

# CALCEPH Library

---

Reference manual  
version 1.1.2  
21 March 2011

M. Gastineau, J. Laskar, A. Fienga, H. Manche  
[inpop@imcce.fr](mailto:inpop@imcce.fr)

---

This manual documents how to install and use the CALCEPH Library, version 1.1.2.

Copyright © 2008, 2009, 2010, 2011, CNRS - Observatoire de besançon

Contributed by

M. Gastineau, J. Laskar, H. Manche, Astronomie et Systèmes Dynamiques, IMCCE, CNRS,  
Observatoire de Paris, UPMC

A. Fienga, Observatoire de besançon

[inpop@imcce.fr](mailto:inpop@imcce.fr)

# Table of Contents

<b>1</b>	<b>CALCEPH Library Copying conditions .....</b>	<b>1</b>
<b>2</b>	<b>Introduction to CALCEPH Library .....</b>	<b>2</b>
<b>3</b>	<b>Installing CALCEPH Library .....</b>	<b>3</b>
3.1	Installation on a Unix-like system (Linux, Mac OS X, BSD, cygwin, ...) .....	3
3.1.1	Other ‘make’ Targets .....	4
3.2	Installation on Windows system .....	4
3.2.1	Using the Windows SDK .....	4
3.2.2	Using the MinGW .....	6
<b>4</b>	<b>Reporting bugs .....</b>	<b>8</b>
<b>5</b>	<b>CALCEPH Library Interface .....</b>	<b>9</b>
5.1	C Usage .....	9
5.1.1	Headers and Libraries .....	9
5.1.1.1	Compilation on a Unix-like system .....	9
5.1.1.2	Compilation on a Windows system .....	9
5.1.2	Constants .....	9
5.1.3	Types .....	10
5.2	Fortran 2003 Usage .....	10
5.2.1	Modules and Libraries .....	10
5.2.2	Compilation on a Unix-like system .....	10
5.2.3	Compilation on a Windows system .....	10
5.2.4	Constants .....	11
5.3	Fortran 77/90/95 Usage .....	11
5.3.1	Headers and Libraries .....	11
5.3.2	Compilation on a Unix-like system .....	11
5.3.3	Compilation on a Windows system .....	11
5.3.4	Constants .....	11
5.4	Single file access functions .....	12
5.4.1	Thread notes .....	12
5.4.2	Usage .....	12
5.4.3	Functions .....	14
5.4.3.1	calceph_sopen .....	14
5.4.3.2	calceph_scompute .....	14
5.4.3.3	calceph_sgetconstant .....	16
5.4.3.4	calceph_sgetconstantcount .....	17
5.4.3.5	calceph_sgetconstantindex .....	17
5.4.3.6	calceph_sclose .....	18
5.5	Multiple file access functions .....	19

5.5.1	Thread notes .....	19
5.5.2	Usage .....	19
5.5.3	Functions .....	21
5.5.3.1	calceph_open .....	21
5.5.3.2	calceph_compute .....	21
5.5.3.3	calceph_getconstant .....	23
5.5.3.4	calceph_getconstantcount .....	24
5.5.3.5	calceph_getconstantindex .....	25
5.5.3.6	calceph_close .....	26
5.6	Error functions .....	26
5.6.1	Usage .....	26
5.6.2	calceph_seterrorhandler .....	29
<b>Appendix A Release notes .....</b>		<b>30</b>

# 1 CALCEPH Library Copying conditions

Copyright © 2008, 2009, 2010, 2011, CNRS - Observatoire de besançon

Contributed by

M. Gastineau, J. Laskar, H. Manche, Astronomie et Systèmes Dynamiques, IMCCE, CNRS, Observatoire de Paris, UPMC

A. Fienga, Observatoire de besançon

[inpop@imcce.fr](mailto:inpop@imcce.fr)

This library is governed by the CeCILL-C or CeCILL version 2 license under French law and abiding by the rules of distribution of free software. You can use, modify and/ or redistribute the software under the terms of the CeCILL-C or CeCILL version 2 license as circulated by CEA, CNRS and INRIA at the following URL "<http://www.cecill.info>".

As a counterpart to the access to the source code and rights to copy, modify and redistribute granted by the license, users are provided only with a limited warranty and the software's author, the holder of the economic rights, and the successive licensors have only limited liability.

In this respect, the user's attention is drawn to the risks associated with loading, using, modifying and/or developing or reproducing the software by the user in light of its specific status of free software, that may mean that it is complicated to manipulate, and that also therefore means that it is reserved for developers and experienced professionals having in-depth computer knowledge. Users are therefore encouraged to load and test the software's suitability as regards their requirements in conditions enabling the security of their systems and/or data to be ensured and, more generally, to use and operate it in the same conditions as regards security.

The fact that you are presently reading this means that you have had knowledge of the CeCILL-C or CeCILL version 2 license and that you accept its terms.

## 2 Introduction to CALCEPH Library

This library is designed to access the binary planetary ephemeris files, such INPOPxx and JPL DExxx ephemeris files.

This library provides a C interface and, optionnally, a Fortran 77 or 2003 interface to be called by the application.

Two groups of functions enable the access to the ephemeris files :

- Single file access functions

These functions provide access to only one ephemeris file at the same time. They are provided to make transition easier from the JPL functions, such as PLEPH, to this library.

- Multiple file access functions

These functions provide access to many ephemeris file at the same time.

This library could access to the following ephemeris

- INPOP06 or later
- DE200
- DE403
- DE405
- DE406
- DE411
- DE414
- DE418
- DE421
- DE423

Although computers have different endianness (order in which integers are stored as bytes in computer memory), the library could handle the binary ephemeris files with any endianness. This library automatically swaps the bytes when it performs read operations on the ephemeris file.

The internal format of the binary planetary ephemeris files is described in the paper : David Hoffman : 1998, A Set of C Utility Programs for Processing JPL Ephemeris Data, <ftp://ssd.jpl.nasa.gov/pub/eph/export/C-versions/hoffman/EphemUtilVer0.1.tar>

## 3 Installing CALCEPH Library

### 3.1 Installation on a Unix-like system (Linux, Mac OS X, BSD, cygwin, ...)

You need a C compiler, such as gcc. A fortran compiler, compliant with the ANSI Fortran 77 specifications, is required to compile the fortran-77/90/95 interface of the library. A fortran compiler, compliant with the Fortran 2003 specifications, is required to compile the fortran-2003 interface of the library. And you need a standard Unix ‘make’ program, plus some other standard Unix utility programs.

Here are the steps needed to install the library on Unix systems:

1. ‘tar xzf calceph-1.1.2.tar.gz’
2. ‘cd calceph-1.1.2’
3. ‘./configure’

Running `configure` might take a while. While running, it prints some messages telling which features it is checking for.

`configure` recognizes the following options to control how it operates.

‘--enable-fortran={yes|no}’

Enable or disable the fortran-77 and fortran-2003 interface. The default is ‘yes’.

‘--enable-thread={yes|no}’

Enable or disable the thread-safe version of the functions `calcephinit` and `calceph`. The default is ‘no’.

‘--disable-shared’

Disable shared library.

‘--disable-static’

Disable static library.

‘--help’

‘-h’ Print a summary of all of the options to `configure`, and exit.

‘--prefix=*dir*’

Use *dir* as the installation prefix. See the command `make install` for the installation names.

The default compilers could be changed using the variable `CC` for C compiler and `FC` for the Fortran compiler. The default compilerflags could be changed using the variable `CFLAGS` for C compiler and `FCFLAGS` for the Fortran compiler.

4. ‘make’

This compiles the CALCEPH Library in the working directory.

5. ‘make check’

This will make sure that the CALCEPH Library was built correctly.

If you get error messages, please report them to [inpop@imcce.fr](mailto:inpop@imcce.fr) (See [Chapter 4 \[Reporting bugs\]](#), [page 8](#), for information on what to include in useful bug reports).

#### 6. 'make install'

This will copy the file 'calceph.h', 'calceph.mod' and 'f90calceph.h' to the directory '/usr/local/include', the file 'libcalceph.a', 'libcalceph.so' to the directory '/usr/local/lib', and the file 'calceph.info' to the directory '/usr/local/share/info' (or if you passed the '--prefix' option to 'configure', using the prefix directory given as argument to '--prefix' instead of '/usr/local'). Note: you need write permissions on these directories.

### 3.1.1 Other 'make' Targets

There are some other useful make targets:

- 'calceph.info' or 'info'  
Create an info version of the manual, in 'calceph.info'.
- 'calceph.pdf' or 'pdf'  
Create a PDF version of the manual, in 'calceph.pdf'.
- 'calceph.dvi' or 'dvi'  
Create a DVI version of the manual, in 'calceph.dvi'.
- 'calceph.ps' or 'ps'  
Create a Postscript version of the manual, in 'calceph.ps'.
- 'calceph.html' or 'html'  
Create an HTML version of the manual, in 'calceph.html'.
- 'clean'  
Delete all object files and archive files, but not the configuration files.
- 'distclean'  
Delete all files not included in the distribution.
- 'uninstall'  
Delete all files copied by 'make install'.

## 3.2 Installation on Windows system

### 3.2.1 Using the Windows SDK

You need a C compiler, such as cl.exe, and a Windows SDK. A fortran compiler, compliant with the ANSI Fortran 77 specifications, is required to compile the fortran-77/90/95 interface of the library. A fortran compiler, compliant with the Fortran 2003 specifications, is required to compile the fortran-2003 interface of the library. It has been successfully compiled with the Windows Server 2003 R2 Platform SDK, the Windows SDK of Vista, and the Windows Server 2008 Platform SDK.

Here are the steps needed to install the library on Windows systems:

1. Expand the file 'calceph-1.1.2.tar.gz'
2. Execute the command 'cmd.exe' from the menu 'Start' / 'Execute...'  
This will open a console window
3. 'cd 'dir'\calceph-1.1.2'  
Go to the directory *dir* where CALCEPH Library has been expanded.



4. `'nmake /f Makefile.vc '`

This compiles CALCEPH Library in the working directory. This command line accepts several options :

- `CC=xx` specifies the name of the C compiler. The default value is `'cl.exe'`
- `FC=xx` specifies the name of the Fortran compiler. The default value is `'gfortran.exe'`
- `F77FUNC=naming` specifies the naming convention of the fortran 77 compiler. The possible value are: `x`, `X`, `x##-`, `X##-`.
- `ENABLEF2003={0|1}` specifies if it must compile the fortran 2003 interface.
- `ENABLEF77={0|1}` specifies if it must compile the fortran 77/90/95 interface.

5. `'nmake /f Makefile.vc check'`

This will make sure that the CALCEPH Library was built correctly.

If you get error messages, please report them to [inpop@imcce.fr](mailto:inpop@imcce.fr) (See [Chapter 4 \[Reporting bugs\]](#), [page 8](#), for information on what to include in useful bug reports).

This command line accepts several options :

- `CC=xx` specifies the name of the C compiler. The default value is `'cl.exe'`
- `FC=xx` specifies the name of the Fortran compiler. The default value is `'gfortran.exe'`
- `F77FUNC=naming` specifies the naming convention of the fortran 77 compiler. The possible value are: `x`, `X`, `x##-`, `X##-`.
- `ENABLEF2003={0|1}` specifies if it must compile the fortran 2003 interface. The default value is `'0'`.
- `ENABLEF77={0|1}` specifies if it must compile the fortran 77/90/95 interface. The default value is `'0'`.

6. `'nmake /f Makefile.vc install DESTDIR=dir'`

This will copy the file `'calceph.h'`, `'calceph.mod'` and `'f90calceph.h'` to the directory `'/usr/local/include'`, the file `'libcalceph.lib'` to the directory `dir'\lib'`, the file `'calceph.pdf'` to the directory `dir'\doc'`. Note: you need write permissions on these directories.

This command line accepts several options :

- `CC=xx` specifies the name of the C compiler. The default value is `'cl.exe'`
- `FC=xx` specifies the name of the Fortran compiler. The default value is `'gfortran.exe'`
- `F77FUNC=naming` specifies the naming convention of the fortran 77 compiler. The possible value are: `x`, `X`, `x##-`, `X##-`.
- `ENABLEF2003={0|1}` specifies if it must compile the fortran 2003 interface. The default value is `'0'`.
- `ENABLEF77={0|1}` specifies if it must compile the fortran 77/90/95 interface. The default value is `'0'`.

### 3.2.2 Using the MinGW

You need a C compiler, such as `gcc.exe`. A fortran compiler, compliant with the ANSI Fortran 77 specifications, is required to compile the fortran-77/90/95 interface of the library. A fortran compiler, such as `gfortran.exe`, compliant with the Fortran 2003 specifications, is required to compile the fortran-2003 interface of the library.

Here are the steps needed to install the library on MinGW :

1. Expand the file '`calceph-1.1.2.tar.gz`'
2. Execute the command '`cmd.exe`' from the menu '**Start**' / '**Execute...**'

This will open a console window

3. '`cd 'dir'\calceph-1.1.2`'

Go to the directory *dir* where CALCEPH Library has been expanded.

4. '`make -f Makefile.mingw`'

This compiles CALCEPH Library in the working directory. This command line accepts several options :

- `CC=xx` specifies the name of the C compiler. The default value is '`gcc.exe`'
- `FC=xx` specifies the name of the Fortran compiler. The default value is '`gfortran.exe`'
- `ENABLEF2003={0|1}` specifies if it must compile the fortran 2003 interface. The default value is '0'.
- `ENABLEF77={0|1}` specifies if it must compile the fortran 77/90/95 interface. The default value is '0'.

5. '`make -f Makefile.mingw check`'

This will make sure that the CALCEPH Library was built correctly.

If you get error messages, please report them to [inpop@imcce.fr](mailto:inpop@imcce.fr) (See [Chapter 4 \[Reporting bugs\]](#), [page 8](#), for information on what to include in useful bug reports).

This command line accepts several options :

- `CC=xx` specifies the name of the C compiler. The default value is '`gcc.exe`'
- `FC=xx` specifies the name of the Fortran compiler. The default value is '`gfortran.exe`'
- `ENABLEF2003={0|1}` specifies if it must compile the fortran 2003 interface. The default value is '0'.
- `ENABLEF77={0|1}` specifies if it must compile the fortran 77/90/95 interface. The default value is '0'.

6. '`make -f Makefile.mingw install DESTDIR=dir`'

This will copy the file '`calceph.h`', '`calceph.mod`' and '`f90calceph.h`' to the directory *dir*, the file '`libcalceph.lib`' to the directory *dir*\lib', the file '`calceph.pdf`' to the directory *dir*\doc'. Note: you need write permissions on these directories.

This command line accepts several options :

- `CC=xx` specifies the name of the C compiler. The default value is '`gcc.exe`'
- `FC=xx` specifies the name of the Fortran compiler. The default value is '`gfortran.exe`'

- `ENABLEF2003={0|1}` specifies if it must compile the fortran 2003 interface. The default value is '0'.
- `ENABLEF77={0|1}` specifies if it must compile the fortran 77/90/95 interface. The default value is '0'.

## 4 Reporting bugs

If you think you have found a bug in the CALCEPH Library, first have a look on the CALCEPH Library web page <http://www.imcce.fr/inpop>, in which case you may find there a workaround for it. Otherwise, please investigate and report it. We have made this library available to you, and it seems very important for us, to ask you to report the bugs that you find.

There are a few things you should think about when you put your bug report together. You have to send us a test case that makes it possible for us to reproduce the bug. Include instructions on the way to run the test case.

You also have to explain what is wrong; if you get a crash, or if the results printed are incorrect and in that case, in what way.

Please include compiler version information in your bug report. This can be extracted using ‘`cc -V`’ on some machines, or, if you’re using gcc, ‘`gcc -v`’. Also, include the output from ‘`uname -a`’ and the CALCEPH version.

Send your bug report to: [inpop@imcce.fr](mailto:inpop@imcce.fr). If you think something in this manual is unclear, or downright incorrect, or if the language needs to be improved, please send a note to the same address.

## 5 CALCEPH Library Interface

### 5.1 C Usage

#### 5.1.1 Headers and Libraries

All declarations needed to use CALCEPH Library are collected in the include file ‘calceph.h’. It is designed to work with both C and C++ compilers.

You should include that file in any program using the CALCEPH library:

```
#include <calceph.h>
```

##### 5.1.1.1 Compilation on a Unix-like system

All programs using CALCEPH must link against the ‘libcalceph’ library. On Unix-like system this can be done with ‘-lcalceph’, for example

```
gcc myprogram.c -o myprogram -lcalceph
```

If CALCEPH Library has been installed to a non-standard location then it may be necessary to use ‘-I’ and ‘-L’ compiler options to point to the right directories, and some sort of run-time path for a shared library.

##### 5.1.1.2 Compilation on a Windows system

#### Using the Windows SDK

All programs using CALCEPH must link against the ‘libcalceph.lib’. On Windows system this can be done with ‘libcalceph.lib’, for example

```
cl.exe /out:myprogram myprogram.c libcalceph.lib
```

If CALCEPH Library has been installed to a non-standard location then it may be necessary to use ‘/I’ and ‘/LIBPATH:’ compiler options to point to the right directories.

#### Using the MinGW

All programs using CALCEPH must link against the ‘libcalceph’ library. On the MinGW system, this can be done with ‘-lcalceph’, for example

```
gcc.exe myprogram.c -o myprogram -lcalceph
```

If CALCEPH Library has been installed to a non-standard location then it may be necessary to use ‘-I’ and ‘-L’ compiler options to point to the right directories, and some sort of run-time path for a shared library.

#### 5.1.2 Constants

##### CALCEPH\_VERSION\_MAJOR

This integer constant defines the major revision of this library. It can be used to distinguish different releases of this library.

##### CALCEPH\_VERSION\_MINOR

This integer constant defines the minor revision of this library. It can be used to distinguish different releases of this library.

### CALCEPH\_VERSION\_PATCH

This integer constant defines the patch level revision of this library. It can be used to distinguish different releases of this library.

```
#if (CALCEPH_VERSION_MAJOR>=2)
  || (CALCEPH_VERSION_MAJOR>=1 && CALCEPH_VERSION_MINOR>=1)
...
#endif
```

### CALCEPH\_MAX\_CONSTANTNAME

This integer defines the maximum number of characters, including the trailing '\0', that the name of a constant, available from the ephemeris file, could contain.

## 5.1.3 Types

`t_calcephbin` [Data type]

This type contains all information to access a binary ephemeris file.

## 5.2 Fortran 2003 Usage

### 5.2.1 Modules and Libraries

All declarations needed to use CALCEPH Library are collected in the module files 'calceph.mod'. The library is designed to work with Fortran compilers compliant with the Fortran 2003 standard. All declarations use the standard 'ISO\_C\_BINDING' module.

You should include that module in any program using the CALCEPH library:

```
use calceph
```

When a fortran string is given as a parameter to a function of this library, you should append this string with '//C\_NULL\_CHAR' because the C library works only with C string.

### 5.2.2 Compilation on a Unix-like system

All programs using CALCEPH must link against the 'libcalceph' library. On Unix-like system this can be done with '-lcalceph', for example

```
gfortran -I/usr/local/include myprogram.f -o myprogram -lcalceph
```

If CALCEPH Library has been installed to a non-standard location then it may be necessary to use '-I' and '-L' compiler options to point to the right directories, and some sort of run-time path for a shared library.

### 5.2.3 Compilation on a Windows system

All programs using CALCEPH must link against the 'libcalceph.lib'. On Windows system this can be done with 'libcalceph.lib', for example

```
gfortran.exe /out:myprogram.exe myprogram.f libcalceph.lib
```

If CALCEPH Library has been installed to a non-standard location then it may be necessary to use '/I' and '/LIBPATH:' compiler options to point to the right directories.

### 5.2.4 Constants

The following constants are defined in the module ‘calceph.mod’.

CALCEPH\_MAX\_CONSTANTNAME

This integer defines the maximum number of characters, including the trailing ‘\0’, that the name of a constant, available from the ephemeris file, could contain.

## 5.3 Fortran 77/90/95 Usage

### 5.3.1 Headers and Libraries

It is designed to work with Fortran compilers compliant with the Fortran 77, 90 or 95 standard with wrappers. All declarations are implicit, so you should take care about the types of the arguments. All functions are prefixed by ‘f90’. This interface is only provided as compatibility layer and have a small overhead due to the wrappers. So if you have a fortran compiler compliant with 2003 standard, you should use the fortran 2003 interface of this library.

All declarations needed to use CALCEPH Library are collected in the header file ‘f90calceph.h’. It is designed to work with Fortran compilers compliant with the Fortran 77 , 90 or 95 standard.

You should include that file in every subroutine or function in any program using the CALCEPH library:

```
include 'f90calceph.h'
```

### 5.3.2 Compilation on a Unix-like system

All programs using CALCEPH must link against the ‘libcalceph’ library. On Unix-like system this can be done with ‘-lcalceph’, for example

```
gfortran -I/usr/local/include myprogram.f -o myprogram -lcalceph
```

If CALCEPH Library has been installed to a non-standard location then it may be necessary to use ‘-I’ and ‘-L’ compiler options to point to the right directories, and some sort of run-time path for a shared library.

### 5.3.3 Compilation on a Windows system

All programs using CALCEPH must link against the ‘libcalceph.lib’. On Windows system this can be done with ‘libcalceph.lib’, for example

```
gfortran.exe /out:myprogram.exe myprogram.f libcalceph.lib
```

If CALCEPH Library has been installed to a non-standard location then it may be necessary to use ‘/I’ and ‘/LIBPATH:’ compiler options to point to the right directories.

### 5.3.4 Constants

The following constants are defined in the file ‘f90calceph.h’.

CALCEPH\_MAX\_CONSTANTNAME

This integer defines the maximum number of characters, including the trailing ‘\0’, that the name of a constant, available from the ephemeris file, could contain.

## 5.4 Single file access functions

This group of functions works on a single binary ephemeris file at a given instant. They use an internal global variable to store information about the current opened ephemeris file.

They are provided to have a similar interface of the fortran PLEPH function, supplied with the JPL ephemeris files. So the following call to PLEPH

```
PLEPH(46550D0, 3, 12, PV)
```

could be replaced by

```
calceph_sopen("ephemerisfile.dat")
calceph_scompute(46550D0, 0, 3, 12, PV)
calceph_sclose()
```

While the function PLEPH could access only one file in a program, these functions could access on multiple files in a program but not at same time. To access multiple files at a same time, the functions listed in the section ‘Multiple file access functions’ must be used.

When an error occurs, these functions execute error handlers according to the behavior defined by the function `calceph_seterrorhandler` (see [Section 5.6 \[Error functions\]](#), [page 26](#)).

### 5.4.1 Thread notes

If the standard I/O functions such as `fread` are not reentrant then the CALCEPH I/O functions using them will not be reentrant either.

If the library was configured with the option ‘`--enable-thread=yes`’, these functions use an internal global variable per thread. Each thread could access to different ephemeris file and compute ephemeris data at same time. But each thread must call the function `calceph_sopen` to open ephemeris file even if all threads work on the same file.

If the library was configured with the default option ‘`--enable-thread=no`’, these functions use an internal global variable per process and are not thread-safe. If multiple threads are used in the process and call the function `calceph_scompute` at the same time, the caller thread must surround the call to this function with locking primitives, such as `pthread_lock/pthread_unlock` if POSIX Pthreads are used.

### 5.4.2 Usage

The following examples, that can be founded in the directory ‘`examples`’ of the library sources, show the typical usage of this group of functions. The example in C language is ‘`csingle.c`’. The example in Fortran 2003 language is ‘`f2003single.f`’. The example in Fortran 77/90/95 language is ‘`f77single.f`’.



```

#include <stdio.h>
#include "calceph.h"

/*-----*/
/* main program */
/*-----*/
int main()
{
    int res;
    double AU, EMRAT, GM_Mer;
    double jd0=2451624;
    double dt=0.5E0;
    double PV[6];

    /* open the ephemeris file */
    res = calceph_sopen("example1.dat");
    if (res)
    {
        printf("The ephemeris is already opened\n");
        /* print the values of AU, EMRAT and GM_Mer */
        if (calceph_sgetconstant("AU", &AU))
            printf("AU=%23.16E\n", AU);

        if (calceph_sgetconstant("EMRAT", &EMRAT))
            printf("EMRAT=%23.16E\n", EMRAT);

        if (calceph_sgetconstant("GM_Mer", &GM_Mer))
            printf("GM_Mer=%23.16E\n", GM_Mer);

        /* compute and print the coordinates */
        /* the geocentric moon coordinates */
        calceph_scompute(jd0, dt, 10, 3, PV);
        printcoord(PV,"geocentric coordinates of the Moon");

        /* the value TT-TDB */
        calceph_scompute(jd0, dt, 16, 0, PV);
        printf("TT-TDB = %23.16E\n", PV[0]);

        /* the heliocentric coordinates of Mars */
        calceph_scompute(jd0, dt, 4, 11, PV);
        printcoord(PV,"heliocentric coordinates of Mars");

        /* close the ephemeris file */
        calceph_sclose();
        printf("The ephemeris is already closed\n");
    }
    else
    {
        printf("The ephemeris can't be opened\n");
    }
    return res;
}

```

### 5.4.3 Functions

#### 5.4.3.1 calceph\_sopen

```

int calceph_sopen ( const char *filename ) [C]
function calceph_sopen (filename) BIND(C) [Fortran 2003]
    CHARACTER(len=1,kind=C_CHAR), intent(in) :: filename
    INTEGER(C_INT) :: calceph_sopen

function f90calceph_sopen (filename) [Fortran 77/90/95]
    CHARACTER(len=*), intent(in) :: filename
    INTEGER :: f90calceph_sopen

```

This function opens the file whose pathname is the string pointed to by *filename*, reads the two header blocks of this file and associates an ephemeris descriptor to an internal variable. This file must be a binary ephemeris file.

The function `calceph_sclose` must be called to free allocated memory by this function.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

The following example opens the ephemeris file `example1.dat`

```

int res;
res = calceph_sopen("example1.dat");
if (res)
{
    /*
     ... computation ...
    */
    /* close the file */
    calceph_sclose();
}

```

#### 5.4.3.2 calceph\_scompute

```

int calceph_scompute ( double JD0, double time, int target, int center, [C]
    double PV[6] )
function calceph_scompute (JD0, time, target, center, PV) [Fortran 2003]
    BIND(C)
    REAL(C_DOUBLE), VALUE, intent(in) :: JD0
    REAL(C_DOUBLE), VALUE, intent(in) :: time
    INTEGER(C_INT), VALUE, intent(in) :: target
    INTEGER(C_INT), VALUE, intent(in) :: center
    REAL(C_DOUBLE), intent(out) :: PV(6)
    INTEGER(C_INT) :: calceph_scompute

```

```

function f90calceph_scompute (JD0, time, target,           [Fortran 77/90/95]
                             center, PV )
    REAL(8), intent(in) :: JD0
    REAL(8), intent(in) :: time
    INTEGER, intent(in) :: target
    INTEGER, intent(in) :: center
    REAL(8), intent(out) :: PV(6)
    INTEGER :: f90calceph_scompute

```

This function reads, if needed, and interpolates a single object, usually the position and velocity of one body (*target*) relative to another (*center*), from the binary ephemeris file, previously opened with the function `calceph_sopen`, for the time  $JD0+time$  and stores the results to *PV*.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

The arguments are :

<i>JD0</i>	Integer part of the Julian Date.
<i>time</i>	Fraction part of the Julian Date.
<i>target</i>	The body or reference point whose coordinates are required (see the list, below).
<i>center</i>	The origin of the coordinate system (see the list, below). If <i>target</i> is 15 or 16 (libration or TT-TDB), <i>center</i> must be '0'.
<i>PV</i>	An array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot). The position is expressed in Astronomical Unit (au) and the velocity is expressed in Astronomical Unit per day (au/day). If the target is <i>TT-TDB</i> , only the first element of this array will get the result. The time scale transformation TT-TDB is expressed in seconds.

To get the best precision for the interpolation, the time is splitted in two floating-point numbers. The argument *JD0* should be an integer and *time* should be a fraction of the day. But you may call this function with *time*=0 and *JD0*, the desired time, if you don't take care about precision.

The possible values for *target* and *center* are :

value	meaning
1	Mercury
2	Venus
3	Earth
4	Mars
5	Jupiter
6	Saturn
7	Uranus
8	Neptune
9	Pluto

```

10             Moon
11             Sun
12             Solar Sytem barycenter
13             Earth-moon barycenter
15             Librations
16             TT-TDB

```

These accepted values by this function are the same as the value for the JPL function PLEPH, except for the value TT-TDB.

The following example prints the heliocentric coordinates of Mars at time=2451624.5 and at 2451624.9

```

int res;
int j;
double jd0=2451624;
double dt1=0.5E0;
double dt2=0.9E0;

double PV[6];
/* open the ephemeris file */
res = calceph_sopen("example1.dat");
if (res)
{
    /* the heliocentric coordinates of Mars */
    calceph_scompute(jd0, dt1, 4, 11, PV);
    for(j=0; j<6; j++) printf("%23.16E\n", PV[j]);

    calceph_scompute(jd0, dt2, 4, 11, PV);
    for(j=0; j<6; j++) printf("%23.16E\n", PV[j]);

    /* close the ephemeris file */
    calceph_sclose();
}

```

### 5.4.3.3 calceph\_sgetconstant

```
int calceph_sgetconstant ( const char* name, double *value ) [C]
```

```
function calceph_sgetconstant (name, value ) BIND(C) [Fortran 2003]
    CHARACTER(len=1,kind=C_CHAR), intent(in) :: name
    REAL(C_DOUBLE), intent(out) :: value
    INTEGER(C_INT) :: calceph_sgetconstant
```

```
function f90calceph_sgetconstant (name, value ) [Fortran 77/90/95]
    CHARACTER(len=*), intent(in) :: name
    REAL(8), intent(out) :: value
    INTEGER :: f90calceph_sgetconstant
```

This function returns the value associated to the constant *name* in the header of the binary ephemeris file.

The function `calceph_sopen` must be previously called before. On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

The following example prints the value of the astronomical unit stored in the ephemeris file

```
int res;
double UA;
calceph_sopen("example1.dat");
res = calceph_sgetconstant("UA",&UA);
if (res)
{
    printf("astronomical unit=%23.16E\n", UA);
}
```

#### 5.4.3.4 calceph\_sgetconstantcount

```
int calceph_sgetconstantcount ( ) [C]
function calceph_sgetconstantcount ( ) BIND(C) [Fortran 2003]
    INTEGER(C_INT) :: calceph_sgetconstantcount
function f90calceph_sgetconstantcount ( ) [Fortran 77/90/95]
    INTEGER :: f90calceph_sgetconstantcount
```

This function returns the number of constants available in the header of the binary ephemeris file.

The function `calceph_sopen` must be previously called before. On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

The following example prints the number of available constants stored in the ephemeris file

```
int res, count;
calceph_sopen("example1.dat");
count = calceph_sgetconstantcount();
printf("number of constants : %d\n", count);
```

#### 5.4.3.5 calceph\_sgetconstantindex

```
int calceph_sgetconstantindex (int index, char [C]
    name[CALCEPH_MAX_CONSTANTNAME], double *value)
function calceph_sgetconstantindex (index, name, value) [Fortran 2003]
    BIND(C)
    INTEGER(C_INT), VALUE, intent(in) :: index
```

```

        CHARACTER(len=1,kind=C_CHAR),
        dimension(CALCEPH_MAX_CONSTANTNAME), intent(out) :: name
        REAL(C_DOUBLE), intent(out) :: value
        INTEGER(C_INT) :: calceph_sgetconstantindex

function f90calceph_sgetconstantindex (index, name,      [Fortran 77/90/95]
    value)
    INTEGER(INT), intent(in) :: index
    CHARACTER(len=CALCEPH_MAX_CONSTANTNAME), intent(out) ::
    name
    REAL(8), intent(out) :: value
    INTEGER :: f90calceph_sgetconstantindex

```

This function returns the name and its value of the constant available at the specified index in the header of the binary ephemeris file. The value of *index* must be between 1 and `calceph_sgetconstantcount()`.

The function `calceph_sopen` must be previously called before. On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

The following example displays the name of the constants, stored in the ephemeris file, and their values

```

        integer res
        integer j
        real(8) valueconstant
        character(len=CALCEPH_MAX_CONSTANTNAME) nameconstant

! open the ephemeris file
        res = calceph_sopen("example1.dat"//C_NULL_CHAR)
        if (res.eq.1) then

! print the list of the constants
        do j=1, calceph_sgetconstantcount()
            res = calceph_sgetconstantindex(j,nameconstant,      &
&                                     valueconstant)
            write (*,*) nameconstant,"=",valueconstant
        enddo

! close the ephemeris file
        call calceph_sclose

```

#### 5.4.3.6 calceph\_sclose

```
void calceph_sclose ( )
```

[C]

```
subroutine calceph_sclose ( ) [Fortran 2003]
subroutine f90calceph_sclose ( ) [Fortran 77/90/95]
```

This function closes the ephemeris data file and frees allocated memory by the function `calceph_sopen`.

## 5.5 Multiple file access functions

The following group of functions should be the preferred method to access to the library. They allow to access to multiple ephemeris files at the same time, even by multiple threads. When an error occurs, these functions execute error handlers according to the behavior defined by the function `calceph_seterrorhandler` (see [Section 5.6 \[Error functions\]](#), [page 26](#)).

### 5.5.1 Thread notes

If the standard I/O functions such as `fread` are not reentrant then the CALCEPH I/O functions using them will not be reentrant either.

It's not safe for two threads to call the functions with same object of type `t_calcephbin`. But it's safe for two threads to access simultaneously to the same binary ephemeris file with two different objects of type `t_calcephbin`. In this case, each thread must open the same file.

### 5.5.2 Usage

The following examples, that can be founded in the directory '`examples`' of the library sources, show the typical usage of this group of functions. The example in C language is '`cmultiple.c`'. The example in Fortran 2003 language is '`f2003multiple.f`'. The example in Fortran 77/90/95 language is '`f77multiple.f`'.

```

program f2003multiple
  USE, INTRINSIC :: ISO_C_BINDING
  use calceph
  implicit none
  integer res
  real(8) AU, EMRAT, GM_Mer
  real(8) jd0
  real(8) dt
  real(8) PV(6)
  TYPE(C_PTR) :: peph

  jd0 = 2451624
  dt = 0.5E0
! open the ephemeris file
  peph = calceph_open("example1.dat"//C_NULL_CHAR)
  if (C_ASSOCIATED(peph)) then
    write (*,*) "The ephemeris is already opened"
! print the values of AU, EMRAT and GM_Mer
    if (calceph_getconstant(peph, "AU"//C_NULL_CHAR,      &
&      AU).eq.1) then
      write (*,*) "AU=", AU
    endif
    if (calceph_getconstant(peph,"EMRAT"//C_NULL_CHAR,    &
&      EMRAT).eq.1) then
      write (*,*) "EMRAT=", EMRAT
    endif
    if (calceph_getconstant(peph,"GM_Mer"//C_NULL_CHAR,   &
&      GM_Mer).eq.1) then
      write (*,*) "GM_Mer=", GM_Mer
    endif

! compute and print the coordinates
! the geocentric moon coordinates
    res = calceph_compute(peph,jd0, dt, 10, 3, PV)
    call printcoord(PV,"geocentric coordinates of the Moon")
! the value TT-TDB
    if (calceph_compute(peph,jd0, dt, 16, 0, PV).eq.1) then
      write (*,*) "TT-TDB = ", PV(1)
    endif
! the heliocentric coordinates of Mars
    res = calceph_compute(peph,jd0, dt, 4, 11, PV)
    call printcoord(PV,"heliocentric coordinates of Mars")

! close the ephemeris file
    call calceph_close(peph)
    write (*,*) "The ephemeris is already closed"
  else
    write (*,*) "The ephemeris can't be opened"
  endif
stop
end

```



### 5.5.3 Functions

#### 5.5.3.1 calceph\_open

```

t_calcephbin* calceph_open ( const char *filename )           [C]
function calceph_open (filename) BIND(C)                     [Fortran 2003]
    CHARACTER(len=1,kind=C_CHAR), intent(in) :: filename
    TYPE(C_PTR) :: calceph_open

function f90calceph_open (eph, filename)                     [Fortran 77/90/95]
    CHARACTER(len=*), intent(in) :: filename
    INTEGER(8), intent(out) :: eph
    INTEGER :: f90calceph_open

```

This function opens the file whose pathname is the string pointed to by filename, reads the two header blocks of this file and returns an ephemeris descriptor associated to it. This file must be a binary ephemeris file.

The function `calceph_close` must be called to free allocated memory by this function.

On exit, it returns NULL (0 for the fortran 77/90/95 interface) if an error occurs, otherwise the return value is a non-NULL value.

The following example opens the ephemeris file `example1.dat` and `example2.dat`

```

t_calcephbin *peph1;
t_calcephbin *peph2;
peph1 = calceph_open("example1.dat");
peph2 = calceph_open("example2.dat");
if (peph1 && peph2)
{
    /*
    ... computation ...
    */
}
/* close the files */
if (peph1) calceph_close(peph1);
if (peph2) calceph_close(peph2);

```

#### 5.5.3.2 calceph\_compute

```

int calceph_compute (t_calcephbin* eph, double JD0, double time, int target, int center, double PV[6]) [C]

function calceph_compute (eph, JD0, time, target, center, PV [Fortran 2003]
) BIND(C)
    TYPE(C_PTR), VALUE, intent(in) :: eph
    REAL(C_DOUBLE), VALUE, intent(in) :: JD0

```

```

REAL(C_DOUBLE), VALUE, intent(in) :: time
INTEGER(C_INT), VALUE, intent(in) :: target
INTEGER(C_INT), VALUE, intent(in) :: center
REAL(C_DOUBLE), intent(out) :: PV(6)
INTEGER(C_INT) :: calceph_compute

function f90calceph_compute (eph, JD0, time, target,           [Fortran 77/90/95]
                           center, PV )
  INTEGER(8), intent(in) :: eph
  REAL(8), intent(in) :: JD0
  REAL(8), intent(in) :: time
  INTEGER, intent(in) :: target
  INTEGER, intent(in) :: center
  REAL(8), intent(out) :: PV(6)
  INTEGER :: f90calceph_compute

```

This function reads, if needed, in the ephemeris file associated to *eph* and interpolates a single object, usually the position and velocity of one body (*target*) relative to another (*center*), from the binary ephemeris file, previously opened with the function `calceph_sopen`, for the time  $JD0+time$  and stores the results to *PV*.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

The arguments are :

<i>JD0</i>	Integer part of the Julian Date.
<i>time</i>	Fraction part of the Julian Date.
<i>target</i>	The body or reference point whose coordinates are required (see the list, below).
<i>center</i>	The origin of the coordinate system (see the list, below). If <i>target</i> is 15 or 16 (libration or TT-TDB), <i>center</i> must be '0'.
<i>PV</i>	An array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot).  The position is expressed in Astronomical Unit (au) and the velocity is expressed in Astronomical Unit per day (au/day).  If the target is <i>TT-TDB</i> , only the first element of this array will get the result. The time scale transformation TT-TDB is expressed in seconds.

To get the best precision for the interpolation, the time is splitted in two floating-point numbers. The argument *JD0* should be an integer and *time* should be a fraction of the day. But you may call this function with *time*=0 and *JD0*, the desired time, if you don't take care about precision.

The possible values for *target* and *center* are :

value	meaning
1	Mercury
2	Venus

3	Earth
4	Mars
5	Jupiter
6	Saturn
7	Uranus
8	Neptune
9	Pluto
10	Moon
11	Sun
12	Solar Sytem barycenter
13	Earth-moon barycenter
15	Librations
16	TT-TDB

These accepted values by this function are the same as the value for the JPL function `PLEPH`, except for the value `TT-TDB`.

The following example prints the heliocentric coordinates of Mars at time=2451624.5 and at 2451624.9

```
int res;
int j;
double jd0=2451624;
double dt1=0.5E0;
double dt2=0.9E0;
t_calcephbin *peph;
double PV[6];

/* open the ephemeris file */
peph = calceph_open("example1.dat");
if (peph)
{
    /* the heliocentric coordinates of Mars */
    calceph_compute(peph, jd0, dt1, 4, 11, PV);
    for(j=0; j<6; j++) printf("%23.16E\n", PV[j]);

    calceph_compute(peph, jd0, dt2, 4, 11, PV);
    for(j=0; j<6; j++) printf("%23.16E\n", PV[j]);

    /* close the ephemeris file */
    calceph_close(peph);
}
```

### 5.5.3.3 calceph\_getconstant

```
int calceph_getconstant ( t_calcephbin* eph, const char* name, double value ) [C]
```

```

function calceph_getconstant (eph, name, value ) BIND(C)           [Fortran 2003]
    TYPE(C_PTR), VALUE, intent(in) :: eph
    CHARACTER(len=1,kind=C_CHAR), intent(in) :: name
    REAL(C_DOUBLE), intent(out) :: value
    INTEGER(C_INT) :: calceph_getconstant

function f90calceph_getconstant (eph, name, value )                [Fortran 77/90/95]
    INTEGER(8), intent(in) :: eph
    CHARACTER(len=*), intent(in) :: name
    REAL(8), intent(out) :: value
    INTEGER :: f90calceph_getconstant

```

This function returns the value associated to the constant *name* in the header of the binary ephemeris file associated to the descriptor *eph*.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

The following example prints the value of the astronomical unit stored in the ephemeris file

```

double AU;
t_calcephbin *peph;

/* open the ephemeris file */
peph = calceph_open("example1.dat");
if (peph)
{
    /* print the values of AU */
    if (calceph_getconstant(peph, "AU", &AU)) printf("AU=%23.16E\n", AU);

    /* close the ephemeris file */
    calceph_close(peph);
}

```

#### 5.5.3.4 calceph\_getconstantcount

```

int calceph_getconstantcount (t_calcephbin* eph )                 [C]
function calceph_getconstantcount (eph) BIND(C)                   [Fortran 2003]
    TYPE(C_PTR), VALUE, intent(in) :: eph
    INTEGER(C_INT) :: calceph_getconstantcount

function f90calceph_getconstantcount (eph)                         [Fortran 77/90/95]
    INTEGER(8), intent(in) :: eph
    INTEGER :: f90calceph_getconstantcount

```

This function returns the number of constants available in the header of the binary ephemeris file associated to the descriptor *eph*.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

The following example prints the number of available constants stored in the ephemeris file

```
int count;
t_calcephbin *peph;

/* open the ephemeris file */
peph = calceph_open("example1.dat");
if (peph)
{
    /* print the number of constants */
    count = calceph_getconstantcount(peph);
    printf("number of constants : %d\n", count);

    /* close the ephemeris file */
    calceph_close(peph);
}
```

### 5.5.3.5 calceph\_getconstantindex

```
int calceph_getconstantindex (t_calcephbin* eph, int index, char name[CALCEPH_MAX_CONSTANTNAME], double *value) [C]
```

```
function calceph_getconstantindex (eph, index, name, value) BIND(C) [Fortran 2003]
```

```
    TYPE(C_PTR), VALUE, intent(in) :: eph
    INTEGER(C_INT), VALUE, intent(in) :: index
    CHARACTER(len=1,kind=C_CHAR),
    dimension(CALCEPH_MAX_CONSTANTNAME), intent(out) :: name
    REAL(C_DOUBLE), intent(out) :: value
    INTEGER(C_INT) :: calceph_getconstantindex
```

```
function f90calceph_getconstantindex (eph, index, name, value) [Fortran 77/90/95]
    INTEGER(8), intent(in) :: eph
    INTEGER(INT), intent(in) :: index
    CHARACTER(len=CALCEPH_MAX_CONSTANTNAME), intent(out) :: name
    REAL(8), intent(out) :: value
    INTEGER :: f90calceph_getconstantindex
```

This function returns the name and its value of the constant available at the specified index in the header of the binary ephemeris file associated to the descriptor *eph*. The value of *index* must be between 1 and `calceph_getconstantcount(eph)`.

On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

The following example displays the name of the constants, stored in the ephemeris file, and their values

```

      USE, INTRINSIC :: ISO_C_BINDING
      use calceph
      implicit none
      integer res
      integer j
      real(8) valueconstant
      character(len=CALCEPH_MAX_CONSTANTNAME) nameconstant
      TYPE(C_PTR) :: peph

! open the ephemeris file
      peph = calceph_open("example1.dat"//C_NULL_CHAR)
      if (C_ASSOCIATED(peph)) then

! print the list of constants
          do j=1, calceph_getconstantcount(peph)
              res = calceph_getconstantindex(peph,j,nameconstant,
&                                     valueconstant)
              write (*,*) nameconstant,"=",valueconstant
          enddo

! close the ephemeris file
          call calceph_close(peph)
      endif

```

### 5.5.3.6 calceph\_close

`void calceph_close ( t_calcephbin* eph )` [C]

`function calceph_close (eph) BIND(C)` [Fortran 2003]  
`TYPE(C_PTR), VALUE, intent(in) :: eph`

`subroutine f90calceph_close (eph)` [Fortran 77/90/95]  
`INTEGER(8), intent(in) :: eph`

This function closes the access associated to the ephemeris descriptor *eph* and frees allocated memory for it.

## 5.6 Error functions

The following group of functions defines the behavior of the library when errors occur during the execution.

### 5.6.1 Usage

The following examples, that can be founded in the directory ‘examples’ of the library sources, show the typical usage of this group of functions. The example in C language is

‘cerror.c’. The example in Fortran 2003 language is ‘f2003error.f’. The example in Fortran 77/90/95 language is ‘f77error.f’.

The following example shows how to stop the execution on the error with the Fortran 2003 interface.

```
program f2003error
  USE, INTRINSIC :: ISO_C_BINDING
  use calceph
  implicit none
  integer res
  real(8) jd0
  real(8) dt
  real(8) PV(6)

! set the error handler to stop on error
  call calceph_seterrorhandler(2, C_NULL_FUNPTR)

! open the ephemeris file
  res = calceph_sopen("example1.dat"//C_NULL_CHAR)
  ...

  stop
end
```

The following example shows how to define a custom error handler function with the Fortran 2003 interface.

```

!/*-----*/
!/* custom error handler */
!/*-----*/

      subroutine myhandler(msg, msglen) BIND(C)
        USE, INTRINSIC :: ISO_C_BINDING
        implicit none
        character(kind=C_CHAR), dimension(msglen), intent(in) :: msg
        integer(C_INT), VALUE, intent(in) :: msglen
        write (*,*) "The calceph calls the function myhandler"
        write (*,*) "The message contains ",msglen," characters"
        write(*,*) "The error message is :"
        write(*,*) "-----"
        write(*,*) msg
        write(*,*) "-----"
        write(*,*) "The error handler returns"
      end

!/*-----*/
!/* main program */
!/*-----*/

      program f2003error
        USE, INTRINSIC :: ISO_C_BINDING
        use calceph
        implicit none
        integer res
        real(8) jd0
        real(8) dt
        real(8) PV(6)

        interface
          subroutine myhandler(msg, msglen) BIND(C)
            USE, INTRINSIC :: ISO_C_BINDING
            implicit none
            character(kind=C_CHAR), dimension(msglen), intent(in) :: msg
            integer(C_INT), VALUE, intent(in) :: msglen
          end subroutine
        end interface

! set the error handler to use my own callback
        call calceph_seterrorhandler(3, c_funloc(myhandler))

! open the ephemeris file
        res = calceph_sopen("example1.dat"//C_NULL_CHAR)

        ....

      stop
      end

```



### 5.6.2 calceph\_seterrorhandler

```
void calceph_seterrorhandler (int typehandler, void (*userfunc))(const [C]
    char*) )
```

```
subroutine calceph_seterrorhandler (typehandler, [Fortran 2003]
    userfunc ) BIND(C)
    TYPE(C_INT), VALUE, intent(in) :: typehandler
    TYPE(C_FUNPTR), VALUE, intent(in) :: userfunc
```

```
subroutine f90calceph_seterrorhandler (typehandler, [Fortran 77/90/95]
    userfunc )
    INTEGER, intent(in) :: typehandler
    EXTERNAL, intent(in) :: userfunc
```

This function defines the behavior of the library when an error occurs during the execution of the library's functions. This function should be (not mandatory) called before any other functions of the library. The behavior depends on the value of *typehandler*.

The possible values for *typehandler* are :

value	meaning
1	The library displays a message and continues the execution. The functions return an error code. This is the default behavior of the library.
2	The library displays a message and terminates the execution with a system call to the function <code>exit</code> .
3	The library calls the user function <i>userfunc</i> with the message.

If the function is called with 1 or 2 for *typehandler*, the parameter *userfunc* must be set to NULL in C, to C\_NULL\_FUNPTR in Fortran 2003, or to 0 in Fortran 77/90/95.

The function *userfunc* must be defined as

```
subroutine userfunc (msg, msglen ) BIND(C) [Fortran 2003]
    USE, INTRINSIC :: ISO_C_BINDING
    implicit none
    CHARACTER(kind=C_CHAR), dimension(msglen), intent(in) :: msg
    INTEGER(C_INT), VALUE, intent(in) :: msglen
```

```
subroutine userfunc (msg) BIND(C) [Fortran 77/90/95]
    implicit none
    CHARACTER(len=*), intent(in) :: msg
```

With Fortran 2003 interface, this function must have an explicit interface. With fortran 77/90/95 interface, this function must be declared as `EXTERNAL`.

## Appendix A Release notes

- Version 1.0.0  
Initial release.
- Version 1.0.1  
Supports the large binary ephemeris files (>2GB) on 32-bit operating systems.  
Fixes the documentation of the function `f90calceph_sopen`.  
Fixes an invalid open mode on Windows operating systems.  
Reports accurately the I/O errors.
- Version 1.0.2  
Fixes memory leaks in the fortran-90 interface.
- Version 1.0.3  
Supports the JPL ephemeris file DE423.
- Version 1.1.0  
Adds the function `calceph_seterrorhandler` for the custom error handlers.
- Version 1.1.1  
Fixes a compilation error in `util.h` and a warning with the sun studio compilers.
- Version 1.1.2  
Fixes a compilation warning with oracle studio compiler 12.  
Fixes a bug with gcc on solaris in 64 bit mode.  
Fixes the copyright statements.