Earth Observation Mission CFI Software

EOCFI

Use of EOCFI from MATLAB and IDL

Code : P/TN/DMS/01/026-144
Issue : 1.1
Date : 01/09/2022

	Name	Function	Signature
Prepared by	Jonathan L. Gimeno Ioan Mugurel Tulan Carlos Villanueva	Project Engineer Project Engineer Project Manager	
Reviewed by	Inês Estrela	Project Manager	
Approved by	Inês Estrela	Project Manager	
Signatures and approvals on original			

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

This page intentionally left blank

Document Information

Contract Data	
Contract Number:	4000102614/10/NL/FF/ef
Contract Issuer:	ESA / ESTEC

Internal Distribution		
Name	Unit	Copies
Jose A. Gonzalez	MDS	1
Internal Confidentiality Level (DMG-COV-POL05)		
Unclassified ☐	Restricted ☑	Confidential ☐

External Distribution		
Name	Organisation	Copies
Michele Zundo	ESA / ESTEC	1
Maurizio Di Bartolomei	ESA / ESTEC	1

Archiving	
Word Processor:	MS Word 2000
File Name:	EOCFI_MATLAB_IDL_TN

Document Status Log

Issue	Section	Change Description	Date
1.0		First version	18/05/2015
1.1		Updated specs for MATLAB and EOCFI (Sections 3 and 4) Corrected enumerate values (Section 3.7) Corrected windows paths set in %PATH% and librarypath.txt (3.1)	01/09/2022

Table of Contents

List of Tables

1. INTRODUCTION

MATLAB and IDL are powerful tools for mathematical computations and plotting. When these tools are to be used in the analysis of satellite related issues, it is desirable to be able to call EOCFI functions from within MATLAB and IDL, so the data needed for analysis is computed just before it is used. In this technical note it is explained how to call EOCFI functions from MATLAB and IDL.

1.1. Purpose and scope

In this technical note, the steps required to call the EOCFI libraries from MATLAB and IDL are described, including:
- How to call the EOCFI functions from MATLAB and IDL.
- Configurations needed.
- Mapping of variables.
- Persistency of data.
- Examples

1.2. Acronyms and Abbreviations

The acronyms and abbreviations used in this document are the following ones:

Acronym	Description
EOCFI	Earth Observation CFI
IDL	Interactive Data Language
MATLAB	Matrix Laboratory
RD	Reference Document

2. RELATED DOCUMENTS

2.1. Applicable Documents

The following table specifies the applicable documents that shall be complied with during project development.

Table 1: Applicable documents

Reference	Code	Title	Issue	Date
[AD 1]				

2.2. Reference Documents

The following table specifies the reference documents that shall be taken into account during project development.

Table 2: Reference documents

Reference	Title	Issue	Date
[RD1]	http://uk.mathworks.com/help/matlab/matlab_external/bringing-java-classes-and-methods-into-matlab-workspace.html		
[RD2]	http://uk.mathworks.com/help/matlab/matlab_external/creating-and-using-java-objects.html		
[RD3]	http://uk.mathworks.com/help/matlab/ref/import.html		
[RD4]	http://uk.mathworks.com/help/matlab/matlab_external/working-with-java-arrays.html		
[RD5]	http://uk.mathworks.com/help/matlab/matlab_prog/destroying-objects.html		
[RD6]	http://www.exelisvis.com/docs/CALL_EXTERNAL.html		
[RD7]	http://www.exelisvis.com/docs/MAKE_DLL.html		

3. CALLING EOCFI FROM MATLAB

The following versions of MATLAB and EOCFI were used to write this technical note:

- MATLAB v9.12.0.2009381 (R2022a) (64 bit for WINDOWS operating system)
- EOCFI 4.24 (JAVA libraries)

3.1. Configurations needed

3.1.1. Path to CFI libraries

The environment variable *PATH* for WINDOWS (*LD_LIBRARY_PATH* for LINUX, *DYLD_LIBRARY_PATH* for MAC OSX) should be updated to point to the location of JNI libraries from **Java** distribution package. The JNI libraries are located in the EOCFI distribution package in the folder: libraries*[System].*

Additionally, for Windows platforms, the path to **pthread** library should be set (pthread library is located in the EOCFI distribution package in the folder: cfi_tools).

Example how to set PATH environment variable (WINDOWS), assuming that EOCFI_DIRECTORY is the directory where the EOCFI libraries are installed:

set PATH=%PATH%;C:\EOCFI_DIRECTORY\libraries\WINDOWS64;C:\EOCFI_DIRECTORY\cfi_tools;

3.1.2. The file "librarypath.txt"

The file "librarypath.txt" should be updated similarly to environment variable "PATH"/ "LD_LIBRARY_PATH"/ "DYLD_LIBRARY_PATH". The file is located in the folder "toolbox\local" of MATLAB installation directory. MATLAB installation directory can be obtained executing the command "**matlabroot**" in the MATLAB command window.

A new line with the location of JNI libraries must be added in this file. For Windows platforms, a new line with the location of **pthread** library should be added too.

Example of file "librarypath.txt":

##

FILE: librarypath.txt

##

Entries:

o path_to_jnifile

o [alpha,glnx86,sol2,unix,win32,mac]=path_to_jnifile

o $matlabroot/path_to_jnifile

o $jre_home/path_to_jnifile

##

$matlabroot/bin/$arch

$matlabroot/sys/jxbrowser/$arch/lib

C:\CFI_HOME\libraries\WINDOWS64

C:\CFI_HOME\cfi_tools

3.1.3. The file javaclasspath.txt

The file "javaclasspath.txt" should be updated with the full path to each JAVA library (.jar files) of EOCFI distribution package. The file "javaclasspath.txt" is located either in the **preferences folder** (run the command *prefdir* in MATLAB command WINDOW to obtain the preferences folder) or in MATLAB startup folder (run the command *userpath* in MATLAB command WINDOW to obtain the MATLAB startup folder). If the file does not exist in the stated locations, create a new file named "javaclasspath.txt".

Note: Classes specified in javaclasspath.txt from **MATLAB startup** folder override classes specified in the same file from **preferences folder**.

Reference: [RD1]

Example how to fill the file "javaclasspath.txt":

C:\CFI_HOME\libraries\WINDOWS64\EECommon.jar

C:\CFI_HOME\libraries\WINDOWS64\FileHandling.jar

C:\CFI_HOME\libraries\WINDOWS64\DataHandling.jar

C:\CFI_HOME\libraries\WINDOWS64\Lib.jar

C:\CFI_HOME\libraries\WINDOWS64\Orbit.jar

C:\CFI_HOME\libraries\WINDOWS64\Pointing.jar

C:\CFI_HOME\libraries\WINDOWS64\Visibility.jar

3.2. Creating Java objects in MATLAB workspace

Java objects can be created in MATLAB calling the Java class constructor.

Reference: [RD2]

For example, to create a Java "Date" object, the following statement should be executed in MATLAB (in command WINDOW or in a user defined function).

```
myDate = java.util.Date
```

MATLAB output:

myDate =

Mon Apr 27 11:57:06 EEST 2015

The Java members of the object "myDate" can be used in MATLAB. For example, to get the hours from the object "myDate", in MATLAB, the following statement can be used:

```
hours=myDate.getHours()
```

MATLAB output:

hours =

11

3.3. Creating EOCFI objects in MATLAB workspace

EOCFI Java objects can be created and used in MATLAB. For example to create a "SatId" object the following statement can be used:

```
import EECFI.*;
satId1 = SatId(EnumOrbit.XO_SAT_SENTINEL_1A);
```

Now, the members of SatId object can be used in MATLAB. For example, to get the satellite number the function "getSatellite()" can be used in MATLAB:

```
satId1.getSatellite()
```

MATLAB output:

ans =

110

Notes:

- The EOCFI package must be imported in MATLAB before using functionalities from it. This can be done using "import" keyword.
- Static members must be prefixed by the class name

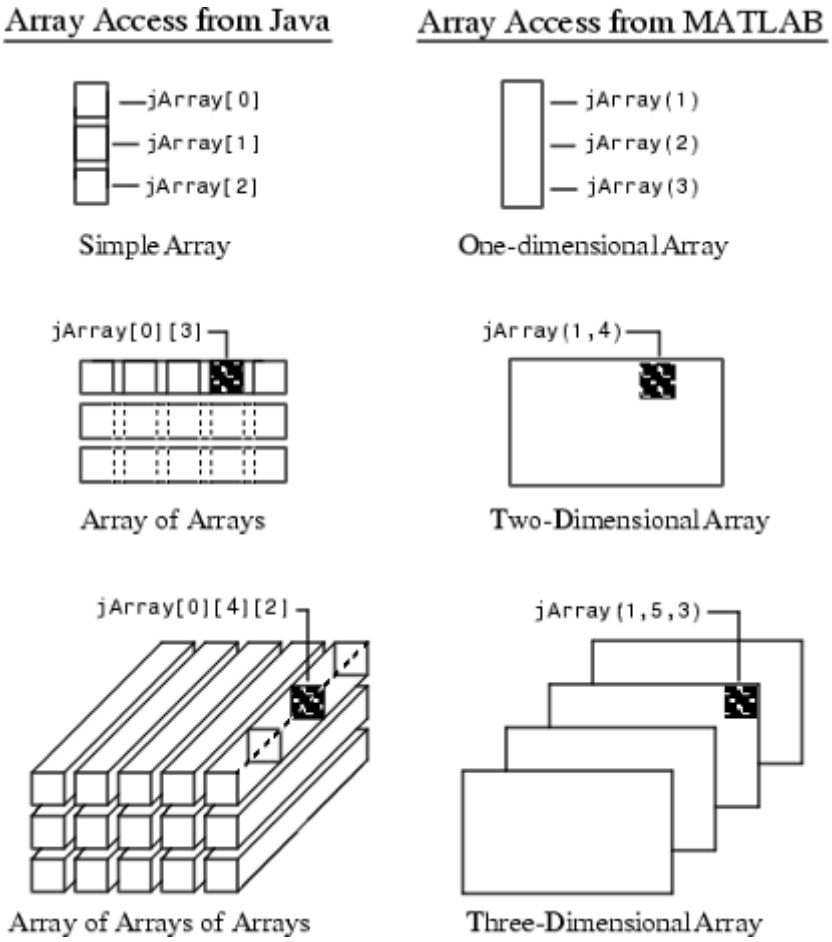
Reference: [RD3]

3.4. Accessing Java arrays in MATLAB

The structure of a Java array is different from the structure of a MATLAB matrix or array. However, MATLAB allow to operate on the Java arrays using the usual MATLAB command syntax.

Note: Java array indexing is different than MATLAB array indexing. Java array indices are zero-based, but MATLAB array indices are one-based. In Java programming, elements of an array *y* of length *N* can be accessed using *y[0]* through *y[N-1]*. When working with this array in MATLAB, these same elements can be accessed using the MATLAB indexing style from *y(1)* to *y(N)*. Thus, if you have a Java array of 10 elements, the seventh element is obtained in MATLAB using *y(7)*, and not *y[6]* as it is done when writing a Java program.

Reference: [RD4]



3.5. Catching EOCFI errors in MATLAB

MATLAB allows to execute the statements and catch resulting errors. EOCFI errors can be differentiated from the other type of errors and specific behaviour can be implemented for those errors.

In the following catch, the EOCFI exceptions are identified; then the CFI function ***getMsg()*** from ***CfiError*** object is called to obtain the error description as text message which is displayed.

```
catch e
    if (isa(e, 'matlab.exception.JavaException') && isa(e.ExceptionObject,
'EECFI.CfiError'))
        cfiError = e.ExceptionObject;
        errorMessages = java.util.Vector;
        errorMessages = cfiError.getMsg(errorMessages);
        fprintf('CFI error:\n');
        for i = 0:errorMessages.size()-1
            fprintf('%s\n', errorMessages.elementAt(i));
        end
    else
        e
    end
end
```

3.6. Persistency of data

An object's lifecycle ends when:

- You reassign a new value to that variable.
- The object is no longer used in a function.
- Function execution ends.

Reference: [RD5]

3.7. Example

The following MATLAB function shows an example how to call OrbitId::osvCompute and OrbitId::osvComputeExtra in MATLAB.

The function takes as input arguments:

- the orbit file; is an external file and the full path to the file must be used if the file is not located in MATLAB start up folder: **[path_to_file]\orbit.file**

  In section 3.7.3 can be found an orbit file example (OSF with one orbital change)
- the number of points on which the time interval will be divided.

The *time interval* is represented by the UTC time of the first/last osv record.

The results of osvComputeExtra:

- Geocentric longitude
- Geodetic latitude
- Geodetic altitude

are stored for each iteration and finally a plot is drawn.

3.7.1. Source code

```
%Example how to call OrbitId::osvCompute and OrbitId::osvComputeExtra.
%The function takes as input arguments the orbit file and the number of points
on which the time interval will be divided.
%The time interval is represented by the UTC time of the first/last osv record
% The results of osvComputeExtra:
% - Geocentric longitude
% - Geodetic latitude
% - Geodetic altitude
% are stored for each iteration and finally a plot is drawn.


function [  ] = OrbitTest( orbitFile, points )
import java.io.File;
import EECFI.*;
try
    sat_id1 = EnumOrbit.XO_SAT_SENTINEL_1A; %You must still use the class name
to call static methods: check this:
http://uk.mathworks.com/help/matlab/ref/import.html
    satId1 = SatId(sat_id1);
```

```
    modelId = ModelId;

    tai = -1100.100000000;
    utc = tai - 35.0/86400.;
    ut1 = tai - 35.3/86400.;
    gps = tai - 19.0/86400.;

    timeCorrValues = [tai utc ut1 gps];
    timeId = TimeCorrelation(timeCorrValues);

    time_ref = EnumOrbit.XO_TIME_UTC;
    orbit_mode = EnumOrbit.XO_ORBIT_INIT_AUTO;

    filesVector = java.util.Vector;
    oemFile = File(orbitFile);
    filesVector.addElement(oemFile.getPath());
    time_mode  = EnumOrbit.XO_SEL_FILE;
    time0 = 0.0;
    time1 = 0.0;
    orbit0 = 0;
    orbit1 = 0;

    orbitId1 = OrbitId(satId1, modelId, timeId, ...
                       time_ref, orbit_mode, ...
                       filesVector, ...
                       time_mode, ...
                       time0, time1, ...
                       orbit0, orbit1);

    mode = EnumOrbit.XO_INTERPOL_MODEL_DEFAULT;
    extraChoice = EnumOrbit.XO_ORBIT_EXTRA_ALL_RESULTS;

    n = 1;
    longArray=zeros(points,1);
    latArray=zeros(points,1);
    altArray=zeros(points,1);
    timeArray=zeros(points,1);

    time0=5378.5;
    time1=5388.5;
    for time_i = time0 : (time1-time0)/points : time1
      stateVector = orbitId1.osvCompute( mode, time_ref, time_i );
      stateVectorExtra = orbitId1.osvComputeExtra(extraChoice);

      %NOTE: the indices of the arrays are one-based (not zero-based like in
Java)
      %check this: http://uk.mathworks.com/help/matlab/matlab_external/working-
with-java-arrays.html

longArray(n)=stateVectorExtra.modelIndep(EnumOrbit.XO_ORBIT_EXTRA_GEOC_LONG +
1);
      latArray(n) =
stateVectorExtra.modelIndep(EnumOrbit.XO_ORBIT_EXTRA_GEOD_LAT + 1);
```

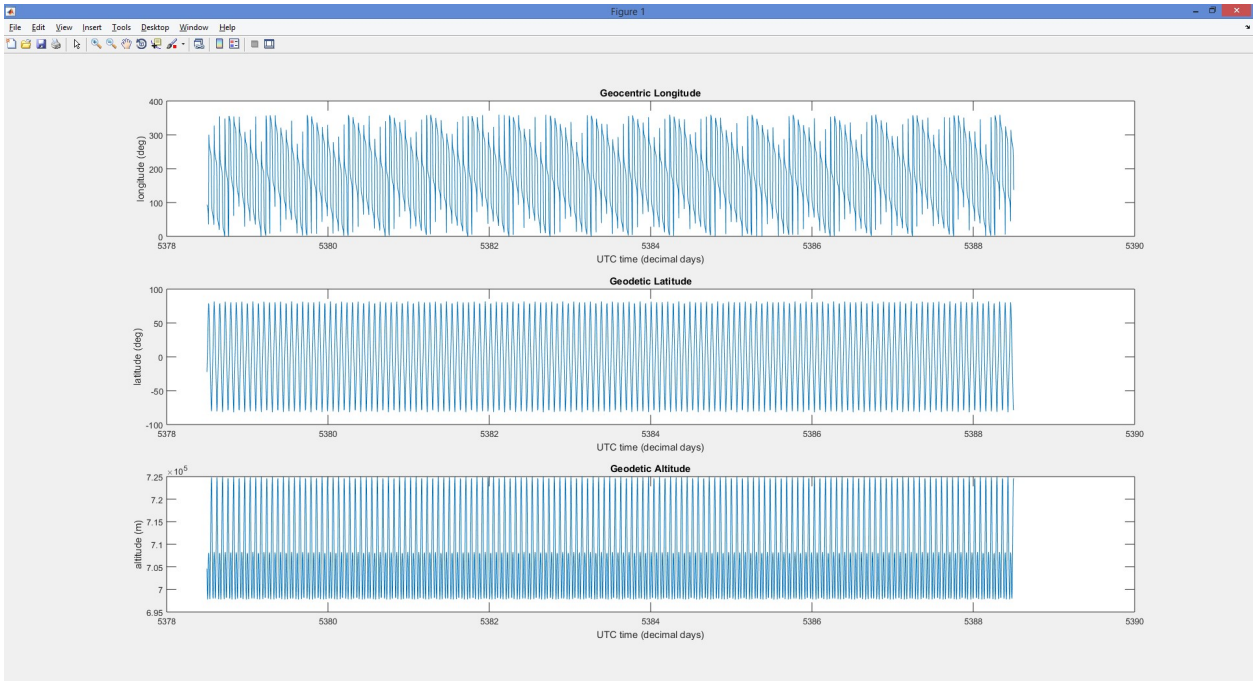
```
      altArray(n) =
stateVectorExtra.modelIndep(EnumOrbit.XO_ORBIT_EXTRA_GEOD_ALT + 1);
      timeArray(n) = time_i;

      n=n+1;
    end

 figure
 ax1 = subplot(3,1,1); % top subplot
 ax2 = subplot(3,1,2); % middle subplot
 ax3 = subplot(3,1,3); % bottom subplot

 plot(ax1,timeArray,longArray);
 title(ax1, 'Geocentric Longitude');
 xlabel(ax1, 'UTC time (decimal days)');
 ylabel(ax1, 'longitude (deg)');

 plot(ax2,timeArray,latArray);
 title(ax2, 'Geodetic Latitude');
 xlabel(ax2, 'UTC time (decimal days)');
 ylabel(ax2, 'latitude (deg)');

 plot(ax3,timeArray,altArray);
 title(ax3, 'Geodetic Altitude');
 xlabel(ax3, 'UTC time (decimal days)');
 ylabel(ax3, 'altitude (m)');

catch e
    if (isa(e, 'matlab.exception.JavaException') && isa(e.ExceptionObject,
'EECFI.CfiError'))
        cfiError = e.ExceptionObject;
        errorMessages = java.util.Vector;
        errorMessages = cfiError.getMsg(errorMessages);
        fprintf('CFI error:\n');
        for i = 0:errorMessages.size()-1
            fprintf('%s\n', errorMessages.elementAt(i));
        end
    else
        e
    end
end

end
```

3.7.2. Calling the example

In the MATLAB **Command Window** type:

OrbitTest ('[path_to_file]\orbit.file', 3000)

The results are displayed as a drawing:



3.7.3. Example of orbit file (OSF file with one orbit change)

```
<?xml version="1.0"?>
<Earth_Explorer_File xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://eop-cfi.esa.int/CFI http://eop-cfi.esa.int/CFI/EE_CFI_SCHEMAS/EO_OPER_MPL_ORBSCT_0102.XSD" schemaVersion="1.2" xmlns="http://eop-cfi.esa.int/CFI">
  <Earth_Explorer_Header>
    <Fixed_Header>
      <File_Name>OSF_ALL</File_Name>
      <File_Description>Reference Orbit Scenario File</File_Description>
      <Notes/>
      <Mission>Sentinel1</Mission>
      <File_Class>FILE_CLASS</File_Class>
      <File_Type>MPL_ORBSCT</File_Type>
      <Validity_Period>
        <Validity_Start>UTC=2002-02-28T22:00:03</Validity_Start>
        <Validity_Stop>UTC=9999-99-99T99:99:99</Validity_Stop>
      </Validity_Period>
      <File_Version>0001</File_Version>
      <Source>
        <System>FH_SYSTEM</System>
        <Creator>EO_ORBIT:xo_gen_osf_create</Creator>
```

```
      <Creator_Version>4.2</Creator_Version>
      <Creation_Date>UTC=2011-03-04T08:51:57</Creation_Date>
    </Source>
  </Fixed_Header>
  <Variable_Header>
    <Time_Reference>UT1</Time_Reference>
  </Variable_Header>
</Earth_Explorer_Header>
<Data_Block type="xml">
  <List_of_Orbit_Changes count="1">
    <Orbit_Change>
      <Orbit>
        <Absolute_Orbit>1</Absolute_Orbit>
        <Relative_Orbit>1</Relative_Orbit>
        <Cycle_Number>1</Cycle_Number>
        <Phase_Number>1</Phase_Number>
      </Orbit>
      <Cycle>
        <Repeat_Cycle unit="day">12</Repeat_Cycle>
        <Cycle_Length unit="orbit">175</Cycle_Length>
        <ANX_Longitude unit="deg">0.000000</ANX_Longitude>
        <MLST>18:00:00.000000</MLST>
        <MLST_Drift unit="s/day">0.000000</MLST_Drift>
        <MLST_Nonlinear_Drift>
          <Linear_Approx_Validity unit="orbit">99999</Linear_Approx_Validity>
          <Quadratic_Term unit="s/day^2">0.000000</Quadratic_Term>
          <Harmonics_Terms num="0">
          </Harmonics_Terms>
        </MLST_Nonlinear_Drift>
      </Cycle>
      <Time_of_ANX>
        <TAI>TAI=2011-09-01T18:00:01.192396</TAI>
        <UTC>UTC=2011-09-01T17:59:27.192396</UTC>
        <UT1>UT1=2011-09-01T17:59:27.192398</UT1>
      </Time_of_ANX>
    </Orbit_Change>
  </List_of_Orbit_Changes>
</Data_Block>
```

</Earth_Explorer_File>

4. CALLING EOCFI FROM IDL

The following versions of IDL and EOCFI were used to write this technical note:

- IDL Version 8.8.2 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions
- EOCFI 4.24 (C libraries)

4.1. Creating a function

In order to create a function in IDL to call the EOCFI it is used the IDL function CALL_EXTERNAL over a previously created library compiled with the code written in C where the actual functions of the EOCFI libraries are called.

The arguments to use the CALL_EXTERNAL function are described below using the example described in section 4.6.1.1:

```
IF (CALL_EXTERNAL(library                                   $
     , func                                                 $ ; function name
     , time, n_times, orbit_files                           $ ; inputs
     , n_files,                                             $ ; inputs (passed by
value)
     , lat, lon                                             $ ; outputs
     , VALUE=[0,1,0,1,0,0]                                  $ ; passing method: 0: by
reference, 1: by value
     , /CDECL, AUTO_GLUE=auto_glue                          $ ; keywords 1
     , VERBOSE=verbose, SHOW_ALL_OUTPUT=verbose             $ ; keywords 2
     , UNLOAD=debug                                         $ ; keyword for debugging
purpose 1
     , IGNORE_EXISTING_GLUE=debug                           $ ; keyword for debugging
purpose 2 (with AUTO_GLUE only)
     , COMPILE_DIRECTORY=compile_directory                  $ ; keyword for debugging
purpose 3 (with AUTO_GLUE only)
     , CC=cc_string                                         $ ;
       ) EQ 1) THEN BEGIN
            success = 1
ENDIF ELSE message, 'External call to call_eop_cfi failed'
```

Where:

- `library`: name of the compiled library that is called. In this case is: *call_eop_cfi.so*
- `func`: name of the C function to be executed.
- `time`: vector array of the times used by the EOCFI to initialise the time id.
- `n_times`: size of the time array
- `orbit_files`: vector array with the orbit files used by the EOCFI.
- `n_files`: number of orbit files used as input.
- `lat and lon`: output values calculated by the EO CFI.
- `VALUE`: array that encodes if the values are passed as by reference of by value.

Also are available several keywords used in the CALL_EXTERNAL function for debugging purposes, please refer to the existing literature for more details: [RD6]

In order to create the library file `call_eop_cfi.so`, it is used the function MAKE_DLL:

```
MAKE_DLL, c_source, 'call_eop_cfi', export_rtns       $
     , INPUT_DIRECTORY=source_directory        $
     , COMPILE_DIRECTORY=compile_directory       $
     , DLL_PATH=lib_directory          $
     , EXTRA_CFLAGS=cc_flags           $
     , EXTRA_LFLAGS=ld_flags           $
     , SHOW_ALL_OUTPUT=verbose           $
     , VERBOSE=verbose
```

Where:

- `c_source`: A string (scalar or array) giving the names of the input C source files to be compiled by MAKE_DLL. These names should not include any directory path information or the .c suffix, they are simply the base file names. In the example below: *call_eop_cfi*.

  The input directory is specified using the INPUT_DIRECTORY keyword, and the .c file suffix is assumed.

- `OutputFile ('call_eop_cfi')`: The base name of the resulting sharable library. This name should not include any directory path information or the sharable library suffix, which differs between platforms (for example: .so, .a, .sl, .exe, .dll).

  The output directory can be specified using the OUTPUT_DIRECTORY keyword.

  If the OutputFile argument is omitted, the first name given by InputFile is used as the base name of output file.

- `Export_rtns`: A string (scalar or array) specifying the names of the routines to be exported (i.e., that are visible for linking) from the resulting sharable library.

- `INPUT_DIRECTORY`: If present, the path to the directory containing the source C files listed in *c_source*. If INPUT_DIRECTORY is not specified, the directory given by COMPILE_DIRECTORY is assumed to contain the files.

- `COMPILE_DIRECTORY`: To build a sharable library, MAKE_DLL requires a place to create the necessary intermediate files and possibly the final library itself.

- `DLL_PATH`: If present, the name of a variable to receive the complete file path for the newly created sharable library. The location of the resulting sharable library depends on the setting of the OUTPUT_DIRECTORY or COMPILE_DIRECTORY keywords as well as the !MAKE_DLL.COMPILE_DIRECTORY system variable, and different platforms use different file suffixes to indicate sharable libraries. Use of the DLL_PATH keyword makes it possible to determine the resulting file path in a simple and portable manner.

Also are available several keywords used in the MAKE_DLL function for debugging purposes, please refer to the existing literature for more details: [RD7]

4.2. Mapping of data

IDL accepts two ways to pass data between IDL to the C coded program: by reference or by value. In order to define which data is passed by each method it is used the variable VALUE in the CALL_EXTERNAL function.

The VALUE variable is a byte array, with as many elements as there are optional parameters, indicating the method of parameter passing. Arrays are always passed by reference. For instance, if parameter *Pi* is a scalar, it is passed by reference if *VALUE[i]* is *0*; and by value if it is non-zero.

4.3. Configurations needed

In order to generate the library it is needed to have the EOCFI libraries located in a known folder that in the example is defined by the environment variable ***CFI_HOME***.

4.4. Persistency of data

The data passed between IDL and the C program has a limited persistency, only the output parameters tagged as *outputs* in the C program (***out_lat*** and ***out_lon*** in the example) can be accessed from IDL.

4.5. Execution Procedure description

In this section is showed an example of how the library is compiled and then is used.

4.5.1. Compile Procedure description

In order to compile the C code the following procedure must be executed. Depending on specific needs this procedure can be modified:

- Change to the directory where the *make_dll.pro* is located
- Start the IDL command line and execute the IDL command: `LibName = make_dll()`

```
 build]$> idl
IDL Version 8.8.2 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions


IDL> LibName = make_dll()
% Compiled module: MAKE_DLL.
Compile Directory:/home/user/CALL_IDL2CFI/build/../lib
Build external C-Library!
IDL>
```

This procedure generates the library file call_eop_cfi.so in the folder:

- /home/user/CALL_IDL2CFI/build/../lib

4.5.2. Execution Procedure description

Once the library has been successfully compiled it can be used by executing the `call_eop_cfi` function described in section 4.6.1.2. Again this procedure is just a demonstration of the CALL_EXTERNAL capabilities, the procedure can be modified in order to cover specific needs.

- Change to the directory where the source code *call_eop_cfi.pro* is located
- Start the IDL command line and execute the following IDL commands:

```
 source]$> idl
IDL Version 8.8.2 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions


IDL> orbit_files =
'[path_to_orbit_file]/MPL_ORBPRE_20150423T223001_20150602T223001_0001.EOF'
IDL> time_jd = [5592.1, 5592.2, 5592.3, 5592.4, 5592.5, 5592.6, 5592.7 ]
IDL> result = call_eop_cfi(time_jd, ORBIT_FILES= orbit_files, AUTO_GLUE=1, LATITUDE=latitude,
LONGITUDE=longitude,  /DEBUG)
% CALL_EOP_CFI: TBX -> Calling to EE CFI routines ...
% CALL_EOP_CFI: call_eop_cfi -> library name: /home/user/CALL_IDL2CFI/lib/call_eop_cfi.so
% CALL_EOP_CFI: call_eop_cfi -> Input argument Number of Orbit Files:           1
% CALL_EOP_CFI: call_eop_cfi -> Input argument Number of Times:           7
% CALL_EOP_CFI: call_eop_cfi -> Input argument Start Time:       5592.1001
% CALL_EOP_CFI: call_eop_cfi -> Input argument Stop Time:       5592.7002
% CALL_EOP_CFI: call_eop_cfi -> Input argument Orbit Files:
[path_to_orbit_file]/MPL_ORBPRE_20150423T223001_20150602T223001_0001.EOF


   - validity times = ( 5591.937515 , 5592.007445 )


   XO_PROPAG # 0
   -    time = 5592.100098
   -  pos[0] = -2516597.470059
   -  pos[1] = 2292105.014206
   -  pos[2] = 6299627.648284
   -  vel[0] = -2867.689218
   -  vel[1] = 6111.673220
   -  vel[2] = -3361.927516
   -  acc[0] = 3.600444
   -  acc[1] = -2.049223
   -  acc[2] = -6.831957


   - absolute orbit = 14732 Time since anx = 1963.249020
/...../
```

```
/...../
/...../


% CALL_EOP_CFI: call_eop_cfi -> Number of latitude points returned is :          7
% CALL_EOP_CFI: call_eop_cfi -> Number of longitude points returned is :          7
% CALL_EOP_CFI: call_eop_cfi -> End of call to EE CFI
IDL>
IDL> print,'Latitude: ', latitude
Latitude:       61.758465      -81.148409       63.935787      -40.231045       15.590802       8.7118981
-33.434597
IDL> print,'Longitude: ', longitude
Longitude:       137.67289       339.70296       211.72973       6.2550804       155.10814
302.78652       91.114644
IDL>
```

4.6. Source code Examples

In this section are presented the examples used in the function above to show how are called the EOCFI libraries coded in C from IDL. The examples are the following:

- IDL source code, compiling function (`make_dll.pro`): where it is compiled the C code where the actual EO CFI libraries are called.
- IDL source code, calling function (`call_eop_cfi.pro`): where it is called the compiled library call_eop_cfi.so with the inputs: time array and orbit files.
- C source code (`call_eop_cfi.c`): Example program to show the way to call the EOCFI functions.

4.6.1. IDL Source Code

4.6.1.1. Compiling Function

This function executed in the IDL command line is used to generate the library that will be called upon in the function showed in previous section. The function generates the library file: call_eop_cfi.so in the defined directory: `output_directory`.

```
;+
; NAME:
; make__dll
;
; PURPOSE:
; This function builds the sharable library and
; returns the name of the sharable library which will contain all the external
; C code to be used by the IDL CALL_EXTERNAL routine.
;
; CATEGORY:
; Dynamic Linking
;
; CALLING SEQUENCE
; LibName = make_dll()
;
```

```
; INPUTS:
; None
;
; OUTPUTS:
; Scalar string containing path to sharable library.
; OPTIONAL OUTPUTS:
; COMPILE_DIRECTORY = Scalar string containing the path of the directory
;       used to compile the sharable library.
;
; KEYWORDS:
; VERBOSE
; If set, cause the underlying MAKE_DLL to show the commands it
; executes to build the sharable library, and all of the output
; produced by those commands. If not set, this routine does its
; work silently.
;
; SIDE EFFECTS
; The sharable library is build using the MAKE_DLL procedure.
; All previous linked files or images are overwritten / removed.
;
; RESTRICTIONS
; For MAKE_DLL to be successful, a C compiler compatible with the
; one described in the !MAKE_DLL system variable must be installed
; on the system.
;
; PROCEDURE:
; Caches the library path in a private COMMON block. On the first
; call, builds the library and sets its name in the COMMON block for
; following calls.
;-

FUNCTION make_dll                                $
     , VERBOSE=verbose                           $ ; keyword - flag
     , COMPILE_DIRECTORY=compile_dir_out         ; output


     COMMON GET_IMAGE_SHLIB_CS_MF, lib_directory

     lib_directory = STRING('')


      ; Location of the sharable library files of the EOP-CFI

     eop_cfi_directory = getenv('CFI_HOME')

  include_directory = eop_cfi_directory+'include'
  library_directory = eop_cfi_directory+'lib/LINUX64'


      ; Location of the MAKE_DLL file generated after the compilation of the
external C-routines

     CD, CURRENT=current_dir

     source_directory = current_dir+'/../source'
     output_directory = current_dir+'/../lib'
```

```
    compile_directory = (current_dir EQ './') ? !MAKE_DLL.COMPILE_DIRECTORY :
output_directory

    !MAKE_DLL.COMPILE_DIRECTORY = compile_directory

    PRINT, 'Compile Directory:', compile_directory


    cfi_libraries    = '-lexplorer_orbit -lexplorer_lib  -
lexplorer_data_handling -lexplorer_file_handling -lxml2 -lexplorer_visibility -
lexplorer_pointing -lexplorer_data_handling -lm -lc -lexplorer_lib'


   ; here the names of all used C files must be listed
  ; string of filenames without any path or extension
  ; equal the following calls:
  ; c_files = FILE_SEARCH(call_ex_dir, '*.c')
  ; c_source = FILE_BASENAME(c_files, '.c')

    c_source = [  'call_eop_cfi'  $
    ]

     ; here the names of all routines present in the used C files must be
listed

    export_rtns = [  'call_eop_cfi_natural' $
    , 'call_eop_cfi'    $
    ]

; removed -ansi flag so the comments with // work correctly
  cc_flags = '-I"'+include_directory+'"  -D_REENTRANT'

  ld_flags = '-L"'+library_directory+'" '+cfi_libraries


     ; Use MAKE_DLL to build the widget_call_ex sharable library in the
     ; !MAKE_DLL.COMPILE_DIRECTORY directory.
     ;
     ; Normally, you wouldn't use VERBOSE, or SHOW_ALL_OUTPUT once your
     ; work is debugged, but as a learning exercize it can be useful to
     ; see all the underlying work that gets done. If the user specified
     ; VERBOSE, then use those keywords to show what MAKE_DLL is doing.

     PRINT, 'Build external C-Library!'

    MAKE_DLL, c_source, 'call_eop_cfi', export_rtns       $
      , INPUT_DIRECTORY=source_directory        $
      , COMPILE_DIRECTORY=compile_directory       $
      , DLL_PATH=lib_directory          $
      , EXTRA_CFLAGS=cc_flags           $
      , EXTRA_LFLAGS=ld_flags           $
      , SHOW_ALL_OUTPUT=verbose           $
      , VERBOSE=verbose
```

```
      RETURN, lib_directory

END
```

4.6.1.2. Calling Function

Below is the example used to describe the usage of the CALL_EXTERNAL function to interface with EOCFI coded in C. The function retrieves the input time array and the input orbit files:

```
;+
; NAME:
; call_eop_cfi
;
; PURPOSE:
; This IDL function serves as an interface between the EOP CFI coded in C and IDL.
; The purpose of this function is to retrieve the parameters latitude, longitude
;
;
; CATEGORY:
; Dynamic Linking Examples
;
; CALLING SEQUENCE
; result = call_eop_cfi( time, ORBIT_FILES=orbit_files, $
;      [/DEBUG], [/VERBOSE], [/AUTO_GLUE],              $
;      [LATITUDE=latitude], [LONGITUDE=longitude]
;
; INPUTS:
; time
; Scalar or array of type DOUBLE
;
; orbit files
; Scalar or array of type STRING
;
; OUTPUTS:
; latitude
; Scalar or array of type DOUBLE (will have the same size than the time input)
;
; longitude
; Scalar or array of type DOUBLE (will have the same size than the time input)
;
;
; KEYWORDS:
; AUTO_GLUE
; Use the AUTO_GLUE keyword to CALL_EXTERNAL to call a version of
; the external C function that has a natural C interface rather
; than the IDL portable calling convension (argc, argv).
;
;       DEBUG
; If this keyword is unset, this routine will return to the
; caller on any error. If this keyword is set, this routine will
; stop at the point of the error.
;
; VERBOSE
; If set, cause the underlying MAKE_DLL to show the commands it
; executes to build the sharable library, and all of the output
; produced by those commands. If not set, this routine does its
; work silently.
;
; SIDE EFFECTS
; Sharable library of example code is built.
```

```
;
; RESTRICTIONS
; None.
;
; PROCEDURE:
; Use make_dll.pro to build the sharable library.
;
; EXAMPLE CALL:
;
;   result = call_eop_cfi(time, ORBIT_FILES=orbit_files, LATITUDE=latitude,
LONGITUDE=longitude)
;
;-

FUNCTION call_eop_cfi, time_in, ORBIT_FILES=orbit_files $   ; inputs
  , AUTO_GLUE=auto_glue, DEBUG=debug, VERBOSE=verbose     $   ; keywords
  , LATITUDE=lat_out, LONGITUDE=lon_out           ; outputs
; , ECLIPSE_FLAG=ecl_out, SUN_ZENITH_ANGLE=sza_out      $ ; outputs
; , MEAN_LOCAL_SOLAR_TIME=mlst_out              ; outputs

  message,  'call_eop_cfi ->  Calling to EE CFI routines ...',/INF

    success = -1

     lat    = -1L
     lon    = -1L



    IF NOT(KEYWORD_SET(debug)) THEN ON_ERROR,2

     ; Type checking: Any missing (undefined) arguments will be set
     ; to a default value. All arguments will be forced to a scalar
     ; of the apropriate type, which may cause errors to be thrown
     ; if structures are passed in. Local variables are used so that
     ; the values and types of the user supplied arguments don't change.

     time = (SIZE(time_in,/TYPE) EQ 0) ? DOUBLE(-1) : DOUBLE(time_in)

     func = KEYWORD_SET(auto_glue) ? 'call_eop_cfi_natural' : 'call_eop_cfi'

     ; get name and directory of the sharable library file produced by the make_dll
routine
     ; (UNIX: sharable library , WINDOWS: dll)
     ; generated after the compilation of the external C-routines

  CD, CURRENT=current_dir

     library_path  = current_dir + '/../lib'
     library   =FILE_SEARCH(library_path, '/call_eop_cfi.so')

     IF (library EQ '') OR (FILE_TEST(library, /READ) EQ 0) THEN BEGIN

       message, 'External call failed! Library was not found. Please make use of the
routine <make_dll.pro> to build the sharable library first.'

     ENDIF ELSE BEGIN

       message, 'call_eop_cfi -> library name: ' + string(library, FORMAT='(A)'),/INF

     ; get number of time values

     n_times = LONG(N_ELEMENTS(time))
```

```
      ; get number of orbit files

      n_files = LONG(N_ELEMENTS(orbit_files))

      IF (n_files EQ 0) THEN BEGIN
        message, 'No Orbit Files passed as input!'
      ENDIF


      IF (time[0] NE -1) THEN BEGIN

        IF ARG_PRESENT(lat_out) OR ARG_PRESENT(lon_out) THEN BEGIN

            lat  = DBLARR(N_ELEMENTS(time))
            lon  = DBLARR(N_ELEMENTS(time))

        ENDIF

      message, 'call_eop_cfi -> Input argument Number of Orbit Files: ' +
string(n_files, FORMAT='(I)'),/INF
      message, 'call_eop_cfi -> Input argument Number of Times: ' + string(n_times,
FORMAT='(I)') ,/INF
      message, 'call_eop_cfi -> Input argument Start Time: ' + string(MIN(time))
,/INF
      message, 'call_eop_cfi -> Input argument Stop Time: ' + string(MAX(time))     ,/INF
      message, 'call_eop_cfi -> Input argument Orbit Files: '+ orbit_files         ,/INF

      IF (CALL_EXTERNAL(library       $
            , func                                                  $ ; function name
                      , time, n_times, orbit_files                  $ ; inputs
            , n_files                                               $ ; inputs (passed by
value)
            , lat, lon                                              $ ; outputs
            , VALUE=[0,1,0,1,0,0]                                 $ ; passing method: 0: by
reference, 1: by value
            , /CDECL, AUTO_GLUE=auto_glue                           $ ; keywords 1
            , VERBOSE=verbose, SHOW_ALL_OUTPUT=verbose                $ ; keywords 2
            , UNLOAD=debug                                            $ ; keyword for
debugging purpose 1
            , IGNORE_EXISTING_GLUE=debug                            $ ; keyword for debugging
purpose 2 (with AUTO_GLUE only)
            , COMPILE_DIRECTORY=compile_directory                   $ ; keyword for debugging
purpose 3 (with AUTO_GLUE only)
      ) EQ 1) THEN BEGIN
            success = 1
      ENDIF ELSE message, 'External call to call_eop_cfi failed'
            ENDIF ELSE print, 'Input time not valid'
      ENDELSE

      IF ARG_PRESENT(lat_out) THEN lat_out = lat
      IF ARG_PRESENT(lon_out) THEN lon_out = lon

  lat = 0
  lon = 0

      message,'call_eop_cfi -> Number of latitude points returned is : '+
string(N_ELEMENTS(lat_out)),/INF
      message,'call_eop_cfi -> Number of longitude points returned is : '+
string(N_ELEMENTS(lon_out)),/INF
      message,'call_eop_cfi -> End of call to EE CFI',/INF

      RETURN, success
```

```
END
```

4.6.2. C Source Code

The C program is an example of usage of the EOCFI functions `xo_osv_compute` and `xo_osv_compute_extra`, obtaining for each value of the input time array some orbital parameters such as: latitude, longitude, altitude, time_since_anx or the abs_orbit.

From these the latitude and longitude are passed back to IDL.

```
/*
 *  call_eop_cfi.c
 *
 */


 /*****************************************************************************
 *
 * Project     : EXPCFI
 * Copyright   : DEIMOS Space S.L.
 *
 * Purpose     : Example program to show the way to call the EO CFI from IDL.
 *
 * Created by  : Jonathan L. Gimeno on May 2015.
 *
 *****************************************************************************/

#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#include <explorer_visibility.h>
#include <explorer_data_handling.h>
#include <explorer_orbit.h>
#include <explorer_lib.h>


#include <idl_export.h>    /* IDL external definitions */
```

```
#define XO_MAX_STR_LENGTH        256


#define MAX_OUT_LEN 511    /* truncate any string longer than this */



/*
 * IDL_STRING is declared in idl_export.h like this:
 *
 *    typedef struct {
 *       IDL_STRING_SLEN_T slen;        Length of string, 0 for null
 *       short stype;                   Type of string, static or dynamic
 *       char *s;                       Address of string
 *    } IDL_STRING;
 *
 * However, you should rely on the definition in idl_export.h instead
 * of declaring your own string structure.
*/




/* Main program */
/* ------------ */


int     call_eop_cfi_natural(double     *in_times,     IDL_LONG     in_n_times,     IDL_STRING
*in_orbit_files, IDL_LONG in_n_orbit_files, double *out_lat, double *out_lon)


/*
 * Version with natural C interface. This version can be called directly
 * by IDL using the AUTO_GLUE keyword to CALL_EXTERNAL.
 *
 * input:
 * in_times   - Pointer to array of double values for TAI time        [JD2000]
 * in_n_times - Long scalar of number of time doubles pointed at by the pointer in_times.
 * in_orbit_files  - Pointer to array of string values for orbit filenames
 * in_n_orbit_files- Long scalar of numbers of orbit file strings pointed at by the
pointer in_orbit_files.


 *
 * output:
 * out_lat    - Pointer to array of double values for the latitude      [0    =< degress   <
+360]
```

```
 * out_lon    - Pointer to array of double values for the longitude        [-90 =< degress
=< +90]
 * out_alt    - Pointer to array of double values for the altitude      [meters]

 */

{

  /*  Check the size of the array passed in. n should be > 0.*/
  if (in_n_orbit_files < 1) return(-1);



  /* Earth Explorer Ids.: They should be initialized to NULL allways!!!! */
  xl_time_id     time_id     = {NULL};
  xo_orbit_id    orbit_id    = {NULL};
  xl_model_id    model_id    = {NULL};

  xl_time_id_init_data  time_id_init_data;
  xo_orbit_id_init_data orbit_id_init_data;

 /* Accessors data */
  long num_rec;
  xo_validity_time val_time;

  xo_propag_id_data   propag_data;

  long i, j;

  /* error handling */
  long status,
       ierr[XO_ERR_VECTOR_MAX_LENGTH],
       xv_ierr[XV_ERR_VECTOR_MAX_LENGTH],
       n = 0,
       func_id,
       code[XO_MAX_COD];           /* Error codes vector */

  char msg[XO_MAX_COD][XO_MAX_STR];     /* Error messages vector */

  /* common variables */
  long sat_id      = XO_SAT_SENTINEL_2A;
```

```
  long propag_model  = XO_PROPAG_MODEL_MEAN_KEPL;
  long time_ref_utc  =  XO_TIME_UTC,  time_ref_ut1  =  XO_TIME_UT1,  time_ref_tai  =
XO_TIME_TAI;

  double pos[3];
  double vel[3];
  double acc[3];

  double pos_ini[3];
  double vel_ini[3];

  double time0,
         time,
    val_time0,
    val_time1;

  long time_init_mode,
       drift_mode;

  long time_model;

  double latitude,
    longitude,
    altitude,
    elapsed_anx;


  long orbit0, orbit1, n_files, num, i_loop;


  double time1;

  /* variables for xo_interpol_init */

  long interpol_model;

  /* variables for xo_osv_compute_extra */

  long extra_choice;
  double orbit_model_out[XO_ORBIT_EXTRA_NUM_DEP_ELEMENTS],
```

```
        orbit_extra_out[XO_ORBIT_EXTRA_NUM_INDEP_ELEMENTS];


/* variables for xo_orbit_info_from_... */

long abs_orbit, rel_orbit, cycle, phase;
double result_vector[XO_ORBIT_INFO_EXTRA_NUM_ELEMENTS];

/* Other variables */

double time_since_anx, time_anx, time_ini, dt;

/* xl_time_ref_init_file */
long   trif_time_model, trif_n_files, trif_time_init_mode, trif_time_ref ;
char   *trif_time_file[1];
double trif_time0, trif_time1, trif_val_time0, trif_val_time1;
long   trif_orbit0, trif_orbit1;


/* xl_time_ref_init variable declaration */
/* ------------------------------------ */
long tri_sat_id, tri_orbit_num;
double tri_anx_time, tri_orbit_duration;
double tri_time[XL_TIME_TRANS_DIM_MAX];
long tri_ierr[XL_NUM_ERR_TIME_REF_INIT];

/* orbit initilization */
/* ------------------- */
long time_mode, orbit_mode;
char* files[1];

char **trif_time_files;
char **input_files;


 /* Set error handling mode to SILENT  */
 /* ---------------------------------- */

 xl_silent();
 xo_silent();
```

```
   xv_silent();

   /*============================================================================*/
   /*  xo_orbit_init_file use with multiple files */
/*============================================================================*/

   /* Time Initialization */
   /* ------------------- */

   trif_time_ref        = XL_TIME_TAI;
   propag_model = XO_PROPAG_MODEL_MEAN_KEPL + XO_PROPAG_MODEL_AUTO;

   trif_time_model= XL_TIMEMOD_FOS_PREDICTED;
   orbit_mode = XO_ORBIT_INIT_AUTO;



   trif_time_init_mode  = XL_SEL_FILE;

   trif_n_files    = in_n_orbit_files;

   trif_time_files= (char **) calloc(trif_n_files,sizeof(char*));


   for(i=0; i < trif_n_files ; i++) {
   trif_time_files[i] = (char *) calloc(MAX_OUT_LEN+1,sizeof(char));
   strncpy(trif_time_files[i], in_orbit_files[i].s, MAX_OUT_LEN);
   /*fprintf(stdout, trif_time_files[i]);*/
   }

   status = xl_time_ref_init_file(&trif_time_model, &trif_n_files, trif_time_files,
                                  &trif_time_init_mode,    &trif_time_ref,    &trif_time0,
&trif_time1,
                                  &trif_orbit0,       &trif_orbit1,       &trif_val_time0,
&trif_val_time1,
                                  &time_id, ierr);
   if (status != XO_OK)
   {
     func_id = XL_TIME_REF_INIT_FILE_ID;
     xo_get_msg(&func_id, ierr, &n, msg);
```

```
    xl_print_msg(&n, msg);
  }

  /* Orbit initialization */
  /* -------------------- */

  time_init_mode = XO_SEL_FILE;

  n_files      = in_n_orbit_files;

  input_files    =(char **) calloc(n_files,sizeof(char*));

  for(i=0; i < n_files ; i++) {

     input_files[i] = (char *) calloc(MAX_OUT_LEN+1,sizeof(char));

     strncpy(input_files[i], in_orbit_files[i].s, MAX_OUT_LEN);
     /*fprintf(stdout, input_files[i]);*/
  }


  status =  xo_orbit_init_file(&sat_id, &model_id, &time_id,
                               &orbit_mode, &n_files, input_files,
                               &time_init_mode, &time_ref_utc,
                               &time0, &time1, &orbit0, &orbit1,
                               &val_time0, &val_time1, &orbit_id,
                               ierr);

  if (status != XO_OK)
  {
    func_id = XO_ORBIT_INIT_FILE_ID;
    xo_get_msg(&func_id, ierr, &n, msg);
    xo_print_msg(&n, msg);
    if (status <= XO_ERR) return(XO_ERR); /*Lets the execution continue if there are
warnings.*/
    /*return -1;*/
  }

 status = xo_orbit_get_propag_config(&orbit_id, &propag_data);
```

```
  val_time0 = propag_data.propag_osv.val_time.start;
  val_time1 = propag_data.propag_osv.val_time.stop;

fprintf(stdout, "\n\t- validity times = ( %lf , %lf )", val_time0, val_time1 );



  num = in_n_times;

  for (i_loop = 0; i_loop < num; i_loop++)
  {

    time = in_times[i_loop];

     fprintf(stdout, "\n\n\tXO_PROPAG # %li", (long) i_loop);

    status = xo_osv_compute(&orbit_id, &propag_model, &time_ref_utc, &time,
                            /* outputs */
                            pos, vel, acc, ierr);

    if (status != XO_OK)
    {
      func_id = XO_OSV_COMPUTE_ID;
      xo_get_msg(&func_id, ierr, &n, msg);
      xo_print_msg(&n, msg);
    }



    fprintf(stdout, "\n\t-    time = %lf", time );
    fprintf(stdout, "\n\t-  pos[0] = %lf", pos[0] );
    fprintf(stdout, "\n\t-  pos[1] = %lf", pos[1] );
    fprintf(stdout, "\n\t-  pos[2] = %lf", pos[2] );
    fprintf(stdout, "\n\t-  vel[0] = %lf", vel[0] );
    fprintf(stdout, "\n\t-  vel[1] = %lf", vel[1] );
    fprintf(stdout, "\n\t-  vel[2] = %lf", vel[2] );
    fprintf(stdout, "\n\t-  acc[0] = %lf", acc[0] );
    fprintf(stdout, "\n\t-  acc[1] = %lf", acc[1] );
    fprintf(stdout, "\n\t-  acc[2] = %lf", acc[2] );

fprintf(stdout, "\n");
```

```
    /* Latitude / Longitude / Altitude */

    extra_choice = XO_ORBIT_EXTRA_GEOLOCATION + XO_ORBIT_EXTRA_DEP_ANX_TIMING;

    status = xo_osv_compute_extra(&orbit_id, &extra_choice,
                                  orbit_model_out, orbit_extra_out, ierr);
  if (status != XO_OK)
  {
    func_id = XO_OSV_COMPUTE_EXTRA_ID;
    xo_get_msg(&func_id, ierr, &n, msg);
    xo_print_msg(&n, msg);
  }

    latitude   = orbit_extra_out[XO_ORBIT_EXTRA_GEOD_LAT];
    longitude    = orbit_extra_out[XO_ORBIT_EXTRA_GEOC_LONG];
    altitude   = orbit_extra_out[XO_ORBIT_EXTRA_GEOD_ALT];
    time_since_anx  = orbit_model_out[XO_ORBIT_EXTRA_DEP_SEC_SINCE_ANX];
    abs_orbit       = orbit_model_out[XO_ORBIT_EXTRA_DEP_ORBIT_NUMBER];

    fprintf(stdout, "\n\t- absolute orbit = %ld Time since anx = %lf",   abs_orbit,
time_since_anx);

  out_lat[i_loop] = latitude;
  out_lon[i_loop]= longitude;

   }

  status = xo_orbit_close(&orbit_id, ierr);
  if (status != XO_OK)
  {
    func_id = XO_ORBIT_CLOSE_ID;
    xo_get_msg(&func_id, ierr, &n, msg);
    xo_print_msg(&n, msg);
  }

  xl_time_close(&time_id, ierr);

  free (trif_time_files);
  free (input_files);
```

```
   return(1);

}

int call_eop_cfi(int argc, void* argv[])
/*
 * Version with IDL portable calling convension.
 *
 * inputs:
 * argc - Must be 6.
 * argv[0] - Address of double array.- input -> times        [JD2000]
 * argv[1] - Address of long scalar. - input -> number of time values
 * argv[2] - Address of string array.- input -> orbit files
 * argv[3] - Address of long scalar. - input -> number of orbit files
 * outputs:
 * argv[4] - Address of double array.- output -> latitude  [0   =< degress  < +360]
 * argv[5] - Address of double array.- output -> longitude [-90 =< degress =< +90]
 *
 * exit:
 * Returns TRUE for success, and FALSE for failure.
 */

{
  /* Insure that the correct number of arguments were passed in */
  if(argc != 6) return 0;

  return call_eop_cfi_natural((double *) argv[0], (IDL_LONG) argv[1], (IDL_STRING *) argv[2], (IDL_LONG) argv[3], (double *) argv[4], (double *) argv[5]);

}

#undef MAX_OUT_LEN
#undef XO_MAX_STR_LENGTH
```