
5.4	• Maintenance Release	17/05/05	

Issue	Change Description	Date	Approval
5.5	• Library optimization	30/01/06	
5.6	• New libraries for LINUX 64-bit • Maintenance Release	22/12/06	
5.7	• Changes for Envisat extended life • New libraries for MAC OS on Intel platforms • Maintenance Release	11/04/08	
5.8	• Changes for Envisat extended life • Adaptation to version 2 of files. • New functions to generate swath and Orbit Scenario Files. • Maintenance release	15/09/09	
5.9	• Improved the Sun position calculation. • Reference orbit in which there is no DRS visibility for the mission extension. • ASAR swath file generation corrected. • Maintenance release.	30/05/11	

Table of Contents

1 SCOPE

The Software User Manual (SUM) of the Envisat-1 mission CFI software is composed of

- a general document describing the sections common to all the CFI software libraries
- a 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:

- PPF_LIB Software User Manual issue 5.9 (see RD 3)
- PPF_ORBIT Software User Manual issue 5.9 (see RD 4)
- PPF_POINTING Software User Manual issue 5.9 (see RD 5)
- PPF GENREF Software User Manual issue 3.9 (see RD 6)
- PPF VISIBILITY Software User Manual issue 3.9 (see RD 7)

Note: This document and the associated documents listed above supersede the following documents (see section 3.3):

- XD 1 Orbit Propagator Software Interface & Installation Guide. PPF-TN-ESA-GS-0006
- XD 2 TARGET Software Interface & Installation Guide. PPF-TN-ESA-GS-0009
- XD 3 STAVIS Software Interface & Installation Guide. PPF-TN-ESA-GS-0007
- XD 4 DRSVIS Software Interface & Installation Guide. PPF-TN-ESA-GS-0008
- XD 5 Time Handling and Processing Software Interface & Installation Guide. PO-TN-ESA-GS-0248
- XD 6 GENOPS Software Interface. PO-TN-ESA-GS-0455
- XD 7 INTERPOL Software Interface. PO-TN-ESA-GS-0456

2 ACRONYMS AND NOMENCLATURE

2.1 Acronyms

ANX	Ascending Node Crossing
CFI	Customer Furnished Item
ESA	European Space Agency
ESTEC	European Space Technology and Research Centre
H/W	Hardware
I/F	Interface
PDF	Page Description Format
PS	PostScript
SBT	Satellite Binary Time
SUM	Software User Manual
S/W	Software
UTC	Universal Time Coordinated
WWW	World Wide Web

2.2 Nomenclature

CFI A group of CFI functions, and related software and documentation, that will be distributed by ESA to the users as an independent unit

CFI function A single function within a CFI that can be called by the user

Library A software library containing all the CFI functions included within a CFI plus the supporting functions used by those CFI functions (transparently to the user)

3 APPLICABLE AND REFERENCE DOCUMENTS

3.1 Applicable documents

AD 1 Re-engineering of Mission Analysis Software for Envisat-1: Statement of Work. PO-SW-ESA-GS-0344. ESA/ESTEC/NW. Issue 3.0. 19/12/1995.

AD 2 ESA Software Engineering Standards. ESA PSS-05-0. ESA. Issue 2. February 1991

3.2 Reference documents

RD 1 Envisat-1 Mission CFI Software - Description and Interface Definition Document. PO-ID-ESA-SY-00412.

RD 2 Envisat-1 Mission CFI Software - Mission Conventions Document. PO-IS-GMV-GS-0561

RD 3 Envisat-1 Mission CFI Software - PPF_LIB Software User Manual. PO-IS-DMS-GS-0557

RD 4 Envisat-1 Mission CFI Software - PPF_ORBIT Software User Manual. PO-IS-DMS-GS-0558

RD 5 Envisat-1 Mission CFI Software - PPF_POINTING Software User Manual. PO-IS-DMS-GS-0559

RD 6 Envisat-1 Mission CFI Software - PPF_GENREF Software User Manual. PO-IS-DMS-GS-0560

RD 7 Envisat-1 Mission CFI Software - PPF_VISIBILITY Software User Manual. PO-IS-DMS-GS-0609

RD 8 Envisat-1 Mission CFI Software - PPF_ENVIPLAN Software User Manual PO-IS-DMS-GS-0860

RD 9 Envisat-1 Mission CFI Software - PPF_INSTPLAN Software User Manual PO-IS-DMS-GS-0861

3.3 Superseded documents

XD 1 Orbit Propagator Software Interface & Installation Guide. PPF-TN-ESA-GS-0006

XD 2 TARGET Software Interface & Installation Guide. PPF-TN-ESA-GS-0009

XD 3 STAVIS Software Interface & Installation Guide. PPF-TN-ESA-GS-0007

XD 4 DRSVIS Software Interface & Installation Guide. PPF-TN-ESA-GS-0008

XD 5 Time Handling and Processing Software Interface & Installation Guide. PO-TN-ESA-GS-0248

XD 6 GENOPS Software Interface. PO-TN-ESA-GS-0455

XD 7 INTERPOL Software Interface. PO-TN-ESA-GS-0456

4 INTRODUCTION

This General Software User Manual consists of the following sections:

- an introduction explaining how to use this document(section 4)
- an 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 (for a more detailed description of these libraries and associated CFI functions, please refer to RD 1)
- an 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
- a library usage overview (section 7), describing how to create a user application
- a 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 (see RD 3, RD 4, RD 5, RD 6 and RD 7) 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.

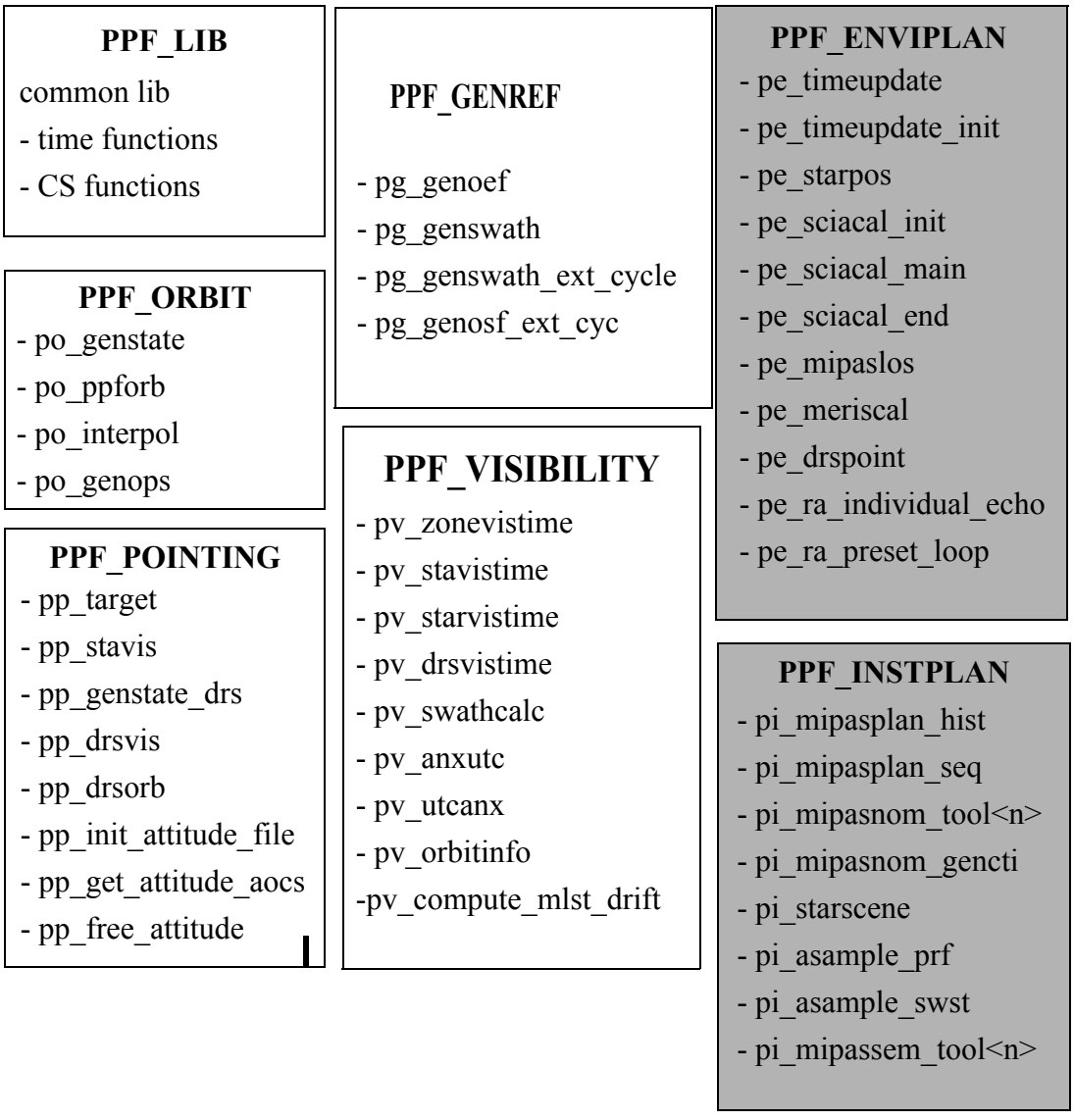
In addition to the general and specific SUM for a CFI library, the user must refer to the Mission Conventions Document (RD 2) for details on the time references and formats, coordinate systems, parameters and models used in all these software user manuals.

5 CFI LIBRARIES OVERVIEW

The Envisat-1 Mission CFI software is a collection of software functions performing accurate computations of mission related parameters for Envisat-1.

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 the next drawing:

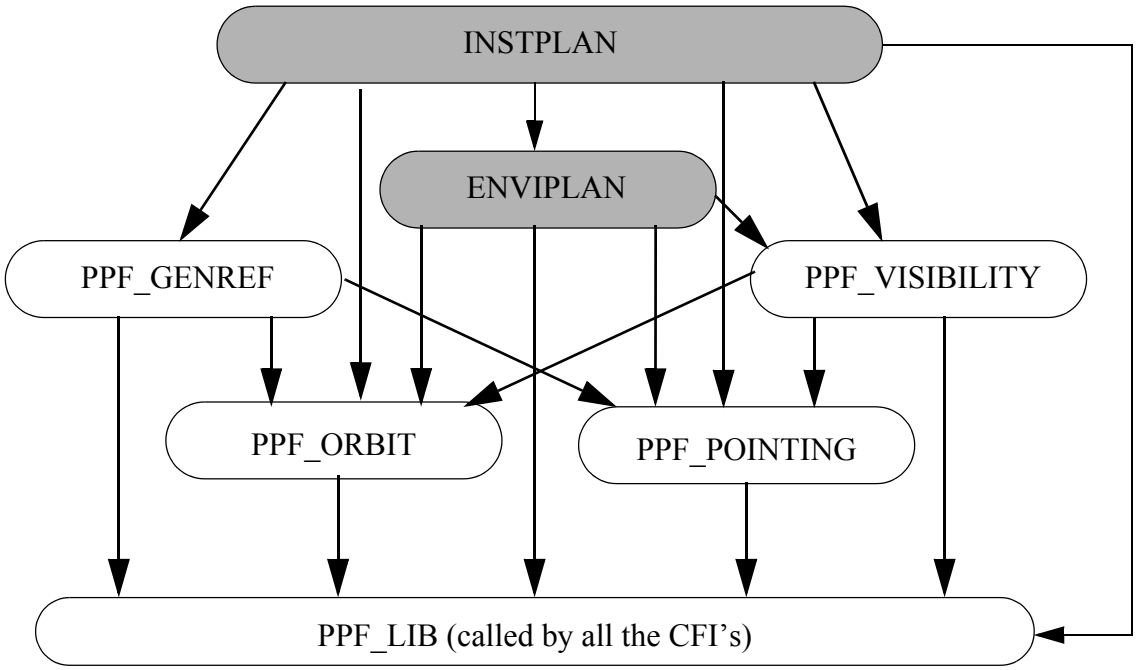


Those libraries depicted in white are currently available, whereas the shadowed ones belong to RGT CFI package and are not provided together the first ones.

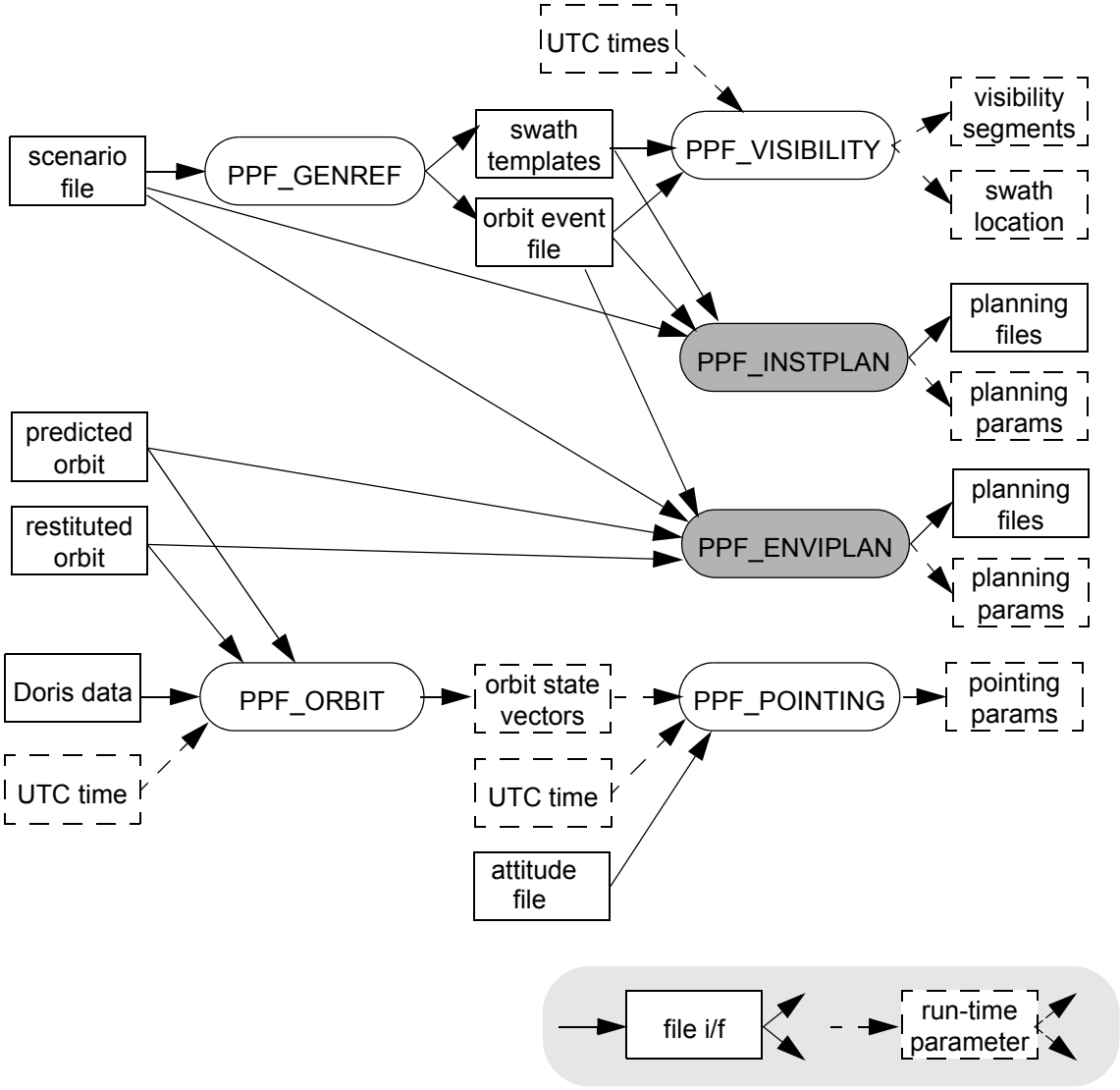
Not all libraries are available for general release, please check section 6.2 for the procedure to get the software.

The CFI software libraries are to be seen as several layers, each layer being directly accessible to a user's program. Lower level 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.

Next figure 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.



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



6 CFI LIBRARIES INSTALLATION

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

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

- ppf_lib
- ppf_orbit
- ppf_pointing
- ppf_genref
- ppf_visibility

6.1 Usage requirements

Each CFI software library is distributed as an object code, which makes it completely system dependent. Six computer systems are supported:

- Sun under Solaris in 32 and 64 bits
- IBM under AIX.
- PC under Windows 2000.
- PC under Suse Linux 9.2.
- Macintosh under MacOS X 10.3.7.

C++ compatibility is supported for all libraries.

6.1.1 Sun under Solaris

The source code has been compiled on a Sun Sparcstation under Solaris 2.5 and using the Sparcworks C compiler `cc` 3.0.1

The software requirements are:

- Solaris 5.7 Operating System
- Solaris 2.5 `libm.a` (or `libm.so`) math library
- Workshop C compiler `cc` 4.2 (if you are linking this software to a C application)
- Sun Fortran Compiler `f77` (if you are linking this software to a Fortran application)

The hardware requirements are:

- Sun Sparcstation
- 76 Mb free of disk space (for all 5 CFIs together)
- 16 Mb RAM (for all 5 CFIs together)

6.1.2 Sun under Solaris 64 Bits

The source code has been compiled on a Sun Sparcstation under Solaris 5.9 and using the SunStudio9 C compiler `cc` 5.6

The software requirements are:

- Solaris 5.9 Operating System
- Solaris `libm.a` (or `libm.so`) math library
- SunStudio9 C compiler `cc` 5.6 (if you are linking this software to a C application)
- Sun Fortran Compiler `f77` (if you are linking this software to a Fortran application)

The hardware requirements are:

- Sun Sparcstation
- 76 Mb free of disk space (for all 5 CFIs together)
- 16 Mb RAM (for all 5 CFIs together)

6.1.3 IBM under AIX

The source code has been compiled on a IBM SP/2 Workstation under AIX 4.3 and using the IBM C compiler `xlc 3.1.4`

In summary the software requirements are:

- AIX 4.3 Operating System
- AIX 3.2 `libm.a` (or `libm.so`) math library
- IBM C compiler `xlc` (if you are linking this software to a C application)

The hardware requirements are:

- IBM SP/2 Sparcstation
- 63 Mb free of disk space (for all 5 CFIs together)
- 16 Mb RAM (for all 5 CFIs together)

6.1.4 PC under Windows 2000

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

In summary the software requirements are:

- Microsoft Windows 2000 Operating System
- Microsoft Visual C++ 6.0 Compiler
- Digital Visual Fortran Compiler

The hardware requirements are:

- PC
- 21 Mb free of disk space (for all 5CFIs together)
- 4 Mb RAM (for all 4 CFIs together)

6.1.5 PC under Suse Linux 9.2

The source code has been compiled on a PC under Suse Linux 9.2 and using the C Compiler gcc 3.3.4.

In summary the software requirements are:

- Suse Linux 9.2 Operating System
- gcc compiler version 3.3.4
- glibc 2.3.3
- Fortran Compiler g77 3.3.4

The hardware requirements are:

- PC
- 76 Mb free of disk space (for all 5 CFIs together)
- 16 Mb RAM (for all 5 CFIs together)

6.1.6 PC under Linux 64-Bits

The source code has been compiled on a PC under Linux 2.6.9.34 (RedHat Enterprise 4) and using the free software *gcc* compiler version 3.4.5.
In summary, the software requirements for are:

- Linux 2.6.9-34 (RedHat Enterprise 4)
- gcc compiler version 3.4.5(for linking the software to a C application)
- glibc 2.3.4
- Fortran Compiler g77

The hardware requirements are:

- PC
- 76 Mb free of disk space (for all 5 CFIs together)
- TBD Mb RAM (for all 5 CFIs together)

6.1.7 Macintosh under MacOS X 10.3.8

The source code has been compiled on a Macintosh under MacOS Xand using the C Compiler gcc 3.3.
In summary the software requirements are:

- MacOS X version 10.3.8
- gcc compiler version 3.3
- *g77* compiler version 3.4.1 (for linking the software to a ForTran application)

The hardware requirements are:

- Macintosh
- 63 Mb free of disk space (for all 5 CFIs together)
- 16 Mb RAM (for all 5 CFIs together)

6.1.8 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 10.4.9 or later
- *gcc* compiler version 4.0.1 (for linking the software to a C application)

The hardware requirements are:

- Macintosh with Intel Processor
- TBD Mb free of disk space (for all 5 CFIs together)
- TBD Mb RAM (for all 5 CFIs together)

6.1.9 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 10.4.9 or later
- *gcc* compiler version 4.0.1 (for linking the software to a C application)

The hardware requirements are:

- Macintosh with Intel Processor
- TBD Mb free of disk space (for all 5 CFIs together)
- TBD Mb RAM (for all 5 CFIs together)

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 Envisat CFI page:

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

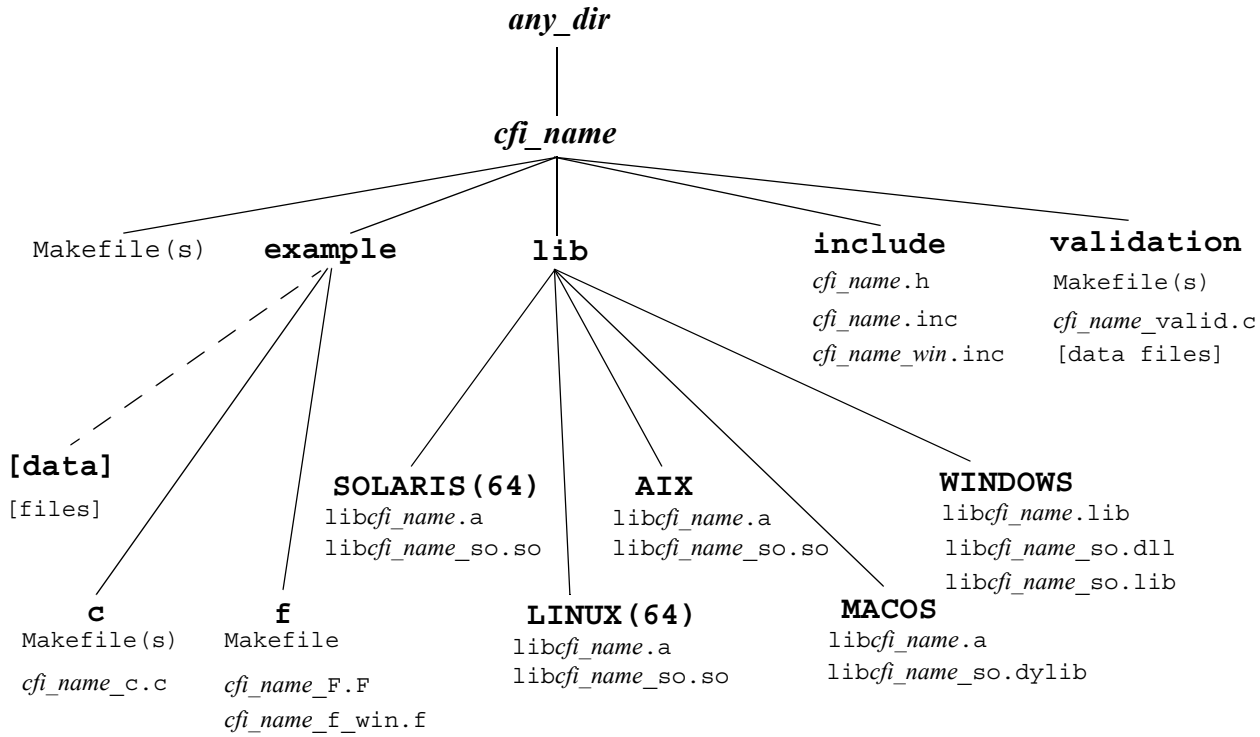
6.3 How to install the software

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 each CFI library:



Makefile shall be chosen for the appropriate platform.

6.4.1 Documents

The following PDF files are available:

- `mcdX.X.pdf`: Mission Conventions Document
- `GeneralSum_X.X.pdf`: General Software User Manual
- `Release Notes_VX.X.pdf`: Release Notes detailing the changes and/or corrections introduced in the new release
- *cfi_name*Sum_X.X.pdf: Specific Software User Manual for every CFI library.

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

6.4.2 Directory: *cfi_name*`/lib`

This directory contains the CFI object libraries. There is 1 file per supported computer platform, each in a separate sub-directory:

- **`SOLARIS/`**`lib`*cfi_name*`.a`: for Sun/SOLARIS users
- **`SOLARIS64/`**`lib`*cfi_name*`.a`: for Sun/SOLARIS 64-bits users
- **`AIX/`**`lib`*cfi_name*`.a`: for IBM/AIX users
- **`WINDOWS/`**`lib`*cfi_name*`.lib`: for PC/Windows 2000 users
- **`LINUX/`**`lib`*cfi_name*`.a`: for PC/Linux users
- **`LINUX64/`**`lib`*cfi_name*`.a`: for PC/Linux 64-bits users
- **`MACOS/`**`lib`*cfi_name*`.a`: for Macintosh/MacOS users
- **`MACIN/`**`lib`*cfi_name*`.a`: for Macintosh/MacOS/Intel platform users
- **`MACIN64/`**`lib`*cfi_name*`.a`: for Macintosh/MacOS/Intel platform 64-bits users

6.4.3 Directory: *cfi_name*`/include`

This directory contains the include files with the function declaration for every CFI function, plus the related enumerations. There are 2 files:

- *cfi_name*`.h`: for C users
- *cfi_name*`.inc`: for Fortran users under Solaris/Solaris64/AIX/Linux/MacOS/Mac-Intel
- *cfi_name_win*`.inc`: for Fortran users under Windows 2000

6.4.4 Directory: *cfi_name*`/validation`

This directory contains the validation program and associated makefile:

- *cfi_name*`_valid.c`

Depending on the CFI, input data files used by the validation program may be included.

6.4.5 Directory: *cfi_name*`/examples`

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

- **`f/`***cfi_name*`_F.F`**: for Fortran users under Solaris/AIX/Linux/MacOS/Mac-Intel
- **`f/`***cfi_name*`_win_f.f`**: for Fortran users under Windows 2000

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

6.4.6 File: *cfi_name*`/Makefile`

This makefile is used by Solaris/AIX users, at the end of the installation procedure, to move the CFI files to their final destination.

6.5 Validation procedure

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

1. Go to directory ***cfi_name*`/validation`**
2. Edit the `Makefile` (one version already exists for each platform: Solaris/Solaris64/AIX/Windows 2000/Linux/MacOS) and configure it to your computer platform. The configuration parameters are all located at the top of the `Makefile`, with instructions on how to use them.

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 6.7, this will be easier (as all already installed include files will be grouped in one directory, and all already installed libraries as well).

3. Run the validation program using

   make (for Solaris/Solaris64/AIX/Linux/MacOS users)

   `nmake` (for Windows 2000 users)

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 6.8).

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

6.6 Examples

Two 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.

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.

6.7 Finalizing the Installation

Once the validation procedure has been run successfully, you can move the CFI files to the appropriate directories. In order to simplify compiling and linking options when using the CFI, it is advisable to:

- place all CFI libraries in the same directory
- place all CFI include files in the same directory

In order to help this final installation, you may want to use the top-level `Makefile` delivered with the CFI (one version exists for each platform: Solaris/Solaris64/AIX/Windows 2000/Linux/MacOS):

1. Go to directory ***cfi_name***
2. Edit the `Makefile` 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. Perform the final installation using

   make install

6.8 Problem reporting

For any problems or questions, please send an email to:

cfi@jw.estec.esa.nl

7 CFI LIBRARIES USAGE

7.1 Using CFIs 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.:

- pl_ for PPF_LIB
- po_ for PPF_ORBIT
- pp_ for PPF_POINTING
- pg_ for PPF_GENREF
- pv_ for PPF_VISIBILITY

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

However, to preserve compatibility with the historical CFI function names, CFI functions can be called from a user application either with or without the corresponding prefix, e.g. xx_cfi_function or cfi_function, if the historical function name existed (see specific SUM's)

Details can be found in the specific Software User Manuals of each CFI (see RD 3, RD 4, RD 5, RD 6 and RD 7).

7.2 Use of enumerations

To help readability in the user application, it is possible to use meaningful enumeration values rather than integer values for all the input modes and switches of the CFI functions.

8 ERROR HANDLING DESCRIPTION

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*_ (see 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 of all the CFI software libraries, except the simpler functions of the PPF_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:

- 0 for NOMINAL
- +1 for WARNING
- -1 for ERROR

All elements of the status vector may take values:

- zero if nominal
- positive if one or more warnings occurred
- negative if one or more errors occurred

8.2 Functions Returning an Extended Status Flag

The simpler CFI functions of the PPF_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 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 == 0)` to detect nominal execution
- `(status >= +1)` to detect warnings
- `(status <= -1)` to detect errors

To facilitate the applications readability it is possible to use the following enumeration values to check the status returned by a CFI function:

Description	Enumeration value	long
An error is returned by the CFI function	*XX*_ERR	-1
Nominal execution of the CFI function	*XX*_OK	0
A warning is returned by the CFI function	*XX*_WARN	1

8.4 Retrieving the 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_vector_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_vector_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_vector_msg** function or printed automatically if the verbose mode is set, is as follows.

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

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

Examples of error messages are:

PPF_POINTING >>> ERROR in pp_target: Target not found (Sat_tar)

PPF_LIB >>> ERROR in pl_emjd: Input MONTH STRING is not correct

Finally, it is also possible for the user to send to the standard error output (STDERR) the error messages returned by the **xx_vector_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 both for C users and Fortran users.

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

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

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

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

8.5 xx_silent

8.5.1 Overview

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

8.5.2 Calling interface

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

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

      status = xx_silent();
}
```

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

```
#include <cfi_name.inc>

      INTEGER*4 STATUS

      STATUS = XX_SILENT()
```

8.5.3 Input parameters

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

8.5.4 Output parameters

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

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 (see section 8.9).

8.6.2 Calling interface

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

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

      status = xx_verbose();
}
```

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

```
#include <cfi_name.inc>

      INTEGER*4 STATUS
      STATUS = XX_VERBOSE()
```

8.6.3 Input parameters

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

8.6.4 Output parameters

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

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

8.7 xx_vector_code

8.7.1 Overview

The **xx_vector_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_vector_code** CFI error handling function is the following (input parameters are underlined):

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

      status = xx_vector_code(&func_id, ierr, &n, vec);
      status = xx_vector_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` variable is defined in the file *cfi_name*`.h`. Its value has been modified in the 4.4/2.3 issue to 256.

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

```
#include <cfi_name.inc>

      INTEGER*4 FUNC_ID, N
      INTEGER*4 IERR(LENGth_ERROR_VECTOR), EXT_STATUS
      INTEGER*4 VEC(256), STATUS

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

8.7.3 Input parameters

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

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_vector_code** CFI error handling function are:

C name	C type	Array Element	Description (Reference)	Unit (Format)	Allowed Range
xx_vector_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_vector_msg

8.8.1 Overview

The **xx_vector_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 (see section 8.9).

8.8.2 Calling interface

The calling interface of the **xx_vector_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[length_error_vector], ext_status, status;

      status = xx_vector_msg(&func_id, &ierr, &n, msg);
      status = xx_vector_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` and *XX*`_MAX_STRING` variable are defined in the file *cfi_name*`.h`

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

```
#include <cfi_name.inc>

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

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

8.8.3 Input parameters

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

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_vector_msg** CFI error handling function are:

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

8.9 xx_print_msg

8.9.1 Overview

The **xx_print_msg** CFI error handling function send 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 variable are defined in the file *cfi_name*.h

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

```
#include <cfi_name.inc>

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

      STATUS = XX_PRINT_MSG(N, MSG)
```

8.9.3 Input parameters

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

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:

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:

CFI library	Problem	Work around solution
	(no known problems)	