
CALCEPH - Fortran 77/90/95 language

Release 3.5.4

M. Gastineau, J. Laskar, A. Fienga, H. Manche

Dec 01, 2023

CONTENTS

| | | |
|----------|--|-----------|
| 1 | Introduction | 3 |
| 2 | Installation | 5 |
| 2.1 | Unix-like system (Linux, Mac OS X, BSD, Cygwin, ...) | 5 |
| 2.1.1 | Quick instructions | 5 |
| 2.1.2 | Detailed instructions | 6 |
| 2.2 | Windows system | 7 |
| 2.2.1 | Using the Microsoft Visual C++ compiler | 7 |
| 2.2.2 | Using the MinGW | 9 |
| 3 | Library interface | 11 |
| 3.1 | A simple example program | 11 |
| 3.2 | Headers and Libraries | 11 |
| 3.2.1 | Compilation on a Unix-like system | 12 |
| 3.2.2 | Compilation on a Windows system | 12 |
| 3.3 | Types | 12 |
| 3.4 | Constants | 12 |
| 4 | Multiple file access functions | 15 |
| 4.1 | Time notes | 15 |
| 4.2 | Thread notes | 15 |
| 4.3 | Usage | 16 |
| 4.4 | Functions | 16 |
| 4.4.1 | f90calceph_open | 16 |
| 4.4.2 | f90calceph_open_array | 17 |
| 4.4.3 | f90calceph_prefetch | 18 |
| 4.4.4 | f90calceph_isthreadsafe | 18 |
| 4.4.5 | f90calceph_compute | 18 |
| 4.4.6 | f90calceph_compute_unit | 20 |
| 4.4.7 | f90calceph_orient_unit | 22 |
| 4.4.8 | f90calceph_rotangmom_unit | 23 |
| 4.4.9 | f90calceph_compute_order | 25 |
| 4.4.10 | f90calceph_orient_order | 27 |
| 4.4.11 | f90calceph_rotangmom_order | 29 |
| 4.4.12 | f90calceph_getconstant | 30 |
| 4.4.13 | f90calceph_getconstantsd | 31 |
| 4.4.14 | f90calceph_getconstantvd | 32 |
| 4.4.15 | f90calceph_getconstantss | 32 |
| 4.4.16 | f90calceph_getconstantvs | 33 |
| 4.4.17 | f90calceph_getconstantcount | 34 |

| | | |
|-----------|--|-----------|
| 4.4.18 | f90calceph_getconstantindex | 34 |
| 4.4.19 | f90calceph_getfileversion | 35 |
| 4.4.20 | f90calceph_gettimescale | 36 |
| 4.4.21 | f90calceph_gettimespan | 36 |
| 4.4.22 | f90calceph_getpositionrecordcount | 37 |
| 4.4.23 | f90calceph_getpositionrecordindex | 37 |
| 4.4.24 | f90calceph_getpositionrecordindex2 | 38 |
| 4.4.25 | f90calceph_getorientrecordcount | 39 |
| 4.4.26 | f90calceph_getorientrecordindex | 40 |
| 4.4.27 | f90calceph_getorientrecordindex2 | 40 |
| 4.4.28 | f90calceph_close | 41 |
| 5 | Single file access functions | 43 |
| 5.1 | Time notes | 43 |
| 5.2 | Thread notes | 44 |
| 5.3 | Usage | 44 |
| 5.4 | Functions | 45 |
| 5.4.1 | f90calceph_sopen | 45 |
| 5.4.2 | f90calceph_scompute | 46 |
| 5.4.3 | f90calceph_sgetconstant | 48 |
| 5.4.4 | f90calceph_sgetconstantcount | 48 |
| 5.4.5 | f90calceph_sgetconstantindex | 49 |
| 5.4.6 | f90calceph_sgetfileversion | 49 |
| 5.4.7 | f90calceph_sgettimescale | 50 |
| 5.4.8 | f90calceph_sgettimespan | 50 |
| 5.4.9 | f90calceph_sclose | 51 |
| 6 | Error functions | 53 |
| 6.1 | Usage | 53 |
| 6.2 | f90calceph_seterrorhandler | 54 |
| 7 | Miscellaneous functions | 57 |
| 7.1 | f90calceph_getmaxsupportedorder | 57 |
| 7.2 | f90calceph_getversion_str | 57 |
| 8 | NAIF identification numbers | 59 |
| 8.1 | Sun and planetary barycenters | 59 |
| 8.2 | Coordinate Time ephemerides | 59 |
| 8.3 | Planet centers and satellites | 60 |
| 8.4 | Comets | 63 |
| 9 | Release notes | 67 |
| 10 | Reporting bugs | 73 |
| 11 | CALCEPH Library Copying conditions | 75 |
| | Index | 77 |

This manual documents how to install and use the CALCEPH Library using the Fortran 77/90/95 interface.

Authors : M. Gastineau, J. Laskar, A. Fienga, H. Manche

INTRODUCTION

The CALCEPH Library is designed to access the binary planetary ephemeris files, such INPOPxx and JPL DExxx ephemeris files, (called 'original JPL binary' or 'INPOP 2.0 or 3.0 binary' ephemeris files in the next sections) and the SPICE kernel files (called 'SPICE' ephemeris files in the next sections). At the moment, supported SPICE files are :

- text Planetary Constants Kernel (KPL/PCK) files
- binary PCK (DAF/PCK) files.
- binary SPK (DAF/SPK) files containing segments of type 1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 20, 21, 102, 103 and 120.
- meta kernel (KPL/MK) files.
- frame kernel (KPL/FK) files. Only a basic support is provided.

This library provides a C interface and, optionally, the Fortran 77 or 2003, Python and Octave/Matlab interfaces, to be called by the application.

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

- Multiple file access functions

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

- 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.

This library could access to the following ephemeris

- INPOP06 or later
- DE200
- DE403 or later
- EPM2011 or later

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 original JPL 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>

The 'INPOP 2.0 binary' file format for planetary ephemeris files is described in the paper :

- M. Gastineau, J. Laskar, A. Fienga, H. Manche : 2012, INPOP binary ephemeris file format - version 2.0
http://www.imcce.fr/inpop/inpop_file_format_2_0.pdf

The 'INPOP 3.0 binary' file format for planetary ephemeris files is described in the paper :

- M. Gastineau, J. Laskar, A. Fienga, H. Manche : 2017, INPOP binary ephemeris file format - version 3.0
http://www.imcce.fr/inpop/inpop_file_format_3_0.pdf

INSTALLATION

The following section describes the installation of the **C** and **Fortran** interface of the library. If you want to install the interface for another programming language, you have to follow the instructions of the manual of that language.

2.1 Unix-like system (Linux, Mac OS X, BSD, Cygwin, ...)

2.1.1 Quick instructions

Here are the quick steps needed to install the library on Unix systems. In the following instructions, you must replace */home/mylogin/mydir* by the directory location where you want to install calceph.

If you use the `gnu gcc` and `gfortran` compilers, the steps are :

```
tar xzf calceph-3.5.4.tar.gz
cd calceph-3.5.4
./configure --disable-shared CC=gcc FC=gfortran --prefix=/home/mylogin/
→mydir
make check && make install
```

If you use the `Intel c++` and `fortran` compilers, the steps are :

```
tar xzf calceph-3.5.4.tar.gz
cd calceph-3.5.4
./configure --disable-shared CC=icc FC=ifort --prefix=/home/mylogin/mydir
make check && make install
```

If you use the `llvm clang` and `flang` compilers, the steps are :

```
tar xzf calceph-3.5.4.tar.gz
cd calceph-3.5.4
./configure --disable-shared CC=clang FC=flang --prefix=/home/mylogin/
→mydir
make check && make install
```

2.1.2 Detailed instructions

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 detailed steps needed to install the library on Unix systems:

- `tar xzf calceph-3.5.4.tar.gz`
- `cd calceph-3.5.4`
- `./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 `f90calceph_sopen()` and `f90calceph_scompute()`, ... and concurrent access to the function `f90calceph_compute()`, The default is *no*.
- `--disable-shared`
Disable shared library.
- `--disable-static`
Disable static library.
- `--help`
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 compiler flags could be changed using the variable `CFLAGS` for C compiler and `FCFLAGS` for the Fortran compiler.

Note: The option `--enable-python...` should not be used. They are reserved for the installation of the python interface.

- `make`

This compiles the CALCEPH Library in the working directory.

- `make check`

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

If you get error messages, please report them to inpop.imcce@obspm.fr (see *Reporting bugs*, for information on what to include in useful bug reports).

- `make install`

This will copy the files `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 documentations files to the directory **/usr/local/doc/calceph/** (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.

Other *make* Targets

There are some other useful make targets:

- `clean`

Delete all object files and archive files, but not the configuration files.

- `distclean`

Delete all files not included in the distribution.

- `installnodoc`

Same as `install`, except that the documentation is not installed.

- `uninstall`

Delete all files copied by `make install`.

2.2 Windows system

2.2.1 Using the Microsoft Visual C++ compiler

You need the Microsoft Visual C++ compiler, such as `cl.exe`, and the Universal CRT SDK or 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.

The "Universal CRT (C runtime) SDK" or a "Windows SDK" are now provided with the Microsoft Visual Studio. You should verify that "Universal CRT (C runtime) SDK" or a "Windows SDK" is selected in the "Visual Studio Installer".

If you use the C or Fortran interface, the steps are :

- Expand the file `calceph-3.5.4.tar.gz`
- Execute the command **cmd.exe** from the menu *Start / Execute...*

This will open a console window

- `cd dir\calceph-3.5.4`

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

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

- `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@obspm.fr (see *Reporting bugs*, 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.

- `nmake /f Makefile.vc 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 documentation files 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.

2.2.2 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.

The steps are :

- Expand the file `calceph-3.5.4.tar.gz`
- Execute the command *MinGW Shell* from the menu *Start*.

This will open a MinGW Shell console window.

- `cd dir\calceph-3.5.4`

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

- `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`

- `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.

- `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@obspm.fr (see *Reporting bugs* , 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*

- 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.

- `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 documentation files 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*

- 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.

Note: The option *...PYTHON...* should not be used. They are reserved for the installation of the python interface.

LIBRARY INTERFACE

3.1 A simple example program

The following example program shows the typical usage of the Fortran 77/90/95 interface.

Other examples using the Fortran 77/90/95 interface can be found in the directory *examples* of the library sources.

```
program F90example
  implicit none
  include 'f90calceph.h'
  integer res
  double precision AU
  integer*8 :: peph

  res = f90calceph_open(peph, "example1.dat")
  if (res.eq.1) then
    write (*,*) "The ephemeris is already opened"
    if (f90calceph_getconstant(peph, "AU", AU).eq.1) then
      write (*,*) "AU=", AU
    endif
    call f90calceph_close(peph)
    write (*,*) "The ephemeris is already closed"
  endif
stop
end
```

3.2 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'
```

3.2.1 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

```
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.

3.2.2 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.

3.3 Types

3.4 Constants

The following constants are defined in the file `f90calceph.h`.

CALCEPH_MAX_CONSTANTNAME [*integer*]

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.

CALCEPH_MAX_CONSTANTVALUE [*integer*]

This integer defines the maximum number of characters, including the trailing `'\0'`, that the value of a constant, available from the ephemeris file, could contain if the value is stored as a string of characters.

CALCEPH_VERSION_MAJOR [*integer*]

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

CALCEPH_VERSION_MINOR [*integer*]

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

CALCEPH_VERSION_PATCH [*integer*]

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

CALCEPH_VERSION_STRING [*character(len=*)*]

This string is the version of the library, which can be compared to the result of `calceph_getversion` to check at run time if the header file and library used match:

Note: Obtaining different strings is not necessarily an error, as in general, a program compiled with some old CALCEPH version can be dynamically linked with a newer CALCEPH library version (if allowed by the operating system).

CALCEPH_ASTEROID *[integer]*

This integer defines the offset value for the asteroids that must be used as target or center for the computation functions, such as `f90calceph_compute()`.

The following constants specify in which units are expressed the output of the computation functions, such as `f90calceph_compute_unit()` :

CALCEPH_UNIT_AU *[integer]*

This integer defines that the unit of the positions and velocities is expressed in astronomical unit.

CALCEPH_UNIT_KM *[integer]*

This integer defines that the unit of the positions and velocities is expressed in kilometer.

CALCEPH_UNIT_DAY *[integer]*

This integer defines that the unit of the velocities or the quantity TT-TDB or TCG-TCB is expressed in day (one day=86400 seconds).

CALCEPH_UNIT_SEC *[integer]*

This integer defines that the unit of the velocities or the quantity TT-TDB or TCG-TCB is expressed in second.

CALCEPH_UNIT_RAD *[integer]*

This integer defines that the unit of the angles is expressed in radian.

CALCEPH_OUTPUT_EULERANGLES *[integer]*

This integer defines that the output array contains the euler angles.

CALCEPH_OUTPUT_NUTATIONANGLES *[integer]*

This integer defines that the output array contains the nutation angles.

CALCEPH_USE_NAIFID *[integer]*

This integer defines that the NAIF identification numbers are used as target or center for the computation functions, such as `f90calceph_compute_unit()`.

The following constants specify the type of segments for the functions, such as `f90calceph_getmaxsupportedorder()` :

CALCEPH_SEGTYPE_ORIG_0 *[integer]*

This integer defines the type of segment for the original INPOP/JPL DE file format.

CALCEPH_SEGTYPE_SPK_1 *[integer]*

CALCEPH_SEGTYPE_SPK_2 *[integer]*

CALCEPH_SEGTYPE_SPK_3 *[integer]*

CALCEPH_SEGTYPE_SPK_5 *[integer]*

CALCEPH_SEGTYPE_SPK_8 *[integer]*

CALCEPH_SEGTYPE_SPK_9 *[integer]*

CALCEPH_SEGTYPE_SPK_12 *[integer]*

CALCEPH_SEGTYPE_SPK_13 *[integer]*

CALCEPH_SEGTYPE_SPK_14 *[integer]*
CALCEPH_SEGTYPE_SPK_17 *[integer]*
CALCEPH_SEGTYPE_SPK_18 *[integer]*
CALCEPH_SEGTYPE_SPK_19 *[integer]*
CALCEPH_SEGTYPE_SPK_20 *[integer]*
CALCEPH_SEGTYPE_SPK_21 *[integer]*
CALCEPH_SEGTYPE_SPK_102 *[integer]*
CALCEPH_SEGTYPE_SPK_103 *[integer]*
CALCEPH_SEGTYPE_SPK_120 *[integer]*

This integer defines the type of segments (1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 20, 21, 102, 103 and 120) for the SPICE Kernel files.

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 *f90calceph_seterrorhandler()*.

4.1 Time notes

The functions *f90calceph_compute()*, *f90calceph_compute_unit()*, *f90calceph_compute_order()*, *f90calceph_orient_unit()*, ... only accept a date expressed in the same timescale as the ephemeris files, which can be retrieved using the function *f90calceph_gettimescale()*. Ephemeris files are generally expressed using the timescale TDB. If a date, expressed in the TT (Terrestrial Time) timescale, is supplied to them, these functions will return an erroneous position of the order of several tens of meters for the planets. If a date, expressed in the Coordinated Universal Time (UTC), is supplied to them, these functions will return a very large erroneous position over several thousand kilometers for the planets.

4.2 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 safe for two threads to call the functions with the same handle of ephemeris object if and only if the function *f90calceph_isthreadsafe()* returns a non-zero value. A previous call to the function *f90calceph_prefetch()* is required for the function *f90calceph_isthreadsafe()* to return a non-zero value.

It's safe for two threads to access simultaneously to the same ephemeris file with two different objects. In this case, each thread must open the same file.

4.3 Usage

The following examples, that can be found in the directory *examples* of the library sources, show the typical usage of this group of functions.

The example in Fortran 77/90/95 language is `f77multiple.f`.

4.4 Functions

4.4.1 f90calceph_open

function `f90calceph_open` (*eph*, *filename*)

Parameters

- **filename** [*CHARACTER(len=*)*, *intent(in)*] :: pathname of the file.
- **eph** [*INTEGER(8)*, *intent(out)*] :: ephemeris descriptor

Return `f90calceph_open` [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

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 compliant to the format specified by the 'original JPL binary', 'INPOP 2.0 binary' or 'SPICE' ephemeris file. At the moment, supported SPICE files are the following :

- text Planetary Constants Kernel (KPL/PCK) files
- binary PCK (DAF/PCK) files.
- binary SPK (DAF/SPK) files containing segments of type 1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 20, 21, 102, 103 and 120.
- meta kernel (KPL/MK) files.
- frame kernel (KPL/FK) files. Only a basic support is provided.

Just after the call of `f90calceph_open()`, the function `f90calceph_prefetch()` should be called to accelerate future computations.

The function `f90calceph_close()` must be called to free allocated memory by this function.

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

```
include 'f90calceph.h'
integer res

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then

    ! ... computation ...

endif
call f90calceph_close(peph)
```

4.4.2 f90calceph_open_array

function `f90calceph_open_array` (*eph, n, array_filename, len_filename*)

Parameters

- **eph** [*INTEGER*(8), *intent(out)*] :: ephemeris descriptor
- **n** [*INTEGER*, *intent(in)*] :: number of files.
- **array_filename** [*CHARACTER*(len=*), *dimension*(*), *intent(in)*] :: array of pathname of the files
- **len_filename** [*INTEGER*, *intent(in)*] :: number of characters of each file's name.

Return `f90calceph_open_array` [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function opens *n* files whose pathnames are the string pointed to by *array_filename*, reads the header blocks of these files and returns an ephemeris descriptor associated to them.

These files must have the same type (e.g., all files are SPICE files or original JPL files). This file must be compliant to the format specified by the 'original JPL binary', 'INPOP 2.0 or 3.0 binary' or 'SPICE' ephemeris file. At the moment, supported SPICE files are the following :

- text Planetary Constants Kernel (KPL/PCK) files
- binary PCK (DAF/PCK) files.
- binary SPK (DAF/SPK) files containing segments of type 1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 20, 21, 102, 103 and 120.
- meta kernel (KPL/MK) files.
- frame kernel (KPL/FK) files. Only a basic support is provided.

Just after the call of `f90calceph_open_array()`, the function `f90calceph_prefetch()` should be called to accelerate future computations.

The function `f90calceph_close()` must be called to free allocated memory by this function.

The following example opens the ephemeris file `example1.bsp` and `example1.tpc`

```
integer*8 peph
integer res
character*80 filear(2)

filear(1) = "example1.bsp"
filear(2) = "example1.tpc"
res = f90calceph_open_array(peph, 2, filear, 80)
if (res.eq.1) then
  res = f90calceph_prefetch(peph)
  ! ... computation ...
  call calceph_close(peph)
endif
```

4.4.3 f90calceph_prefetch

function f90calceph_prefetch(*eph*)

Parameters *eph* [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor

Return f90calceph_prefetch [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function prefetches to the main memory all files associated to the ephemeris descriptor *eph*. This prefetching operation will accelerate the further computations performed with *f90calceph_compute()*, *f90calceph_compute_unit()*, *f90calceph_compute_order()*, *f90calceph_orient_unit()*, ...

It requires that the file is smaller than the main memory. If multiple threads (e.g. threads of openMP or Posix Pthreads) prefetch the data for the same ephemeris file, the used memory will remain the same as if the prefetch operation was done by a single thread if and if the endianness of the file is the same as the computer and if the operating system, such as Linux, MacOS X other unix, supports the function mmap.

4.4.4 f90calceph_isthreadsafe

function f90calceph_isthreadsafe(*eph*)

Parameters *eph* [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor

Return f90calceph_isthreadsafe [*INTEGER*] :: returns 1 if multiple threads can access the same ephemeris descriptor, otherwise 0.

This function returns 1 if multiple threads can access the same ephemeris descriptor, otherwise 0.

A previous call to the function *f90calceph_prefetch()* is required, and the library should be compiled with **--enable-thread=yes** on Unix-like operating system, for the function *f90calceph_isthreadsafe()* to return a non-zero value. If the file is not encoded with the same endian as the current hardware, then function may return 0.

If this function returns 1, several threads may use the same ephemeris descriptor for the computational functions *f90calceph_compute()*, It allows to use the same object for parallel loops.

4.4.5 f90calceph_compute

function f90calceph_compute(*eph*, *JD0*, *time*, *target*, *center*, *PV*)

Parameters

- **eph** [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor
- **JD0** [*REAL*(8), *intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL*(8), *intent(in)*] :: Fraction part of the Julian date (TDB or TCB)
- **target** [*INTEGER*, *intent(in)*] :: The body or reference point whose coordinates are required (see the list, below).
- **center** [*INTEGER*, *intent(in)*] :: The origin of the coordinate system (see the list, below). If *target* is 14, 15, 16 or 17 (nutration, libration, TT-TDB or TCG-TCB), *center* must be 0.
- **PV** (6) [*REAL*(8), *intent(out)*] :: Depending on the target value, an array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot), or a time scale transformation value, or the angles of the librations of the Moon and their derivatives, or the nutration angles and their derivatives.

Return f90calceph_compute [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

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*) for the time $JD0+time$ and stores the results to *PV*. The ephemeris file associated to *eph* must have been previously opened with the function `f90calceph_open()`.

The returned array *PV* has the following properties

- If the target is *TT-TDB*, only the first element of this array will get the result. The time scale transformation TT-TDB is expressed in seconds.
- If the target is *TCG-TCB*, only the first element of this array will get the result. The time scale transformation TCG-TCB is expressed in seconds.
- If the target is *Librations*, the array contains the angles of the librations of the Moon and their derivatives. The angles of the librations of the Moon are expressed in radians and their derivatives are expressed in radians per day.
- If the target is *Nutations*, the array contains the nutation angles and their derivatives. The nutation angles are expressed in radians and their derivatives are expressed in radians per day.
- Otherwise the returned values is the cartesian position (x,y,z), expressed in Astronomical Unit (au), and the velocity (xdot, ydot, zdot), expressed in Astronomical Unit per day (au/day).

The date (JD0, time) should be expressed in the same timescale as the ephemeris files, which can be retrieved using the function `f90calceph_gettimescale()`. To get the best numerical 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 numerical precision.

Warning: If a date, expressed in the Coordinated Universal Time (UTC), is supplied to this function, a very large erroneous position will be returned.

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

| value | meaning |
|------------------------------------|------------------------|
| 1 | Mercury Barycenter |
| 2 | Venus Barycenter |
| 3 | Earth |
| 4 | Mars Barycenter |
| 5 | Jupiter Barycenter |
| 6 | Saturn Barycenter |
| 7 | Uranus Barycenter |
| 8 | Neptune Barycenter |
| 9 | Pluto Barycenter |
| 10 | Moon |
| 11 | Sun |
| 12 | Solar Sytem barycenter |
| 13 | Earth-moon barycenter |
| 14 | Nutation angles |
| 15 | Librations |
| 16 | TT-TDB |
| 17 | TCG-TCB |
| asteroid number + CALCEPH_asteroid | asteroid |

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

For example, the value "CALCEPH_asteroid+4" for target or center specifies the asteroid Vesta.

The following example prints the heliocentric coordinates of Mars at time=2442457.5 and at 2442457.9

```
integer res
double precision jd0
double precision dt1, dt2
double precision PV(6)

jd0 = 2442457
dt1 = 0.5D0
dt2 = 0.9D0
res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then

  res = f90calceph_compute(peph, jd0, dt1, 4, 11, PV)
  write(*,*) PV

  res = f90calceph_compute(peph, jd0, dt2, 4, 11, PV)
  write(*,*) PV

  call f90calceph_close(peph)
endif
```

4.4.6 f90calceph_compute_unit

function f90calceph_compute_unit (eph, JD0, time, target, center, unit, PV)

Parameters

- **eph** [INTEGER(8), intent(in)] :: ephemeris descriptor
- **JD0** [REAL(8), intent(in)] :: Integer part of the Julian date (TDB or TCB)
- **time** [REAL(8), intent(in)] :: Fraction part of the Julian date (TDB or TCB)
- **target** [INTEGER, intent(in)] :: The body or reference point whose coordinates are required. The numbering system depends on the parameter unit.
- **center** [INTEGER, intent(in)] :: The origin of the coordinate system. The numbering system depends on the parameter unit.
- **unit** [INTEGER, intent(in)] ::

The units of PV.

This integer is a sum of some unit constants (CALCEPH_UNIT_???) and/or the constant CALCEPH_USE_NAIFID.

If the unit contains CALCEPH_USE_NAIFID, the NAIF identification numbering system is used for the target and the center (NAIF identification numbers for the list).

If the unit doesnot contain CALCEPH_USE_NAIFID, the old number system is used for the target and the center (see the list in the function f90calceph_compute()).

- **PV** (6) [REAL(8), intent(out)] :: Depending on the target value, an array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot), or a time scale transformation value, or the angles of the librations of the Moon and their derivatives, or the nutation angles and their derivatives.

Return f90calceph_compute_unit [INTEGER] :: 0 if an error occurs, otherwise non-zero value.

This function is similar to the function `f90calceph_compute()`, except that the units of the output are specified.

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*) for the time *JD0+time* and stores the results to *PV*. The ephemeris file associated to *eph* must have been previously opened with the function `f90calceph_open()`. The output values are expressed in the units specified by *unit*.

This function checks the units if invalid combinations of units are given to the function.

The date (JD0, time) should be expressed in the same timescale as the ephemeris files, which can be retrieved using the function `f90calceph_gettimescale()`.

Warning: If a date, expressed in the Coordinated Universal Time (UTC), is supplied to this function, a very large erroneous position will be returned.

The returned array *PV* has the following properties

- If the target is the time scale transformation TT-TDB, only the first element of this array will get the result.
- If the target is the time scale transformation TCG-TCB, only the first element of this array will get the result.
- If the target is *Librations*, the array contains the angles of the librations of the Moon and their derivatives.
- If the target is *Nutations*, the array contains the nutation angles and their derivatives.
- Otherwise the returned value is the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot).

The values stored in the array *PV* are expressed in the following units

- The position and velocity are expressed in Astronomical Unit (au) if unit contains `CALCEPH_UNIT_AU`.
- The position and velocity are expressed in kilometers if unit contains `CALCEPH_UNIT_KM`.
- The velocity, TT-TDB, TCG-TCB, the derivatives of the angles of the nutation, or the derivatives of the librations of the Moon or are expressed in days if unit contains `CALCEPH_UNIT_DAY`.
- The velocity, TT-TDB, TCG-TCB, the derivatives of the angles of the nutation, or the derivatives of the librations of the Moon are expressed in seconds if unit contains `CALCEPH_UNIT_SEC`.
- The angles of the librations of the Moon or the nutation angles are expressed in radians if unit contains `CALCEPH_UNIT_RAD`.

For example, to get the position and velocities expressed in kilometers and kilometers/seconds, the unit must be set to `CALCEPH_UNIT_KM + CALCEPH_UNIT_SEC`.

The following example prints the heliocentric coordinates of Mars at time=2442457.5

```
integer*8 peph
integer res
double precision jd0
double precision dt1
double precision PV(6)

jd0 = 2442457
dt1 = 0.5D0
res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then

    ! the heliocentric coordinates of Mars in km and km/s
    res = f90calceph_compute_unit(peph,jd0, dt1, 4, 11,
&                                CALCEPH_USE_NAIFID+CALCEPH_UNIT_KM+CALCEPH_UNIT_SEC,
```

(continues on next page)

(continued from previous page)

```

&                                PV)
  write(*,*) PV

  ! compute same quantity as the previous call using NAIF ID
  res = f90calceph_compute_unit(peph,jd0, dt1,
&                                NAIFID_MARS_BARYCENTER, NAIFID_SUN,
&                                CALCEPH_USE_NAIFID+CALCEPH_UNIT_KM+CALCEPH_UNIT_SEC,
&                                PV)
  write(*,*) PV

  call f90calceph_close(peph)
endif

```

4.4.7 f90calceph_orient_unit

function f90calceph_orient_unit (*eph, JD0, time, target, unit, PV*)

Parameters

- **eph** [*INTEGER(8), intent(in)*] :: ephemeris descriptor
- **JD0** [*REAL(8), intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL(8), intent(in)*] :: Fraction part of the Julian date (TDB or TCB)
- **target** [*INTEGER, intent(in)*] :: The body whose orientations are requested. The numbering system depends on the parameter unit.
- **unit** [*INTEGER, intent(in)*] ::
The units of PV.
This integer is a sum of some unit constants (*CALCEPH_UNIT_???*) and/or the constant *CALCEPH_USE_NAIFID*.
If the unit contains *CALCEPH_USE_NAIFID*, the NAIF identification numbering system is used for the target (*NAIF identification numbers* for the list).
If the unit does not contain *CALCEPH_USE_NAIFID*, the old number system is used for the target (see the list in the function *f90calceph_compute()*).
- **PV** (6) [*REAL(8), intent(out)*] :: An array to receive the euler angles, or nutation angles, and their derivatives for the orientation of the body.

Return **f90calceph_orient_unit** [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function reads, if needed, in the ephemeris file associated to *eph* and interpolates the orientation of a single body (*target*) for the time *JD0+time* and stores the results to *PV*. The ephemeris file associated to *eph* must have been previously opened with the function *f90calceph_open()*. The output values are expressed in the units specified by *unit*.

The date (JD0, time) should be expressed in the same timescale as the ephemeris files, which can be retrieved using the function *f90calceph_gettimescale()*.

This function checks the units if invalid combinations of units are given to the function.

The returned array *PV* has the following properties

- If *unit* contains *CALCEPH_OUTPUT_NUTATIONANGLES*, the array contains the nutation angles and their derivatives for the orientation of the body. At the present moment, only the nutation for the earth are supported in the original DE files.

- If *unit* contains `CALCEPH_OUTPUT_EULERANGLES`, or doesnot contain `CALCEPH_OUTPUT_NUTATIONANGLES`, the array contains the euler angles and their derivatives for the orientation of the body.

The values stored in the array *PV* are expressed in the following units

- The derivatives of the angles are expressed in days if unit contains `CALCEPH_UNIT_DAY`.
- The derivatives of the angles are expressed in seconds if unit contains `CALCEPH_UNIT_SEC`.
- The angles and their derivatives are expressed in radians if unit contains `CALCEPH_UNIT_RAD`.

For example, to get the nutation angles of the Earth and their derivatives expressed in radian and radian/seconds using the NAIF identification numbering system, the target must be set to `NAIFID_EARTH` and the unit must be set to `CALCEPH_OUTPUT_NUTATIONANGLES + CALCEPH_UNIT_RAD + CALCEPH_UNIT_SEC`.

The following example prints the angles of libration of the Moon at time=2442457.5

```
integer*8 peph
integer res
double precision jd0
double precision dt1
double precision PV(6)

jd0 = 2442457
dt1 = 0.5D0
res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then

  res = f90calceph_orient_unit(peph, jd0, dt1, NAIFID_MOON,
&                                CALCEPH_USE_NAIFID+CALCEPH_UNIT_RAD+CALCEPH_UNIT_SEC,
&                                PV)
  write(*,*) PV

  call f90calceph_close(peph)
endif
```

4.4.8 f90calceph_rotangmom_unit

function `f90calceph_rotangmom_unit` (*eph, JD0, time, target, unit, PV*)

Parameters

- **eph** [*INTEGER(8), intent(in)*] :: ephemeris descriptor
- **JD0** [*REAL(8), intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL(8), intent(in)*] :: Fraction part of the Julian date (TDB or TCB)
- **target** [*INTEGER, intent(in)*] :: The body whose orientations are requested. The numbering system depends on the parameter unit.
- **unit** [*INTEGER, intent(in)*] ::

The units of PV.

This integer is a sum of some unit constants (`CALCEPH_UNIT_???`) and/or the constant `CALCEPH_USE_NAIFID`.

If the unit contains `CALCEPH_USE_NAIFID`, the NAIF identification numbering system is used for the target (*NAIF identification numbers* for the list).

If the unit does not contain `CALCEPH_USE_NAIFID`, the old number system is used for the target (see the list in the function `f90calceph_compute()`).

- **PV** (6) [REAL(8), intent(out) :: An array to receive the angular momentum due to its rotation, divided by the product of the mass and of the square of the radius, and the derivatives, of the body.

Return `f90calceph_rotangmom_unit [INTEGER]` :: 0 if an error occurs, otherwise non-zero value.

This function reads, if needed, in the ephemeris file associated to *eph* and interpolates the angular momentum vector due to the rotation of the body, divided by the product of the mass *m* and of the square of the radius *R*, of a single body (*target*) for the time *JD0+time* and stores the results to *PV*. The ephemeris file associated to *eph* must have been previously opened with the function `f90calceph_open()`. The angular momentum *L*, due to the rotation of the body, is defined as the product of the inertia matrix *I* by the angular velocity vector ω . So the returned value is $L/(mR^2) = (I\omega)/(mR^2)$

The date (JD0, time) should be expressed in the same timescale as the ephemeris files, which can be retrieved using the function `f90calceph_gettimescale()`.

The output values are expressed in the units specified by *unit*.

This function checks the units if invalid combinations of units are given to the function.

The values stored in the array *PV* are expressed in the following units

- The angular momentum and its derivative are expressed in days if unit contains `CALCEPH_UNIT_DAY`.
- The angular momentum and its derivative are expressed in seconds if unit contains `CALCEPH_UNIT_SEC`.

The following example prints the angular momentum, due to its rotation, for the Earth at time=2451419.5

```
integer*8 peph
integer res
double precision jd0
double precision dt1
double precision G(6)

jd0 = 2451419
dt1 = 0.5D0
res = f90calceph_open(peph, "example2_rotangmom.dat")
if (res.eq.1) then

    res = f90calceph_rotangmom_unit(peph, jd0, dt1, NAIFID_EARTH,
&                                     CALCEPH_USE_NAIFID+CALCEPH_UNIT_SEC,
&                                     G)
    write(*,*) G

    call f90calceph_close(peph)
endif
```

4.4.9 f90calceph_compute_order

function `f90calceph_compute_order` (*eph*, *JD0*, *time*, *target*, *center*, *unit*, *order*, *PVAJ*)

Parameters

- **eph** [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor
- **JD0** [*REAL*(8), *intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL*(8), *intent(in)*] :: Fraction part of the Julian date (TDB or TCB)
- **target** [*INTEGER*, *intent(in)*] :: The body or reference point whose coordinates are required. The numbering system depends on the parameter unit.
- **center** [*INTEGER*, *intent(in)*] :: The origin of the coordinate system. The numbering system depends on the parameter unit.
- **unit** [*INTEGER*, *intent(in)*] ::

The units of PVAJ.

This integer is a sum of some unit constants (`CALCEPH_UNIT_???`) and/or the constant `CALCEPH_USE_NAIFID`.

If the unit contains `CALCEPH_USE_NAIFID`, the NAIF identification numbering system is used for the target and the center (*NAIF identification numbers* for the list).

If the unit doesnot contain `CALCEPH_USE_NAIFID`, the old number system is used for the target and the center (see the list in the function `f90calceph_compute()`).

- **order** [*INTEGER*, *intent(in)*] :: The order of derivatives
 - = 0 , only the position is computed. The first three numbers of PVAJ are valid for the results.
 - = 1 , only the position and velocity are computed. The first six numbers of PVAJ are valid for the results.
 - = 2 , only the position, velocity and acceleration are computed. The first nine numbers of PVAJ are valid for the results.
 - = 3 , the position, velocity and acceleration and jerk are computed. The first twelve numbers of PVAJ are valid for the results.

If order equals to 1, the behavior of `f90calceph_compute_order()` is the same as `f90calceph_compute_unit()`.

- **PVAJ** (12) [*REAL*(8), *intent(out)*] :: Depending on the target value, an array to receive the cartesian position (x,y,z), the velocity (xdot, ydot, zdot), the acceleration and the jerk, or a time scale transformation value, or the angles of the librations of the Moon and their successive derivatives, or the nutation angles and their successive derivatives.

Return `f90calceph_compute_order` [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function is similar to the function `f90calceph_compute_unit()`, except that the order of the computed derivatives is specified.

This function reads, if needed, in the ephemeris file associated to *eph* and interpolates a single object, usually the position and their derivatives of one body (*target*) relative to another (*center*) for the time *JD0+time* and stores the results to *PVAJ*. The ephemeris file associated to *eph* must have been previously opened with the function `f90calceph_open()`. The order of the derivatives are specified by *order*. The output values are expressed in the units specified by *unit*.

The returned array *PVAJ* has the following properties

- If the target is the time scale transformation TT-TDB, only the first elements of each component will get the result.
- If the target is the time scale transformation *TCG-TCB*, only the first elements of each component will get the result.
- If the target is *Librations*, the array contains the angles of the librations of the Moon and their successive derivatives.
- If the target is *Nutations*, the array contains the nutation angles and their successive derivatives.
- Otherwise the returned value is the cartesian position (x,y,z), the velocity (xdot, ydot, zdot), the jerk and the acceleration.

The returned array *PVAJ* must be large enough to store the results.

- PVAJ[1:3] contain the position (x,y,z) and is always valid.
- PVAJ[4:6] contain the velocity (dx/dt,dy/dt,dz/dt) and is only valid if *order* is greater or equal to 1.
- PVAJ[7:9] contain the acceleration ($d^2x/dt^2, d^2y/dt^2, d^2z/dt^2$) and is only valid if *order* is greater or equal to 2.
- PVAJ[10:12] contain the jerk ($d^3x/dt^3, d^3y/dt^3, d^3z/dt^3$) and is only valid if *order* is equal to 3.

The date (JD0, time) should be expressed in the same timescale as the ephemeris files, which can be retrieved using the function `f90calceph_gettimescale()`.

Warning: If a date, expressed in the Coordinated Universal Time (UTC), is supplied to this function, a very large erroneous position will be returned.

The values stored in the array *PVAJ* are expressed in the following units

- The position, velocity, acceleration and jerk are expressed in Astronomical Unit (au) if unit contains `CALCEPH_UNIT_AU`.
- The position, velocity, acceleration and jerk are expressed in kilometers if unit contains `CALCEPH_UNIT_KM`.
- The velocity, acceleration, jerk, TT-TDB, TCG-TCB or the derivatives of the angles of the librations of the Moon are expressed in days if unit contains `CALCEPH_UNIT_DAY`.
- The velocity, acceleration, jerk, TT-TDB, TCG-TCB or the derivatives of the angles of the librations of the Moon are expressed in seconds if unit contains `CALCEPH_UNIT_SEC`.
- The angles of the librations of the Moon are expressed in radians if unit contains `CALCEPH_UNIT_RAD`.

For example, to get the positions, velocities, accelerations and jerks expressed in kilometers and kilometers/seconds, the unit must be set to `CALCEPH_UNIT_KM + CALCEPH_UNIT_SEC`.

This function checks the units if invalid combinations of units are given to the function.

The following example prints the heliocentric coordinates of Mars at time=2442457.5

```
integer*8 peph
integer res
double precision jd0
double precision dt1
double precision P(3), PVAJ(12)

jd0 = 2442457
dt1 = 0.5D0
res = f90calceph_open(peph, "example1.dat")
```

(continues on next page)

(continued from previous page)

```

if (res.eq.1) then

    ! compute only the heliocentric position of Mars in km
    res = f90calceph_compute_order(peph, jd0, dt1,
&                                NAIFID_MARS_BARYCENTER,
&                                NAIFID_SUN,
&                                CALCEPH_USE_NAIFID+CALCEPH_UNIT_KM+CALCEPH_UNIT_SEC,
&                                0, P);
    write(*,*) P

    ! compute positions, velocities, accelerations and jerks of Mars in km and seconds
    res = f90calceph_compute_order(peph, jd0, dt1,
&                                NAIFID_MARS_BARYCENTER,
&                                NAIFID_SUN,
&                                CALCEPH_USE_NAIFID+CALCEPH_UNIT_KM+CALCEPH_UNIT_SEC,
&                                3, PVAJ);
    write(*,*) PVAJ

    call f90calceph_close(peph)
endif

```

4.4.10 f90calceph_orient_order

function f90calceph_orient_order (eph, JD0, time, target, unit, order, PVAJ)

Parameters

- **eph** [INTEGER(8), intent(in)] :: ephemeris descriptor
- **JD0** [REAL(8), intent(in)] :: Integer part of the Julian date (TDB or TCB)
- **time** [REAL(8), intent(in)] :: Fraction part of the Julian date (TDB or TCB)
- **target** [INTEGER, intent(in)] :: The body whose orientations are requested. The numbering system depends on the parameter unit.
- **unit** [INTEGER, intent(in)] ::
 The units of PV.
 This integer is a sum of some unit constants (CALCEPH_UNIT_???) and/or the constant *CALCEPH_USE_NAIFID*.
 If the unit contains *CALCEPH_USE_NAIFID*, the NAIF identification numbering system is used for the target (*NAIF identification numbers* for the list).
 If the unit does not contain *CALCEPH_USE_NAIFID*, the old number system is used for the target (see the list in the function *f90calceph_compute()*).
- **order** [INTEGER, intent(in)] :: The order of derivatives.
 - = 0 , only the angles is computed. The first three numbers of PVAJ are valid for the results.
 - = 1 , only the angles and the first derivative are computed. The first six numbers of PVAJ are valid for the results.
 - = 2 , only the angles and the first and second derivatives are computed. The first nine numbers of PVAJ are valid for the results.
 - = 3 , the angles and the first, second and third derivatives are computed. The first twelve numbers of PVAJ are valid for the results.

If order equals to 1, the behavior of `f90calceph_orient_order()` is the same as `f90calceph_orient_unit()`.

- **PVAJ** (12) [REAL(8), intent(out) :: An array to receive the euler angles, or nutation angles, and their derivatives for the orientation of the body.

Return `f90calceph_orient_order [INTEGER]` :: 0 if an error occurs, otherwise non-zero value.

This function is similar to the function `f90calceph_orient_unit()`, except that the order of the computed derivatives is specified.

This function reads, if needed, in the ephemeris file associated to `eph` and interpolates the orientation of a single body (*target*) for the time `JD0+time` and stores the results to `PVAJ`. The order of the derivatives are specified by `order`. The ephemeris file associated to `eph` must have been previously opened with the function `f90calceph_open()`. The output values are expressed in the units specified by `unit`.

This function checks the units if invalid combinations of units are given to the function.

The returned array `PVAJ` has the following properties

- If `unit` contains `CALCEPH_OUTPUT_NUTATIONANGLES`, the array contains the nutation angles and their successive derivatives for the orientation of the body. At the present moment, only the nutation for the earth are supported in the original DE files.
- If `unit` contains `CALCEPH_OUTPUT_EULERANGLES`, or doesnot contain `CALCEPH_OUTPUT_NUTATIONANGLES`, the array contains the euler angles and their successive derivatives for the orientation of the body.

The returned array `PVAJ` must be large enough to store the results.

- `PVAJ[1:3]` contain the angles and is always valid.
- `PVAJ[4:6]` contain the first derivative and is only valid if `order` is greater or equal to 1.
- `PVAJ[7:9]` contain the second derivative and is only valid if `order` is greater or equal to 2.
- `PVAJ[10:12]` contain the third derivative and is only valid if `order` is equal to 3.

The values stored in the array `PVAJ` are expressed in the following units

- The derivatives of the angles are expressed in days if unit contains `CALCEPH_UNIT_DAY`.
- The derivatives of the angles are expressed in seconds if unit contains `CALCEPH_UNIT_SEC`.
- The angles and their derivatives are expressed in radians if unit contains `CALCEPH_UNIT_RAD`.

The date (JD0, time) should be expressed in the same timescale as the ephemeris files, which can be retrieved using the function `f90calceph_gettimescale()`.

The following example prints only the angles of libration of the Moon at time=2442457.5

```
integer*8 peph
integer res
double precision jd0
double precision dt1
double precision P(3)

jd0 = 2442457
dt1 = 0.5D0
res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then

    res = f90calceph_orient_order(peph, jd0, dt1, NAIFID_MOON,
&                                CALCEPH_USE_NAIFID+CALCEPH_UNIT_RAD+CALCEPH_UNIT_SEC,
```

(continues on next page)

(continued from previous page)

```

&                                0, P)
  write(*,*) P
  call f90calceph_close(peph)
endif

```

4.4.11 f90calceph_rotangmom_order

function `f90calceph_rotangmom_order` (*eph*, *JD0*, *time*, *target*, *unit*, *order*, *PVAJ*)

Parameters

- **eph** [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor
- **JD0** [*REAL*(8), *intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL*(8), *intent(in)*] :: Fraction part of the Julian date (TDB or TCB)
- **target** [*INTEGER*, *intent(in)*] :: The body whose orientations are requested. The numbering system depends on the parameter unit.
- **unit** [*INTEGER*, *intent(in)*] ::

The units of PV.

This integer is a sum of some unit constants (`CALCEPH_UNIT_???`) and/or the constant `CALCEPH_USE_NAIFID`.

If the unit contains `CALCEPH_USE_NAIFID`, the NAIF identification numbering system is used for the target (*NAIF identification numbers* for the list).

If the unit does not contain `CALCEPH_USE_NAIFID`, the old number system is used for the target (see the list in the function `f90calceph_compute()`).

- **order** [*INTEGER*, *intent(in)*] :: The order of derivatives.
 - = 0 , only the angular momentum is computed. The first three numbers of PVAJ are valid for the results.
 - = 1 , only the angular momentum and the first derivative are computed. The first six numbers of PVAJ are valid for the results.
 - = 2 , only the angular momentum and the first and second derivatives are computed. The first nine numbers of PVAJ are valid for the results.
 - = 3 , the angular momentum and the first, second and third derivatives are computed. The first twelve numbers of PVAJ are valid for the results.

If order equals to 1, the behavior of `f90calceph_rotangmom_order()` is the same as `f90calceph_rotangmom_unit()`.

- **PVAJ** (12) [*REAL*(8), *intent(out)*] :: An array to receive the angular momentum due to its rotation, divided by the product of the mass and of the square of the radius, and their different order of the derivatives, of the body.

Return `f90calceph_rotangmom_order` [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function is similar to the function `f90calceph_orient_unit()`, except that the order of the computed derivatives is specified.

This function reads, if needed, in the ephemeris file associated to *eph* and interpolates the angular momentum vector due to the rotation of the body, divided by the product of the mass *m* and of the square of the radius *R*, of a single

body (*target*) for the time $JD0+time$ and stores the results to *PVAJ*. The angular momentum L , due to the rotation of the body, is defined as the product of the inertia matrix I by the angular velocity vector ω . So the returned value is $L/(mR^2) = (I\omega)/(mR^2)$. The order of the derivatives are specified by *order*. The ephemeris file associated to *eph* must have been previously opened with the function `f90calceph_open()`. The output values are expressed in the units specified by *unit*.

This function checks the units if invalid combinations of units are given to the function.

The returned array *PVAJ* must be large enough to store the results.

- *PVAJ*[1:3] contain the angular momentum and is always valid.
- *PVAJ*[4:6] contain the first derivative and is only valid if *order* is greater or equal to 1.
- *PVAJ*[7:9] contain the second derivative and is only valid if *order* is greater or equal to 2.
- *PVAJ*[10:12] contain the third derivative and is only valid if *order* is equal to 3.

The values stored in the array *PVAJ* are expressed in the following units

- The angular momentum and its derivatives are expressed in days if unit contains `CALCEPH_UNIT_DAY`.
- The angular momentum and its derivatives are expressed in seconds if unit contains `CALCEPH_UNIT_SEC`.

The date (JD0, time) should be expressed in the same timescale as the ephemeris files, which can be retrieved using the function `f90calceph_gettimescale()`.

The following example prints only the angular momentum, due to its rotation, of the Earth at time=2451419.5

```
integer*8 peph
integer res
double precision jd0
double precision dt1
double precision G(3)

jd0 = 2451419
dt1 = 0.5D0
res = f90calceph_open(peph, "example2_rotangmom.dat")
if (res.eq.1) then

    res = f90calceph_rotangmom_order(peph, jd0, dt1, NAIFID_EARTH,
&                                     CALCEPH_USE_NAIFID+CALCEPH_UNIT_SEC,
&                                     G)
    write(*,*) G

    call f90calceph_close(peph)
endif
```

4.4.12 f90calceph_getconstant

function `f90calceph_getconstant` (*eph, name, value*)

Parameters

- **eph** [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor
- **name** [*CHARACTER*(len=*), *intent(in)*] :: name of the constant
- **value** [*REAL*(8), *intent(out)*] :: first value of the constant

Return `f90calceph_getconstant` [*INTEGER*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

This function returns the value associated to the constant *name* in the header of the ephemeris file associated to *eph*. Only the first value is returned if multiple values are associated to a constant, such as a list of values.

This function is the same function as `f90calceph_getconstantsd()`.

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

```
integer*8 peph
integer res
double precision AU

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  ! print the value of AU
  if (f90calceph_getconstant(peph, "AU", AU).eq.1) then
    write (*,*) "AU=", AU
  endif
endif

call f90calceph_close(peph)
endif
```

4.4.13 f90calceph_getconstantsd

function `f90calceph_getconstantsd` (*eph*, *name*, *value*)

Parameters

- **eph** [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor
- **name** [*CHARACTER*(len=*), *intent(in)*] :: name of the constant
- **value** [*REAL*(8), *intent(out)*] :: first value of the constant

Return `f90calceph_getconstantsd` [*INTEGER*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

This function returns, as a floating-point number, the value associated to the constant *name* in the header of the ephemeris file associated to *eph*. Only the first value is returned if multiple values are associated to a constant, such as a list of values. The value must be a floating-point or integer number, otherwise an error is reported.

This function is the same function as `f90calceph_getconstant()`.

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

```
integer*8 peph
integer res
double precision AU

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  ! print the value of AU
  if (f90calceph_getconstantsd(peph, "AU", AU).eq.1) then
    write (*,*) "AU=", AU
  endif
endif

call f90calceph_close(peph)
endif
```

4.4.14 f90calceph_getconstantvd

function f90calceph_getconstantvd (*eph*, *name*, *arrayvalue*, *nvalue*)

Parameters

- **eph** [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor
- **name** [*CHARACTER*(*len*=*), *intent(in)*] :: name of the constant
- **value** [*REAL*(8), *dimension*(1:*nvalue*), *intent(inout)*] :: array of values for the constant
- **nvalue** [*INTEGER*, *intent(in)*] :: number of elements of the array

Return f90calceph_getconstantvd [*INTEGER*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

This function stores, to the array *arrayvalue* as floating-point numbers, the *nvalue* first values associated to the constant *name* in the header of the ephemeris file associated to *eph*. The integer value returned by the function is equal to the number of valid entries in the *arrayvalue* if *nvalue* is greater or equal to that integer value..

The required value *nvalue* to store all values can be determined with the previous call to *f90calceph_getconstantsd*.

The values must be floating-point or integer numbers, otherwise an error is reported.

The following example prints the body radii of the earth stored in the ephemeris file

```
integer*8 peph
integer res, nvalue
double precision, allocatable :: radii
double precision svalue

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
    ! get the number of values
    nvalue = calceph_getconstantsd(peph, "BODY399_RADII", svalue)
    ! fill the array
    allocate(radii(1:nvalue))
    res = calceph_getconstantvd(peph, "BODY399_RADII", radii, nvalue)
    write(*,*) radii

    call f90calceph_close(peph)
endif
```

4.4.15 f90calceph_getconstantss

function f90calceph_getconstantss (*eph*, *name*, *value*)

Parameters

- **eph** [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor
- **name** [*CHARACTER*(*len*=*), *intent(in)*] :: name of the constant
- **value** [*CHARACTER*(*len*=*CALCEPH_MAX_CONSTANTNAME*), *intent(out)*] :: first value of the constant

Return f90calceph_getconstantss [*INTEGER*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

This function returns, as a string of character, the value associated to the constant *name* in the header of the ephemeris file associated to *eph*. Only the first value is returned if multiple values are associated to a constant, such as a list of values. The value must be a string, otherwise an error is reported.

Trailing blanks are added to each value.

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

```
integer*8 peph
integer res
character(len=CALCEPH_MAX_CONSTANTVALUE) UNIT

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  ! print the value of UNIT
  if (f90calceph_getconstantss(peph, "UNIT", UNIT).eq.1) then
    write (*,*) "UNIT=", trim(UNIT)
  endif
endif

call f90calceph_close(peph)
endif
```

4.4.16 f90calceph_getconstantvs

function f90calceph_getconstantvs (*eph, name, arrayvalue, nvalue*)

Parameters

- **eph** [*INTEGER(8), intent(in)*] :: ephemeris descriptor
- **name** [*CHARACTER(len=*)*, *intent(in)*] :: name of the constant
- **value** [*CHARACTER(len=CALCEPH_MAX_CONSTANTNAME)*, *dimension(1:nvalue)*, *intent(inout)*] :: array of values for the constant
- **nvalue** [*INTEGER, intent(in)*] :: number of elements of the array

Return f90calceph_getconstantvs [*INTEGER*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

This function stores, to the array *arrayvalue* as strings of characters, the *nvalue* first values associated to the constant *name* in the header of the ephemeris file associated to *eph*. The integer value returned by the function is equal to the number of valid entries in the *arrayvalue* if *nvalue* is greater or equal to that integer value.

The required value *nvalue* to store all values can be determined with the previous call to *f90calceph_getconstantss*.

The values must be strings, otherwise an error is reported.

Trailing blanks are added to each value.

The following example prints the units of the mission stored in the ephemeris file

```
integer*8 peph
integer res, nvalue
character(len=CALCEPH_MAX_CONSTANTVALUE), allocatable :: mission_units
character(len=CALCEPH_MAX_CONSTANTVALUE) svalue

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  ! get the number of values
```

(continues on next page)

(continued from previous page)

```

nvalue = calceph_getconstantss(peph, "MISSION_UNITS", svalue)
! fill the array
allocate(mission_units(1:nvalue))
res = calceph_getconstantvs(peph, "MISSION_UNITS", mission_units, nvalue)
write(*,*) mission_units

call f90calceph_close(peph)
endif

```

4.4.17 f90calceph_getconstantcount

function f90calceph_getconstantcount (*eph*)

Parameters *eph* [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor

Return **f90calceph_getconstantcount** [*INTEGER*] :: number of constants. 0 if an error occurs, otherwise non-zero value.

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

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

```

integer*8 peph
integer res
integer n

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  n = f90calceph_getconstantcount(peph)
  write (*,*) "number of constants", n
  call f90calceph_close(peph)
endif

```

4.4.18 f90calceph_getconstantindex

function f90calceph_getconstantindex (*eph*, *index*, *name*, *value*)

Parameters

- **eph** [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor
- **index** [*INTEGER*, *intent(in)*] :: index of the constant, between 1 and *f90calceph_getconstantcount* ()
- **name** [*CHARACTER*(*len*=*CALCEPH_MAX_CONSTANTNAME*), *intent(out)*] :: name of the constant
- **value** [*REAL*(8), *intent(out)*] :: first value of the constant

Return **f90calceph_getconstantindex** [*INTEGER*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

This function returns the name and its value of the constant available at the specified index in the header of the ephemeris file associated to *eph*. The value of *index* must be between 1 and *f90calceph_getconstantcount* ().

Only the first value is returned if multiple values are associated to a constant, such as a list of values. If the first value is not an floating-point number, such as a string, then the function returns 0 without raising an error.

Trailing blanks are added to the name of the constant.

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

```
integer*8 peph
integer res
integer n

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  do j=1, f90calceph_getconstantcount(peph)
    res = f90calceph_getconstantindex(peph, j, nameconstant, valueconstant)
    write (*,*) nameconstant, "=", valueconstant
  enddo
  call f90calceph_close(peph)
endif
```

4.4.19 f90calceph_getfileversion

function f90calceph_getfileversion (*eph, version*)

Parameters

- **eph** [*INTEGER(8), intent(in)*] :: ephemeris descriptor
- **version** [*CHARACTER(len=CALCEPH_MAX_CONSTANTVALUE), intent(out)*] :: version of the file

Return f90calceph_getfileversion [*INTEGER*] :: returns 0 if the file version was not found, otherwise non-zero value.

This function returns the version of the ephemeris file, as a string. For example, the argument version will contain 'INPOP10B', 'EPM2017' or 'DE405',

If the file is an original JPL binary planetary ephemeris, then the version of the file can always be determined. If the file is a spice kernel, the version of the file is retrieved from the constant *INPOP_PCK_VERSION*, *EPM_PCK_VERSION*, or *PCK_VERSION*.

The following example prints the version of the ephemeris file.

```
integer*8 peph
integer res
character(len=CALCEPH_MAX_CONSTANTVALUE) version

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then

  res = f90calceph_getfileversion(peph, version)
  write (*,*) "The version of the file is ", version

  call f90calceph_close(peph)
endif
```

4.4.20 f90calceph_gettimescale

function f90calceph_gettimescale (*eph*)

Parameters *eph* [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor

Return f90calceph_gettimescale [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function returns the timescale of the ephemeris file associated to *eph* :

- 1 if the quantities of all bodies are expressed in the TDB time scale.
- 2 if the quantities of all bodies are expressed in the TCB time scale.

The following example prints the time scale available in the ephemeris file

```
integer*8 peph
integer res
double precision AU

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  ! print the time scale
  timescale = calceph_gettimescale(peph)
  write (*,*) "timescale=", timescale

  call f90calceph_close(peph)
endif
```

4.4.21 f90calceph_gettimespan

function f90calceph_gettimespan (*eph*, *firsttime*, *lasttime*, *continuous*)

Parameters

- **eph** [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor
- **firsttime** [*REAL*(8), *intent(out)*] :: Julian date of the first time
- **lasttime** [*REAL*(8), *intent(out)*] :: Julian date of the last time
- **continuous** [*INTEGER*, *intent(out)*] :: information about the availability of the quantities over the time span

Return f90calceph_gettimespan [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function returns the first and last time available in the ephemeris file associated to *eph*. The Julian date for the first and last time are expressed in the time scale returned by *f90calceph_gettimescale()* .

It returns the following value in the parameter *continuous* :

- 1 if the quantities of all bodies are available for any time between the first and last time.
- 2 if the quantities of some bodies are available on discontinuous time intervals between the first and last time.
- 3 if the quantities of each body are available on a continuous time interval between the first and last time, but not available for any time between the first and last time.

The following example prints the first and last time available in the ephemeris file


```

integer*8 peph
integer res
integer continuous
double precision firsttime, lasttime

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  if (f90calceph_gettimespan(peph, firsttime, lasttime, continuous).eq.1) then
    write (*,*) firsttime, lasttime, continuous
  endif
endif

call f90calceph_close(peph)
endif

```

4.4.22 f90calceph_getpositionrecordcount

function f90calceph_getpositionrecordcount (*eph*)

Parameters *eph* [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor

Return f90calceph_getpositionrecordcount [*INTEGER*] :: number of position's records. 0 if an error occurs, otherwise non-zero value.

This function returns the number of position's records available in the ephemeris file associated to *eph*. Usually, the number of records is equal to the number of bodies in the ephemeris file if the timespan is continuous. If the timespan is discontinuous for the target and center bodies, then each different timespan is counted as a different record. If the ephemeris file contain timescale transformations' records, such as *TT-TDB* or *TCG-TCB*, then these records are included in the returned value.

The following example prints the number of position's records available in the ephemeris file

```

integer*8 peph
integer res
integer n

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  n = f90calceph_getpositionrecordcount(peph)
  write (*,*) "number of position's record", n
  call f90calceph_close(peph)
endif

```

4.4.23 f90calceph_getpositionrecordindex

function f90calceph_getpositionrecordindex (*eph*, *index*, *target*, *center*, *firsttime*, *lasttime*, *frame*)

Parameters

- **eph** [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor
- **index** [*INTEGER*, *intent(int)*] :: index of the position's record, between 1 and *f90calceph_getpositionrecordcount()*
- **target** [*INTEGER*, *intent(out)*] :: The target body
- **center** [*INTEGER*, *intent(out)*] :: The origin body

- **firsttime** [*REAL(8), intent(out)*] :: Julian date of the first time
- **lasttime** [*REAL(8), intent(out)*] :: Julian date of the last time
- **frame** [*INTEGER, intent(out)*] :: reference frame (see the list, below)

Return **f90calceph_getpositionrecordindex** [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function returns the target and origin bodies, the first and last time, and the reference frame available at the specified index for the position's records of the ephemeris file associated to *eph*. The NAIF identification numbering system is used for the target and center integers (*NAIF identification numbers* for the list). The Julian date for the first and last time are expressed in the time scale returned by *f90calceph_gettimescale()*.

It returns the following value in the parameter *frame* :

| value | Name |
|-------|------|
| 1 | ICRF |

The following example displays the position's records stored in the ephemeris file.

```
integer*8 peph
integer j, itarget, icenter, iframe
real*8 firsttime, lasttime
integer res

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  do j=1, f90calceph_getpositionrecordcount(peph)
    res = f90calceph_getpositionrecordindex(peph,j,itarget, icenter, firsttime,
↪lasttime, iframe)
    write (*,*) itarget, icenter, firsttime, lasttime, iframe
  enddo
  call f90calceph_close(peph)
endif
```

4.4.24 f90calceph_getpositionrecordindex2

function **f90calceph_getpositionrecordindex2** (*eph, index, target, center, firsttime, lasttime, frame, segid*)

Parameters

- **eph** [*INTEGER(8), intent(in)*] :: ephemeris descriptor
- **index** [*INTEGER, intent(int)*] :: index of the position's record, between 1 and *f90calceph_getpositionrecordcount()*
- **target** [*INTEGER, intent(out)*] :: The target body
- **center** [*INTEGER, intent(out)*] :: The origin body
- **firsttime** [*REAL(8), intent(out)*] :: Julian date of the first time
- **lasttime** [*REAL(8), intent(out)*] :: Julian date of the last time
- **frame** [*INTEGER, intent(out)*] :: reference frame (see the list, below)
- **segid** [*INTEGER, intent(out)*] :: type of the segment.

Return `f90calceph_getpositionrecordindex2` [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function returns the target and origin bodies, the first and last time, the reference frame, and the segment type available at the specified index for the position's records of the ephemeris file associated to *eph*. The NAIF identification numbering system is used for the target and center integers (*NAIF identification numbers* for the list). The Julian date for the first and last time are expressed in the time scale returned by `f90calceph_gettimescale()`.

It returns the following value in the parameter *frame* :

| value | Name |
|-------|------|
| 1 | ICRF |

It returns in the parameter *segid* one of the predefined constants `CALCEPH_SEGTYPE_...` (*Constants*).

The following example displays the position's records stored in the ephemeris file.

```
integer*8 peph
integer j, itarget, icenter, iframe, iseg
real*8 firsttime, lasttime
integer res

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  do j=1, f90calceph_getpositionrecordcount(peph)
    res = f90calceph_getpositionrecordindex2(peph,j,itarget, icenter, firsttime,
    ↪lasttime, iframe, iseg)
    write (*,*) itarget, icenter, firsttime, lasttime, iframe, iseg
  enddo
  call f90calceph_close(peph)
endif
```

4.4.25 f90calceph_getorientrecordcount

function `f90calceph_getorientrecordcount` (*eph*)

Parameters *eph* [*INTEGER*(8), *intent(in)*] :: ephemeris descriptor

Return `f90calceph_getorientrecordcount` [*INTEGER*] :: number of orientation's records. 0 if an error occurs, otherwise non-zero value.

This function returns the number of orientation's records available in the ephemeris file associated to *eph*. Usually, the number of records is equal to the number of bodies in the ephemeris file if the timespan is continuous. If the timespan is discontinuous for the target body, then each different timespan is counted as a different record.

The following example prints the number of orientation's records available in the ephemeris file

```
integer*8 peph
integer res
integer n

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  n = f90calceph_getorientrecordcount(peph)
  write (*,*) "number of orientation's record", n
  call f90calceph_close(peph)
endif
```

4.4.26 f90calceph_getorientrecordindex

function f90calceph_getpositionrecordindex (*eph, index, target, firsttime, lasttime, frame*)

Parameters

- **eph** [*INTEGER(8), intent(in)*] :: ephemeris descriptor
- **index** [*INTEGER, intent(int)*] :: index of the orientation's record, between 1 and *f90calceph_getorientrecordcount()*
- **target** [*INTEGER, intent(out)*] :: The target body
- **firsttime** [*REAL(8), intent(out)*] :: Julian date of the first time
- **lasttime** [*REAL(8), intent(out)*] :: Julian date of the last time
- **frame** [*INTEGER, intent(out)*] :: reference frame (see the list, below)

Return f90calceph_getorientrecordindex [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function returns the target body, the first and last time, and the reference frame available at the specified index for the orientation's records of the ephemeris file associated to *eph*. The NAIF identification numbering system is used for the target body (*NAIF identification numbers* for the list). The Julian date for the first and last time are expressed in the time scale returned by *f90calceph_gettimescale()*.

It returns the following value in the parameter *frame* :

| value | Name |
|-------|------|
| 1 | ICRF |

The following example displays the orientation's records stored in the ephemeris file.

```
integer*8 peph
integer j, itarget, iframe
real*8 firsttime, lasttime
integer res

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  do j=1, f90calceph_getorientrecordcount(peph)
    res = f90calceph_getorientrecordindex(peph,j,itarget, firsttime, lasttime, iframe)
    write (*,*) itarget, firsttime, lasttime, iframe
  enddo
  call f90calceph_close(peph)
endif
```

4.4.27 f90calceph_getorientrecordindex2

function f90calceph_getpositionrecordindex2 (*eph, index, target, firsttime, lasttime, frame, segid*)

Parameters

- **eph** [*INTEGER(8), intent(in)*] :: ephemeris descriptor
- **index** [*INTEGER, intent(int)*] :: index of the orientation's record, between 1 and *f90calceph_getorientrecordcount()*

- **target** [*INTEGER, intent(out)*] :: The target body
- **firsttime** [*REAL(8), intent(out)*] :: Julian date of the first time
- **lasttime** [*REAL(8), intent(out)*] :: Julian date of the last time
- **frame** [*INTEGER, intent(out)*] :: reference frame (see the list, below)
- **segid** [*INTEGER, intent(out)*] :: type of the segment.

Return **f90calceph_getorientrecordindex2** [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function returns the target body, the first and last time, the reference frame and the segment type available at the specified index for the orientation's records of the ephemeris file associated to *eph*. The NAIF identification numbering system is used for the target body (*NAIF identification numbers* for the list). The Julian date for the first and last time are expressed in the time scale returned by *f90calceph_gettimescale()*.

It returns the following value in the parameter *frame* :

| value | Name |
|-------|------|
| 1 | ICRF |

It returns in the parameter *segid* one of the predefined constants *CALCEPH_SEGTYPE_...* (*Constants*).

The following example displays the orientation's records stored in the ephemeris file.

```
integer*8 peph
integer j, itarget, iframe, iseg
real*8 firsttime, lasttime
integer res

res = f90calceph_open(peph, "example1.dat")
if (res.eq.1) then
  do j=1, f90calceph_getorientrecordcount(peph)
    res = f90calceph_getorientrecordindex2(peph, j, itarget, firsttime, lasttime,
    ↪ iframe, iseg)
    write (*,*) itarget, firsttime, lasttime, iframe, iseg
  enddo
  call f90calceph_close(peph)
endif
```

4.4.28 f90calceph_close

subroutine **f90calceph_close** (*eph*)

Parameters **eph** [*INTEGER(8), intent(in)*] :: ephemeris descriptor

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

SINGLE FILE ACCESS FUNCTIONS

This group of functions works on a single 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 *f90calceph_seterrorhandler()*.

The python interface does not provide these functions, the listed in the section *Multiple file access functions* must be used.

5.1 Time notes

The function *f90calceph_scompute()* only accepts a date expressed in the same timescale as the ephemeris files, which can be retrieved using the function *f90calceph_sgettimescale()*. Ephemeris files are generally expressed using the timescale TDB. If a date, expressed in the TT (Terrestrial Time) timescale, is supplied to this function, *f90calceph_scompute()* will return an erroneous position of the order of several tens of meters for the planets. If a date, expressed in the Coordinated Universal Time (UTC), is supplied to this function, *f90calceph_scompute()* will return a very large erroneous position over several thousand kilometers for the planets.

5.2 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 *f90calceph_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 *f90calceph_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.

5.3 Usage

The following examples, that can be found in the directory *examples* of the library sources, show the typical usage of this group of functions.

The example in Fortran 77/90/95 language is *f77single.f*.

```

program f77single
  implicit none
  include 'f90calceph.h'
  integer res
  double precision AU, EMRAT, GM_Mer
  double precision jd0
  double precision dt
  double precision PV(6)
  integer j
  double precision valueconstant
  character(len=CALCEPH_MAX_CONSTANTNAME) nameconstant

  jd0 = 2442457
  dt = 0.5E0
  ! open the ephemeris file
  res = f90calceph_sopen("example1.dat")
  if (res.eq.1) then
    write (*,*) "The ephemeris is already opened"
    ! print the values of AU, EMRAT and GM_Mer
    if (f90calceph_sgetconstant("AU", AU).eq.1) then
      write (*,*) "AU=", AU
    endif
    if (f90calceph_sgetconstant("EMRAT", EMRAT).eq.1) then
      write (*,*) "EMRAT=", EMRAT
    endif
    if (f90calceph_sgetconstant("GM_Mer", GM_Mer).eq.1) then
      write (*,*) "GM_Mer=", GM_Mer
    endif

    ! compute and print the coordinates
    ! the geocentric moon coordinates
    res = f90calceph_scompute(jd0, dt, 10, 3, PV)
    call printcoord(PV, "geocentric coordinates of the Moon")
    ! the value TT-TDB

```

(continues on next page)

(continued from previous page)

```

    if (f90calceph_scompute(jd0, dt, 16, 0, PV).eq.1) then
      write (*,*) "TT-TDB = ", PV(1)
    endif
    ! the heliocentric coordinates of Mars
    res = f90calceph_scompute(jd0, dt, 4, 11, PV)
    call printcoord(PV,"heliocentric coordinates of Mars")

    ! print the whole list of the constants
    write (*,*) "list of constants"
    do j=1, f90calceph_sgetconstantcount()
      res = f90calceph_sgetconstantindex(j,nameconstant,valueconstant)
      write (*,*) nameconstant,"=",valueconstant
    enddo

    ! close the ephemeris file
    call f90calceph_sclose
    write (*,*) "The ephemeris is already closed"
  else
    write (*,*) "The ephemeris can't be opened"
  endif
stop
end

```

5.4 Functions

5.4.1 f90calceph_sopen

function f90calceph_sopen (filename)

Parameters filename [CHARACTER(len=*), intent(in)] :: pathname of the file.

Return f90calceph_sopen [INTEGER] :: 0 if an error occurs, otherwise non-zero value.

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

This file must be compliant to the format specified by the 'original JPL binary', 'INPOP 2.0 binary' or 'SPICE' ephemeris file. At the moment, supported SPICE files are the following :

- text Planetary Constants Kernel (KPL/PCK) files
- binary PCK (DAF/PCK) files.
- binary SPK (DAF/SPK) files containing segments of type 1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 20, 21, 102, 103 and 120.
- meta kernel (KPL/MK) files.
- frame kernel (KPL/FK) files. Only a basic support is provided.

The function `f90calceph_sclose()` must be called to free allocated memory by this function.

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

```

integer res
res = f90calceph_sopen("example1.dat")
if (res.eq.1) then

```

(continues on next page)

(continued from previous page)

```

! ... computation ...
endif
call f90calceph_sclose

```

5.4.2 f90calceph_scompute

function **f90calceph_scompute** (*JD0, time, target, center, PV*)

Parameters

- **JD0** [*REAL(8), intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL(8), intent(in)*] :: Fraction part of the Julian date (TDB or TCB)
- **target** [*INTEGER, intent(in)*] :: The body or reference point whose coordinates are required (see the list, below).
- **center** [*INTEGER, intent(in)*] :: The origin of the coordinate system (see the list, below). If *target* is 14, 15, 16 or 17 (nutaton, libration, TT-TDB or TCG-TCB), *center* must be 0.
- **PV** (6) [*REAL(8), intent(out)*] :: Depending on the target value, an array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot), or a time scale transformation value, or the angles of the librations of the Moon and their derivatives, or the nutation angles and their derivatives.

Return **f90calceph_scompute** [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

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 ephemeris file, previously opened with the function *f90calceph_sopen()*, for the time *JD0+time* and stores the results to *PV*.

The date (*JD0, time*) should be expressed in the same timescale as the ephemeris files, which can be retrieved using the function *f90calceph_sgettimescale()*.

Warning: If a date, expressed in the Coordinated Universal Time (UTC), is supplied to this function, a very large erroneous position will be returned.

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 Barycenter |
| 2 | Venus Barycenter |
| 3 | Earth |
| 4 | Mars Barycenter |
| 5 | Jupiter Barycenter |
| 6 | Saturn Barycenter |
| 7 | Uranus Barycenter |
| 8 | Neptune Barycenter |
| 9 | Pluto Barycenter |
| 10 | Moon |
| 11 | Sun |
| 12 | Solar Sytem barycenter |
| 13 | Earth-moon barycenter |
| 14 | Nutations |
| 15 | Librations |
| 16 | TT-TDB |
| 17 | TCG-TCB |
| asteroid number + CALCEPH_asteroid | asteroid |

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

For example, the value "CALCEPH_asteroid+4" for target or center specifies the asteroid Vesta.

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

```
double precision PV(6)
integer res
double precision jd0
double precision dt1, dt2

jd0 = 2442457
dt  = 0.5D0
dt2 = 0.9D0

res = f90calceph_sopen("example1.dat")
if (res.eq.1) then

    ! the heliocentric coordinates of Mars
    res = f90calceph_scompute(jd0, dt1, 4, 11, PV)
    write (*,*) PV

    res = f90calceph_scompute(jd0, dt2, 4, 11, PV)
    write (*,*) PV

    call f90calceph_sclose
endif
```

5.4.3 f90calceph_sgetconstant

function f90calceph_sgetconstant (*name*, *value*)

Parameters

- **name** [*CHARACTER(len=*)*, *intent(in)*] :: name of the constant
- **value** [*REAL(8)*, *intent(out)*] :: first value of the constant

Return f90calceph_sgetconstant [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

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

Only the first value is returned if multiple values are associated to a constant, such as a list of values.

The function *f90calceph_sopen()* must be previously called before.

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

```
integer res
double precision AU

res = f90calceph_sopen("example1.dat")
if (res.eq.1) then
    if (f90calceph_sgetconstant("AU", AU).eq.1) then
        write (*,*) "AU=", AU
    endif
endif
endif
```

5.4.4 f90calceph_sgetconstantcount

function f90calceph_sgetconstantcount ()

Return f90calceph_prefetch :: 0 if an error occurs, otherwise non-zero value.

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

The function *f90calceph_sopen()* must be previously called before.

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

```
integer res
integer n
! open the ephemeris file
res = f90calceph_sopen("example1.dat")
if (res.eq.1) then

    n = f90calceph_sgetconstantcount()
    write (*,*) "number of constants", n

    call f90calceph_sclose
endif
```

5.4.5 f90calceph_sgetconstantindex

function `f90calceph_sgetconstantindex` (*index*, *name*, *value*)

Parameters

- **index** [*INTEGER*, *intent(in)*] :: index of the constant, between 1 and `f90calceph_getconstantcount()`
- **name** [*CHARACTER(len=*`CALCEPH_MAX_CONSTANTNAME`*)*, *intent(out)*] :: name of the constant
- **value** [*REAL(8)*, *intent(out)*] :: first value of the constant

Return `f90calceph_sgetconstantindex` [*INTEGER*] :: 0 if an error occurs, otherwise non-zero value.

This function returns the name and its value of the constant available at the specified index in the header of the ephemeris file. The value of *index* must be between 1 and `f90calceph_sgetconstantcount()`.

Trailing blanks are added to the name of the constant.

The function `f90calceph_sopen()` must be previously called before.

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

```
integer res
integer j
double precision valueconstant
character(len=CALCEPH_MAX_CONSTANTNAME) nameconstant

res = f90calceph_sopen("example1.dat")
if (res.eq.1) then

  do j=1, f90calceph_sgetconstantcount()
    res = f90calceph_sgetconstantindex(j,nameconstant,valueconstant)
    write (*,*) nameconstant,"=",valueconstant
  enddo

  call f90calceph_sclose
endif
```

5.4.6 f90calceph_sgetfileversion

function `f90calceph_sgetfileversion` (*version*)

Parameters *version* [*CHARACTER(len=*`CALCEPH_MAX_CONSTANTVALUE`*)*, *intent(out)*] :: version of the file

Return `f90calceph_sgetfileversion` [*INTEGER*] :: returns 0 if the file version was not found, otherwise non-zero value.

This function returns the version of the ephemeris file, as a string. For example, the argument *version* will contain 'INPOP10B', 'EPM2017' or 'DE405',

If the file is an original JPL binary planetary ephemeris, then the version of the file can always be determined. If the file is a spice kernel, the version of the file is retrieved from the constant `INPOP_PCK_VERSION`, `EPM_PCK_VERSION`, or `PCK_VERSION`.

The function `f90calceph_sopen()` must be previously called before.

The following example prints the version of the ephemeris file.

```
integer res
character(len=CALCEPH_MAX_CONSTANTVALUE) version
! open the ephemeris file
res = f90calceph_sopen("example1.dat")
if (res.eq.1) then

    res = f90calceph_sgetfileversion(version)
    write (*,*) "The version of the file is ", version

    call f90calceph_sclose
endif
```

5.4.7 f90calceph_sgettimescale

function f90calceph_sgettimescale()

Return f90calceph_sgettimescale [INTEGER] :: 0 if an error occurs, otherwise non-zero value.

This function returns the timescale of the ephemeris file :

- 1 if the quantities of all bodies are expressed in the TDB time scale.
- 2 if the quantities of all bodies are expressed in the TCB time scale.

The function *f90calceph_sopen()* must be previously called before.

The following example prints the time scale available in the ephemeris file

```
integer res
integer t
! open the ephemeris file
res = f90calceph_sopen("example1.dat")
if (res.eq.1) then

    t = f90calceph_sgettimescale()
    write (*,*) "timescale ", t

    call f90calceph_sclose
endif
```

5.4.8 f90calceph_sgettimespan

function f90calceph_sgettimespan (*firsttime, lasttime, continuous*)

Parameters

- **firsttime** [REAL(8), intent(out)] :: Julian date of the first time
- **lasttime** [REAL(8), intent(out)] :: Julian date of the last time
- **continuous** [INTEGER, intent(out)] :: information about the availability of the quantities over the time span

Return f90calceph_sgettimespan [INTEGER] :: 0 if an error occurs, otherwise non-zero value.

This function returns the first and last time available in the ephemeris file. The Julian date for the first and last time are expressed in the time scale returned by *f90calceph_sgettimescale()* .

It returns the following value in the parameter *continuous* :

- 1 if the quantities of all bodies are available for any time between the first and last time.
- 2 if the quantities of some bodies are available on discontinuous time intervals between the first and last time.
- 3 if the quantities of each body are available on a continuous time interval between the first and last time, but not available for any time between the first and last time.

The function `f90calceph_sopen()` must be previously called before.

The following example prints the first and last time available in the ephemeris file

```
integer res
integer cont
double precision jdfirst, jdlast
! open the ephemeris file
res = f90calceph_sopen("example1.dat")
if (res.eq.1) then

    res = f90calceph_sgettimespan(jdfirst, jdlast, cont)
    write (*,*) "data available between ", jdfirst, "and", jdlast
    write (*,*) "continuous data ", cont

    call f90calceph_sclose
endif
```

5.4.9 f90calceph_sclose

subroutine f90calceph_sclose()

This function closes the ephemeris data file and frees allocated memory by the function `f90calceph_sopen()`.

ERROR FUNCTIONS

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

6.1 Usage

The following examples, that can be found in the directory *examples* of the library sources, show the typical usage of this group of functions.

The example in Fortran 77/90/95 language is `f77error.f`.

The following example shows how to stop the execution on the error.

```
program f77error
  implicit none
  include 'f90calceph.h'
  integer*8 peph
  integer res

  ! set the error handler to stop on error
  call f90calceph_seterrorhandler(2, 0)

  ! open the ephemeris file
  res = f90calceph_open(peph, "example1.dat")
  ! ...
stop
end
```

The following example shows how to define a custom error handler function.

```
!-----
! custom error handler
!-----
subroutine myhandler(msg)
  implicit none
  character(len=*) :: msg
  write (*,*) "The calceph calls the function myhandler"
  write (*,*) "The message contains ",len(msg)," characters"
  write (*,*) "The error message is :"
  write (*,*) "-----"
  write (*,*) msg
  write (*,*) "-----"
  write (*,*) "The error handler returns"
end
```

(continues on next page)

(continued from previous page)

```
!-----  
! main program  
!-----  
program f77error  
  implicit none  
  include 'f90calceph.h'  
  integer res  
  integer*8 peph  
  external myhandler  
  
  ! set the error handler to use my own callback  
  call f90calceph_seterrorhandler(3, myhandler)  
  
  ! open the ephemeris file  
  res = f90calceph_open(peph, "example1.dat")  
  ! ...  
stop  
end
```

6.2 f90calceph_seterrorhandler

subroutine f90calceph_seterrorhandler (*typehandler, userfunc*)

Parameters

- **typehandler** [*INTEGER, intent(in)*] :: type of handler
- **userfunc** [*EXTERNAL, intent(in)*] :: user function

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. The python and Octave/Matlab interfaces raise an exception. 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 <i>exit</i> . |
| 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* is ignored and it may be an empty function.

The function *userfunc* must be defined as

```
subroutine userfunc (msg)
implicit none
CHARACTER(len=*) , intent(in) :: msg
```

This function must be declared as **EXTERNAL**

MISCELLANEOUS FUNCTIONS

7.1 f90calceph_getmaxsupportedorder

subroutine `f90calceph_getmaxsupportedorder` (*segid*)

Parameters `version` [*INTEGER*, *intent(in)*] :: type of the segment.

Return `f90calceph_getmaxsupportedorder` [*INTEGER*] :: maximal order of the computable derivatives for this type of segment.

This function returns the maximal order of the derivatives computed by the functions `f90calceph_compute_order()`, `f90calceph_orient_order()`, for the segment type *segid*. If the segment type is unknown by the library, the function returns -1.

The accepted values of *segid* are the predefined constants `CALCEPH_SEGTYPE_...` (*Constants*).

```
integer maxorder

maxorder = calceph_getmaxsupportedorder(CALCEPH_SEGTYPE_SPK_2)
write(*,*) 'maximal order is ', maxorder
```

7.2 f90calceph_getversion_str

subroutine `f90calceph_getversion_str` (*version*)

Parameters `version` [*CHARACTER(len=CALCEPH_MAX_CONSTANTNAME)*, *intent(out)*] :: version of the library

This function returns the version of the CALCEPH Library, as a string.

Trailing blanks are added to the name version.

```
character(len=CALCEPH_MAX_CONSTANTNAME) version

call f90calceph_getversion_str(version)
write(*,*) 'library version is ', version
```


NAIF IDENTIFICATION NUMBERS

The following predefined values must be used as the target body and origin of the coordinate system with the functions `f90calceph_compute_unit()`, `f90calceph_orient_unit()`, `f90calceph_compute_order()` or `f90calceph_orient_order()` if and only if the value `CALCEPH_USE_NAIFID` has been set in the parameter `unit`.

This list is already predefined in the interface file `f90calceph.h`.

8.1 Sun and planetary barycenters

| Predefined Macros | NAIF ID | Name |
|--------------------------------|---------|-------------------------|
| NAIFID_SOLAR_SYSTEM_BARYCENTER | 0 | Solar System Barycenter |
| NAIFID_MERCURY_BARYCENTER | 1 | Mercury Barycenter |
| NAIFID_VENUS_BARYCENTER | 2 | Venus Barycenter |
| NAIFID_EARTH_MOON_BARYCENTER | 3 | Earth-Moon Barycenter |
| NAIFID_MARS_BARYCENTER | 4 | Mars Barycenter |
| NAIFID_JUPITER_BARYCENTER | 5 | Jupiter Barycenter |
| NAIFID_SATURN_BARYCENTER | 6 | Saturn Barycenter |
| NAIFID_URANUS_BARYCENTER | 7 | Uranus Barycenter |
| NAIFID_NEPTUNE_BARYCENTER | 8 | Neptune Barycenter |
| NAIFID_PLUTO_BARYCENTER | 9 | Pluto Barycenter |
| NAIFID_SUN | 10 | Sun |

8.2 Coordinate Time ephemerides

| Predefined Macros | NAIF ID | Name |
|---------------------|------------|--|
| NAIFID_TIME_CENTER | 1000000000 | center ID for Coordinate Time ephemerides ¹ |
| NAIFID_TIME_TTMTDB | 1000000001 | Coordinate Time ephemeride TT-TDB ² |
| NAIFID_TIME_TCGMTCB | 1000000002 | Coordinate Time ephemeride TCG-TCB ² |

¹ These values must only be used as a center body.

² These values must only be used as a target body.

8.3 Planet centers and satellites

| Predefined Macros | NAIF ID | Name |
|-------------------|---------|------------|
| NAIFID_MERCURY | 199 | Mercury |
| NAIFID_VENUS | 299 | Venus |
| NAIFID_EARTH | 399 | Earth |
| NAIFID_MOON | 301 | Moon |
| NAIFID_MARS | 499 | Mars |
| NAIFID_PHOBOS | 401 | Phobos |
| NAIFID_DEIMOS | 402 | Deimos |
| NAIFID_JUPITER | 599 | Jupiter |
| NAIFID_IO | 501 | Io |
| NAIFID_EUROPA | 502 | Europa |
| NAIFID_GANYMEDE | 503 | Ganymede |
| NAIFID_CALLISTO | 504 | Callisto |
| NAIFID_AMALTHEA | 505 | Amalthea |
| NAIFID_HIMALIA | 506 | Himalia |
| NAIFID_ELARA | 507 | Elara |
| NAIFID_PASIPHAЕ | 508 | Pasiphae |
| NAIFID_SINOPE | 509 | Sinope |
| NAIFID_LYSITHEA | 510 | Lysithea |
| NAIFID_CARME | 511 | Carme |
| NAIFID_ANANKE | 512 | Ananke |
| NAIFID_LEDA | 513 | Leda |
| NAIFID_THEBE | 514 | Thebe |
| NAIFID_ADRASTEА | 515 | Adrastea |
| NAIFID_METIS | 516 | Metis |
| NAIFID_CALLIRRHOE | 517 | Callirrhoe |
| NAIFID_THEMISTO | 518 | Themisto |
| NAIFID_MEGACLITE | 519 | Megaclite |
| NAIFID_TAYGETE | 520 | Taygete |
| NAIFID_CHALDENE | 521 | Chaldene |
| NAIFID_HARPALYKE | 522 | Harpalyke |
| NAIFID_KALYKE | 523 | Kalyke |
| NAIFID_IOCASTE | 524 | Iocaste |
| NAIFID_ERINOME | 525 | Erinome |
| NAIFID_ISONOE | 526 | Isonoe |
| NAIFID_PRAXIDIKE | 527 | Praxidike |
| NAIFID_AUTONOE | 528 | Autonoe |
| NAIFID_THYONE | 529 | Thyone |
| NAIFID_HERMIPPE | 530 | Hermippe |
| NAIFID_AITNE | 531 | Aitne |
| NAIFID_EURYDOME | 532 | Eurydome |
| NAIFID_EUANTHE | 533 | Euanthe |
| NAIFID_EUPORIE | 534 | Euporie |
| NAIFID_ORTHOСIE | 535 | Orthosie |

continues on next page

Table 1 – continued from previous page

| Predefined Macros | NAIF ID | Name |
|-------------------|---------|------------|
| NAIFID_SPONDE | 536 | Sponde |
| NAIFID_KALE | 537 | Kale |
| NAIFID_PASITHEE | 538 | Pasithee |
| NAIFID_HEGEMONE | 539 | Hegemone |
| NAIFID_MNEME | 540 | Mneme |
| NAIFID_AOEDE | 541 | Aoede |
| NAIFID_THELXINOE | 542 | Thelxinoe |
| NAIFID_ARCHE | 543 | Arche |
| NAIFID_KALLICHORE | 544 | Kallichore |
| NAIFID_HELIKE | 545 | Helike |
| NAIFID_CARPO | 546 | Carpo |
| NAIFID_EUKELADE | 547 | Eukelade |
| NAIFID_CYLLENE | 548 | Cyllene |
| NAIFID_KORE | 549 | Kore |
| NAIFID_HERSE | 550 | Herse |
| NAIFID_DIA | 553 | Dia |
| | | |
| NAIFID_SATURN | 699 | Saturn |
| NAIFID_MIMAS | 601 | Mimas |
| NAIFID_ENCELADUS | 602 | Enceladus |
| NAIFID_TETHYS | 603 | Tethys |
| NAIFID_DIONE | 604 | Dione |
| NAIFID_RHEA | 605 | Rhea |
| NAIFID_TITAN | 606 | Titan |
| NAIFID_HYPERION | 607 | Hyperion |
| NAIFID_IAPETUS | 608 | Iapetus |
| NAIFID_PHOEBE | 609 | Phoebe |
| NAIFID_JANUS | 610 | Janus |
| NAIFID_EPIMETHEUS | 611 | Epimetheus |
| NAIFID_HELENE | 612 | Helene |
| NAIFID_TELESTO | 613 | Telesto |
| NAIFID_CALYPSO | 614 | Calypso |
| NAIFID_ATLAS | 615 | Atlas |
| NAIFID_PROMETHEUS | 616 | Prometheus |
| NAIFID_PANDORA | 617 | Pandora |
| NAIFID_PAN | 618 | Pan |
| NAIFID_YMIR | 619 | Ymir |
| NAIFID_PAALIAQ | 620 | Paaliaq |
| NAIFID_TARVOS | 621 | Tarvos |
| NAIFID_IJIRAQ | 622 | Ijiraq |
| NAIFID_SUTTUNGR | 623 | Suttungr |
| NAIFID_KIVIUQ | 624 | Kiviuq |
| NAIFID_MUNDILFARI | 625 | Mundilfari |
| NAIFID_ALBIORIX | 626 | Albiorix |
| NAIFID_SKATHI | 627 | Skathi |
| NAIFID_ERRIAPUS | 628 | Erriapus |
| NAIFID_SIARNAQ | 629 | Siarnaq |
| NAIFID_THRYMR | 630 | Thrymr |
| NAIFID_NARVI | 631 | Narvi |

continues on next page

Table 1 – continued from previous page

| Predefined Macros | NAIF ID | Name |
|-------------------|---------|------------|
| NAIFID_METHONE | 632 | Methone |
| NAIFID_PALLENE | 633 | Pallene |
| NAIFID_POLYDEUCES | 634 | Polydeuces |
| NAIFID_DAPHNIS | 635 | Daphnis |
| NAIFID_AEGIR | 636 | Aegir |
| NAIFID_BEBHIONN | 637 | Bebhionn |
| NAIFID_BERGELMIR | 638 | Bergelmir |
| NAIFID_BESTLA | 639 | Bestla |
| NAIFID_FARBAUTI | 640 | Farbauti |
| NAIFID_FENRIR | 641 | Fenrir |
| NAIFID_FORNJOT | 642 | Fornjot |
| NAIFID_HATI | 643 | Hati |
| NAIFID_HYROKKIN | 644 | Hyrokkin |
| NAIFID_KARI | 645 | Kari |
| NAIFID_LOGE | 646 | Loge |
| NAIFID_SKOLL | 647 | Skoll |
| NAIFID_SURTUR | 648 | Surtur |
| NAIFID_ANTHE | 649 | Anthe |
| NAIFID_JARNSAXA | 650 | Jarnsaxa |
| NAIFID_GREIP | 651 | Greip |
| NAIFID_TARQEQ | 652 | Tarqeq |
| NAIFID_AEGAEON | 653 | Aegaeon |
| | | |
| NAIFID_URANUS | 799 | Uranus |
| NAIFID_ARIEL | 701 | Ariel |
| NAIFID_UMBRIEL | 702 | Umbriel |
| NAIFID_TITANIA | 703 | Titania |
| NAIFID_OBERON | 704 | Oberon |
| NAIFID_MIRANDA | 705 | Miranda |
| NAIFID_CORDELIA | 706 | Cordelia |
| NAIFID_OPHELIA | 707 | Ophelia |
| NAIFID_BIANCA | 708 | Bianca |
| NAIFID_CRESSIDA | 709 | Cressida |
| NAIFID_DESDEMONA | 710 | Desdemona |
| NAIFID_JULIET | 711 | Juliet |
| NAIFID_PORTIA | 712 | Portia |
| NAIFID_ROSALIND | 713 | Rosalind |
| NAIFID_BELINDA | 714 | Belinda |
| NAIFID_PUCK | 715 | Puck |
| NAIFID_CALIBAN | 716 | Caliban |
| NAIFID_SYCORAX | 717 | Sycorax |
| NAIFID_PROSPERO | 718 | Prospero |
| NAIFID_SETEBOS | 719 | Setebos |
| NAIFID_STEPHANO | 720 | Stephano |
| NAIFID_TRINCULO | 721 | Trinculo |
| NAIFID_FRANCISCO | 722 | Francisco |
| NAIFID_MARGARET | 723 | Margaret |
| NAIFID_FERDINAND | 724 | Ferdinand |
| NAIFID_PERDITA | 725 | Perdita |

continues on next page

Table 1 – continued from previous page

| Predefined Macros | NAIF ID | Name |
|-------------------|---------|-----------|
| NAIFID_MAB | 726 | Mab |
| NAIFID_CUPID | 727 | Cupid |
| | | |
| NAIFID_NEPTUNE | 899 | Neptune |
| NAIFID_TRITON | 801 | Triton |
| NAIFID_NEREID | 802 | Nereid |
| NAIFID_NAIAD | 803 | Naiad |
| NAIFID_THALASSA | 804 | Thalassa |
| NAIFID_DESPINA | 805 | Despina |
| NAIFID_GALATEA | 806 | Galatea |
| NAIFID_LARISSA | 807 | Larissa |
| NAIFID_PROTEUS | 808 | Proteus |
| NAIFID_HALIMEDE | 809 | Halimede |
| NAIFID_PSAMATHE | 810 | Psamathe |
| NAIFID_SAO | 811 | Sao |
| NAIFID_LAOMEDEIA | 812 | Laomedeia |
| NAIFID_NESO | 813 | Neso |
| | | |
| NAIFID_PLUTO | 999 | Pluto |
| NAIFID_CHARON | 901 | Charon |
| NAIFID_NIX | 902 | Nix |
| NAIFID_HYDRA | 903 | Hydra |
| NAIFID_KERBEROS | 904 | Kerberos |
| NAIFID_STYX | 905 | Styx |

8.4 Comets

| Predefined Macros | NAIF ID | Name |
|------------------------------|---------|-----------------------|
| NAIFID_AREND | 1000001 | Arend |
| NAIFID_AREND_RIGAUX | 1000002 | Arend-Rigaux |
| NAIFID_ASHBROOK_JACKSON | 1000003 | Ashbrook-Jackson |
| NAIFID_BOETHIN | 1000004 | Boethin |
| NAIFID_BORRELLY | 1000005 | Borrelly |
| NAIFID_BOWELL_SKIFF | 1000006 | Bowell-Skiff |
| NAIFID_BRADFIELD | 1000007 | Bradfield |
| NAIFID_BROOKS_2 | 1000008 | Brooks 2 |
| NAIFID_BRORSEN_METCALF | 1000009 | Brorsen-Metcalf |
| NAIFID_BUS | 1000010 | Bus |
| NAIFID_CHERNYKH | 1000011 | Chernykh |
| NAIFID_CHURYUMOV_GERASIMENKO | 1000012 | Churyumov-Gerasimenko |
| NAIFID_CIFFREO | 1000013 | Ciffreo |
| NAIFID_CLARK | 1000014 | Clark |
| NAIFID_COMAS_SOLA | 1000015 | Comas Sola |
| NAIFID_CROMMELIN | 1000016 | Crommelin |
| NAIFID_D__ARREST | 1000017 | D"Drrest |
| NAIFID_DANIEL | 1000018 | Daniel |
| NAIFID_DE_VICO_SWIFT | 1000019 | De Vico-Swift |

continues on next page

Table 2 – continued from previous page

| Predefined Macros | NAIF ID | Name |
|--------------------------------|---------|-------------------------|
| NAIFID_DENNING_FUJIKAWA | 1000020 | Denning-Fujikawa |
| NAIFID_DU_TOIT_1 | 1000021 | Du Toit 1 |
| NAIFID_DU_TOIT_HARTLEY | 1000022 | Du Toit-Hartley |
| NAIFID_DUTOIT_NEUJMIN_DELPORTE | 1000023 | Dutoit-Neujmin-Delporte |
| NAIFID_DUBIAGO | 1000024 | Dubiago |
| NAIFID_ENCKE | 1000025 | Encke |
| NAIFID_FAYE | 1000026 | Faye |
| NAIFID_FINLAY | 1000027 | Finlay |
| NAIFID_FORBES | 1000028 | Forbes |
| NAIFID_GEHRELS_1 | 1000029 | Gehrels 1 |
| NAIFID_GEHRELS_2 | 1000030 | Gehrels 2 |
| NAIFID_GEHRELS_3 | 1000031 | Gehrels 3 |
| NAIFID_GIACOBINI_ZINNER | 1000032 | Giacobini-Zinner |
| NAIFID_GICLAS | 1000033 | Giclas |
| NAIFID_GRIGG_SKJELLERUP | 1000034 | Grigg-Skjellerup |
| NAIFID_GUNN | 1000035 | Gunn |
| NAIFID_HALLEY | 1000036 | Halley |
| NAIFID_HANEDA_CAMPOS | 1000037 | Haneda-Campos |
| NAIFID_HARRINGTON | 1000038 | Harrington |
| NAIFID_HARRINGTON_ABELL | 1000039 | Harrington-Abell |
| NAIFID_HARTLEY_1 | 1000040 | Hartley 1 |
| NAIFID_HARTLEY_2 | 1000041 | Hartley 2 |
| NAIFID_HARTLEY_IRAS | 1000042 | Hartley-Iras |
| NAIFID_HERSCHEL_RIGOLLET | 1000043 | Herschel-Rigollet |
| NAIFID_HOLMES | 1000044 | Holmes |
| NAIFID_HONDA_MRKOS_PAJDUSAKOVA | 1000045 | Honda-Mrkos-Pajdusakova |
| NAIFID_HOWELL | 1000046 | Howell |
| NAIFID_IRAS | 1000047 | Iras |
| NAIFID_JACKSON_NEUJMIN | 1000048 | Jackson-Neujmin |
| NAIFID_JOHNSON | 1000049 | Johnson |
| NAIFID_KEARNS_KWEE | 1000050 | Kearns-Kwee |
| NAIFID_KLEMOLA | 1000051 | Klemola |
| NAIFID_KOHOUTEK | 1000052 | Kohoutek |
| NAIFID_KOJIMA | 1000053 | Kojima |
| NAIFID_KOPFF | 1000054 | Kopff |
| NAIFID_KOWAL_1 | 1000055 | Kowal 1 |
| NAIFID_KOWAL_2 | 1000056 | Kowal 2 |
| NAIFID_KOWAL_MRKOS | 1000057 | Kowal-Mrkos |
| NAIFID_KOWAL_VAVROVA | 1000058 | Kowal-Vavrova |
| NAIFID_LONGMORE | 1000059 | Longmore |
| NAIFID_LOVAS_1 | 1000060 | Lovas 1 |
| NAIFID_MACHHOLZ | 1000061 | Machholz |
| NAIFID_MAURY | 1000062 | Maury |
| NAIFID_NEUJMIN_1 | 1000063 | Neujmin 1 |
| NAIFID_NEUJMIN_2 | 1000064 | Neujmin 2 |
| NAIFID_NEUJMIN_3 | 1000065 | Neujmin 3 |
| NAIFID_OLBERS | 1000066 | Olbers |
| NAIFID_PETERS_HARTLEY | 1000067 | Peters-Hartley |
| NAIFID_PONS_BROOKS | 1000068 | Pons-Brooks |

continues on next page

Table 2 – continued from previous page

| Predefined Macros | NAIF ID | Name |
|--------------------------------|---------|-------------------------|
| NAIFID_PONS_WINNECKE | 1000069 | Pons-Winnecke |
| NAIFID_REINMUTH_1 | 1000070 | Reinmuth 1 |
| NAIFID_REINMUTH_2 | 1000071 | Reinmuth 2 |
| NAIFID_RUSSELL_1 | 1000072 | Russell 1 |
| NAIFID_RUSSELL_2 | 1000073 | Russell 2 |
| NAIFID_RUSSELL_3 | 1000074 | Russell 3 |
| NAIFID_RUSSELL_4 | 1000075 | Russell 4 |
| NAIFID_SANGUIN | 1000076 | Sanguin |
| NAIFID_SCHAUMASSE | 1000077 | Schaumasse |
| NAIFID_SCHUSTER | 1000078 | Schuster |
| NAIFID_SCHWASSMANN_WACHMANN_1 | 1000079 | Schwassmann-Wachmann 1 |
| NAIFID_SCHWASSMANN_WACHMANN_2 | 1000080 | Schwassmann-Wachmann 2 |
| NAIFID_SCHWASSMANN_WACHMANN_3 | 1000081 | Schwassmann-Wachmann 3 |
| NAIFID_SHAJSN_SCHALDACH | 1000082 | Shajn-Schaldach |
| NAIFID_SHOEMAKER_1 | 1000083 | Shoemaker 1 |
| NAIFID_SHOEMAKER_2 | 1000084 | Shoemaker 2 |
| NAIFID_SHOEMAKER_3 | 1000085 | Shoemaker 3 |
| NAIFID_SINGER_BREWSTER | 1000086 | Singer-Brewster |
| NAIFID_SLAUGHTER_BURNHAM | 1000087 | Slaughter-Burnham |
| NAIFID_SMIRNOVA_CHERNYKH | 1000088 | Smirnova-Chernykh |
| NAIFID_STEPHAN_OTERMA | 1000089 | Stephan-Oterma |
| NAIFID_SWIFT_GEHRELS | 1000090 | Swift-Gehrels |
| NAIFID_TAKAMIZAWA | 1000091 | Takamizawa |
| NAIFID_TAYLOR | 1000092 | Taylor |
| NAIFID_TEMPEL_1 | 1000093 | Tempel 1 |
| NAIFID_TEMPEL_2 | 1000094 | Tempel 2 |
| NAIFID_TEMPEL_TUTTLE | 1000095 | Tempel-Tuttle |
| NAIFID_TRITTON | 1000096 | Tritton |
| NAIFID_TSUCHINSHAN_1 | 1000097 | Tsuchinshan 1 |
| NAIFID_TSUCHINSHAN_2 | 1000098 | Tsuchinshan 2 |
| NAIFID_TUTTLE | 1000099 | Tuttle |
| NAIFID_TUTTLE_GIACOBINI_KRESAK | 1000100 | Tuttle-Giacobini-Kresak |
| NAIFID_VAISALA_1 | 1000101 | Vaisala 1 |
| NAIFID_VAN_BIESBROECK | 1000102 | Van Biesbroeck |
| NAIFID_VAN_HOUTEN | 1000103 | Van Houten |
| NAIFID_WEST_KOHOUTEK_IKEMURA | 1000104 | West-Kohoutek-Ikemura |
| NAIFID_WHIPPLE | 1000105 | Whipple |
| NAIFID_WILD_1 | 1000106 | Wild 1 |
| NAIFID_WILD_2 | 1000107 | Wild 2 |
| NAIFID_WILD_3 | 1000108 | Wild 3 |
| NAIFID_WIRTANEN | 1000109 | Wirtanen |
| NAIFID_WOLF | 1000110 | Wolf |
| NAIFID_WOLF_HARRINGTON | 1000111 | Wolf-Harrington |
| NAIFID_LOVAS_2 | 1000112 | Lovas 2 |
| NAIFID_URATA_NIIJIMA | 1000113 | Urata-Niijima |
| NAIFID_WISEMAN_SKIFF | 1000114 | Wiseman-Skiff |
| NAIFID_HELIN | 1000115 | Helin |
| NAIFID_MUELLER | 1000116 | Mueller |
| NAIFID_SHOEMAKER_HOLT_1 | 1000117 | Shoemaker-Holt 1 |

continues on next page

Table 2 – continued from previous page

| Predefined Macros | NAIF ID | Name |
|-----------------------------|---------|----------------------|
| NAIFID_HELIN_ROMAN_CROCKETT | 1000118 | Helin-Roman-Crockett |
| NAIFID_HARTLEY_3 | 1000119 | Hartley 3 |
| NAIFID_PARKER_HARTLEY | 1000120 | Parker-Hartley |
| NAIFID_HELIN_ROMAN_ALU_1 | 1000121 | Helin-Roman-Alu 1 |
| NAIFID_WILD_4 | 1000122 | Wild 4 |
| NAIFID_MUELLER_2 | 1000123 | Mueller 2 |
| NAIFID_MUELLER_3 | 1000124 | Mueller 3 |
| NAIFID_SHOEMAKER_LEVY_1 | 1000125 | Shoemaker-Levy 1 |
| NAIFID_SHOEMAKER_LEVY_2 | 1000126 | Shoemaker-Levy 2 |
| NAIFID_HOLT_OLMSTEAD | 1000127 | Holt-Olmstead |
| NAIFID_METCALF_BREWINGTON | 1000128 | Metcalf-Brewington |
| NAIFID_LEVY | 1000129 | Levy |
| NAIFID_SHOEMAKER_LEVY_9 | 1000130 | Shoemaker-Levy 9 |
| NAIFID_HYAKUTAKE | 1000131 | Hyakutake |
| NAIFID_HALE_BOPP | 1000132 | Hale-Bopp |
| NAIFID_SIDING_SPRING | 1003228 | Siding Spring |

RELEASE NOTES

- **Version 3.5.4**

Fix the documentation about the unsupported segment 19.
Fix the window size if requested time is near a boundary for SPICE kernel files containing segments of type 18, compliant with the SPICE specifications.
Fix incorrect results for SPICE kernel files containing segments of type 20.
Support the segment 14 in the SPICE kernel file.

- **Version 3.5.3**

Fix compilation error with Cython 3.0 or later.
Fix compilation error with Python 3.10 or later on some linux distributions.
Fix the tests suite with octave, due to a change of the syntax parsing by octave.

- **Version 3.5.2**

Fix a possible buffer overflow in the python interface of getfileversion.
Fix the undefined external symbol rint with visual C compiler 10.0.
Fix the redefinition of vasprintf with the environment MinGW.

- **Version 3.5.1**

Support the incremental assignment in the text SPICE kernel file (files .tpc).
Support the relative frames in the frame SPICE kernel file (files .tf).
Fix the returned values of calceph_getconstantvs with large SPICE kernels.
Fix random crash or double-free error when meta-kernel SPICE kernel files are closed (files .mk).
Raise an error inside the function calceph_compute if center != 0 for the target (14,15,16,17), to conform to the documentation.
Fix the object name Megaclite (519) from NAIFID_MAGACLITE to NAIFID_MEGACLITE.

- **Version 3.5.0**

Add the function calceph_getmaxsupportedorder, calceph_getpositionrecordindex2 and calceph_getorientrecordindex2.
Update config.sub and config.guess to support new processors.
Fix an incorrect result of calceph_(s)getconstant and calceph_getconstantvd with SPICE text kernel if the current locale has a decimal separator different than the decimal point (e.g. comma).
Fix compilation warnings with gcc 11 about pointers and arrays.
Fix the installation with python 3.10 or later (PEP 632).
Fix compilation errors with the android SDK API level < 24.
Reduce the stack size of the arrays for the evaluation with C99 compilers.

Change the array size of the function `calceph_getfileversion` from the value `CALCEPH_MAX_CONSTANTNAME`

to `CALCEPH_MAX_CONSTANTVALUE`, to conform to the documentation and examples.

Only C and Fortran 2003 are affected by this change : previous header file contains an invalid declaration of the function.

Few programs may be affected by this change, as the required array has a smaller size : No buffer overflow may appear.

- **Version 3.4.7**

Fix a decode error of the little-endian SPICE kernel files on the big-endian architectures (e.g. processor s390x).

Fix the transmission of the flags `FCFLAGS` to the fortran compilers.

- **Version 3.4.6**

Fix a wrong error message about unsupported order for the segment 21.

Fix incorrect results for SPICE kernel files containing segments of type 21 with many records (≥ 100) and improved the accuracy if segments of type 21 contain few records (< 100).

- **Version 3.4.5**

Fix a random crash of `calceph_open_array` if one of the file is invalid.

`f90calceph_seterrorhandler` now ignores the parameter `userfunc`, instead of the requirement to set to 0, if the parameter type is 1 or 2. `userfunc` can be an empty function. It fixes compilation errors with gcc 10.1.

- **Version 3.4.4**

Fix a regression introduced in 3.4.3 (remove a recursion with SPICE kernel files).

- **Version 3.4.3**

Remove a recursion to read the segments of the SPICE kernel files. It reduces the usage of the stack.

Fix the installation of python package under Anaconda.

- **Version 3.4.2**

Add a missing makefile for windows system using the Visual C++ compiler.

Support SPICE kernels larger than 4GBytes.

- **Version 3.4.1**

Improve the execution time of `calceph_open` and `calceph_open_array` if the spice kernels contains a large number of bodies.

Update `config.sub` and `config.guess` to support arm processors.

- **Version 3.4.0**

Add the function `calceph_isthreadsafe`.

Multiple threads can now access the same ephemeris descriptor if the function `calceph_isthreadsafe` returns 1.

Fortran and C examples (`f2003parallel.f`, `cparallel.c`), written using OpenMP, are available in the folder `examples`.

Fix an error if multiple SPICE kernels are loaded for the same objects over different time-span.

Fix the MinGW Makefiles if the variable `MAKE` contains spaces.

Support the segment 5 and 18 in the SPICE kernel file.

Support the euler angles for the orientation stored in a text PCK files (`BODY..._POLE_RA`, `BODY..._POLE_DE`, `BODY..._POLE_PM`, `BODY..._NUT_PREC...`).

Support the frame 17 (ECLIPJ2000) in the SPICE kernel file.

Add the utilities `calceph_queryposition` and `calceph_queryorientation`.

- **Version 3.3.1**

Fix the installation with python 3.7.0 or later.

Fix the installation with python and pip on Windows operating system.

Add the missing file `pythonapi/src/Makefile.mingw` for the environment MinGW.

- **Version 3.3.0**

Add the functions `calceph_getfileversion`.

Fix a regression to open some old JPL DE format files.

Fix a compiler warning in the file `util.c`.

Support the segments 8, 9, 17 and 21 in the SPICE kernel file.

Check the validity of the number of constants in the original INPOP/DE files.

For the Python interface, the functions `compute???` and `orient???` supports now a list or numpy's array for the time parameters.

- **Version 3.2.0**

Fix the creation of the dynamic library with `msys/mingw` on Windows.

Fix the returned value of the functions `f90calceph_getconstantvd` and `f90calceph_getconstantvs`.

Fix a compilation warning with the GNU C compilers 8.0 or later.

Support the original JPL files with TT-TDB or with a large number of constants.

Support the IAU 1980 Nutation Angles of the JPL files.

Add the NAIF identification numbers for DIA, KERBEROS, STYX and SIDING SPRING.

Add the option `installnodoc` to the make command.

- **Version 3.1.0**

Add the Mex interface compliant with Octave 4.0+ and Matlab 2017+.

Add the functions `calceph_getconstantsd`, `calceph_getconstantvd` and `calceph_getconstantss` and `calceph_getconstantvs`.

Fix a compilation problem with MinGW if the terminal `cmd.exe` is used.

Fix a wrong function name `open_array` instead of `open` in the documentation of the Python interface.

Fix the return value of the functions `calceph_orient_XXX` when the unit `CALCEPH_UNIT_RAD` is not provided.

The return value of the function `calceph_(s)getconstant(index)` is the number of values associated to the constant.

Display a better message for the unsupported old spice kernel (NAIF/DAF)

- **Version 3.0.0**

Update the license CeCILL v2.0 to CeCILL v2.1.

Fix a decode error for SPICE kernels with a big-endian format.

Add the function `calceph_gettimescale` and `calceph_gettimespan`.

Add the function `calceph_getpositionrecordcount` and `calceph_getpositionrecordindex`.

Add the function `calceph_getorientrecordcount` and `calceph_getorientrecordindex`.

Add the function `calceph_sgettimescale` and `calceph_sgettimespan`.

Support INPOP file format 3.0 (add angular momentum due to the rotation in the binary file).

Use `sphinx-doc` to produce the documentation.

- **Version 2.3.2**

Fix the return value of the function `calceph_getconstant` if the constant name "AU" or "EMRAT" is not available.

Fix the documentation for the fortran interface of the function `calceph_prefetch`.

Fix the return value of the function `calceph_orient_unit` if the frame SPICE kernel file is missing.

- **Version 2.3.1**

Fix the compilation warnings with the Pelles compiler.

Fix the compilation warnings with the C89 standard.

Fix the compilation warnings with the GNU C compilers.

Fix the documentation for the constant `CALCEPH_VERSION_STRING`.

- **Version 2.3.0**

Add the python interface compliant with python 2.6+ and python 3.

Add the preprocessor macro `CALCEPH_VERSION_STRING`.

Add the function `calceph_getversion_str`.

Add the function `calceph_compute_order` and `calceph_orient_order`.

Fix the return value of the functions `calceph_compute_xxx` when the reference frame is not available in the spice kernel files.

The function should produce an error and return 0 (before the function performed no computation but it returned 1).

- **Version 2.2.5**

Fix an incorrect result if `CALCEPH_UNIT_DAY` is provided to `calceph_compute_unit` and the target is TCG-TCB or TT-TDB.

Support the numerical constants declared without parenthesis in the text kernel files (.tpc).

Support the segment 1, 12 and 13 in the SPICE kernel file.

- **Version 2.2.4**

Update the version number of the dynamic library.

- **Version 2.2.3**

Add the predefined constants for calceph version in the fortran interface.

Fix the build chain if calceph is compiled from another folder.

- **Version 2.2.2**

Support the compilation in the standard C89.

- **Version 2.2.1**

Remove debug informations that are printed when errors occur in `calceph_?compute_???`.

Support the Portland compilers.

Fix the info documentation.

Report an error if no asteroid is available in an ephemeris file with the INPOP file format (instead of a crash).

- **Version 2.2.0**

Support the new segments 20, 102, 103 and 120 in the SPICE kernel file.

Support the NAIF identification numbers.

Add the functions `calceph_orient_unit` and `calceph_prefetch`.

- **Version 2.1.0**

Fix a bug in `calceph_getconstant` and `calceph_sgetconstant` with an invalid name

Remove the null character in the name of the constant returned by the function

`(f90)calceph(s)getconstantindex` when the Fortran interface is used.

- **Version 2.0.0**

- Fix memory leaks in calceph_open when errors occur.
- Support INPOP file format 2.0 (supports TCB ephemeris file and add asteroids in the binary file).
- Add the function calceph_open_array and calceph_compute_unit.
- Add the tools calceph_inspector to show details about ephemeris file.
- Support SPICE kernel file (SPK with segment 2 or 3, text and binary PCK, meta kernel, basic frame kernel).
- Improve the performances.
- Correct the Fortran 2003 interface for calceph_sgetconstantindex.
- Add the constant 17 to get TCG-TCB from TCB ephemeris file.

- **Version 1.2.0**

- Change the licensing : triple licenses to support integration in BSD software.
- Remove explicit dependencies on the record size for DExxx.

- **Version 1.1.2**

- Fix a compilation warning with oracle studio compiler 12.
- Fix a bug with gcc on solaris in 64 bit mode.
- Fix the copyright statements.

- **Version 1.1.1**

- Fix a compilation error in util.h and a warning with the sun studio compilers.

- **Version 1.1.0**

- Add the function calceph_seterrorhandler for the custom error handlers.

- **Version 1.0.3**

- Support the JPL ephemeris file DE423.

- **Version 1.0.2**

- Fix memory leaks in the fortran-90 interface.

- **Version 1.0.1**

- Support the large ephemeris files (>2GB) on 32-bit operating systems.
- Fix the documentation of the function f90calceph_sopen.
- Fix an invalid open mode on Windows operating systems.
- Report accurately the I/O errors.

- **Version 1.0.0**

- Initial release.

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@obspm.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.

CALCEPH LIBRARY COPYING CONDITIONS

Copyright 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023

CNRS, Observatoire de Paris, Observatoire de la Côte d'Azur

Contributed by

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

Fienga A. , Observatoire de la Côte d'Azur

inpop.imcce@obspm.fr

This library is governed by the CeCILL-C, CeCILL-B 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, CeCILL-B 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, CeCILL-B or CeCILL version 2.1 license and that you accept its terms.

C

CALCEPH_ ASTEROID (*fortran variable*), **13**
 CALCEPH_MAX_CONSTANTNAME (*fortran variable*),
12
 CALCEPH_MAX_CONSTANTVALUE (*fortran variable*),
12
 CALCEPH_OUTPUT_EULERANGLES (*fortran variable*), **13**
 CALCEPH_OUTPUT_NUTATIONANGLES (*fortran variable*), **13**
 CALCEPH_SEGTYPE_ORIG_0 (*fortran variable*), **13**
 CALCEPH_SEGTYPE_SPK_1 (*fortran variable*), **13**
 CALCEPH_SEGTYPE_SPK_102 (*fortran variable*), **14**
 CALCEPH_SEGTYPE_SPK_103 (*fortran variable*), **14**
 CALCEPH_SEGTYPE_SPK_12 (*fortran variable*), **13**
 CALCEPH_SEGTYPE_SPK_120 (*fortran variable*), **14**
 CALCEPH_SEGTYPE_SPK_13 (*fortran variable*), **13**
 CALCEPH_SEGTYPE_SPK_14 (*fortran variable*), **13**
 CALCEPH_SEGTYPE_SPK_17 (*fortran variable*), **14**
 CALCEPH_SEGTYPE_SPK_18 (*fortran variable*), **14**
 CALCEPH_SEGTYPE_SPK_19 (*fortran variable*), **14**
 CALCEPH_SEGTYPE_SPK_2 (*fortran variable*), **13**
 CALCEPH_SEGTYPE_SPK_20 (*fortran variable*), **14**
 CALCEPH_SEGTYPE_SPK_21 (*fortran variable*), **14**
 CALCEPH_SEGTYPE_SPK_3 (*fortran variable*), **13**
 CALCEPH_SEGTYPE_SPK_5 (*fortran variable*), **13**
 CALCEPH_SEGTYPE_SPK_8 (*fortran variable*), **13**
 CALCEPH_SEGTYPE_SPK_9 (*fortran variable*), **13**
 CALCEPH_UNIT_AU (*fortran variable*), **13**
 CALCEPH_UNIT_DAY (*fortran variable*), **13**
 CALCEPH_UNIT_KM (*fortran variable*), **13**
 CALCEPH_UNIT_RAD (*fortran variable*), **13**
 CALCEPH_UNIT_SEC (*fortran variable*), **13**
 CALCEPH_USE_NAIFID (*fortran variable*), **13**
 CALCEPH_VERSION_MAJOR (*fortran variable*), **12**
 CALCEPH_VERSION_MINOR (*fortran variable*), **12**
 CALCEPH_VERSION_PATCH (*fortran variable*), **12**
 CALCEPH_VERSION_STRING (*fortran variable*), **12**

F

f90calceph_close() (*fortran subroutine*), **41**
 f90calceph_compute() (*fortran function*), **18**

f90calceph_compute_order() (*fortran function*), **25**
 f90calceph_compute_unit() (*fortran function*),
20
 f90calceph_getconstant() (*fortran function*),
30
 f90calceph_getconstantcount() (*fortran function*), **34**
 f90calceph_getconstantindex() (*fortran function*), **34**
 f90calceph_getconstantsd() (*fortran function*), **31**
 f90calceph_getconstantss() (*fortran function*), **32**
 f90calceph_getconstantvd() (*fortran function*), **32**
 f90calceph_getconstantvs() (*fortran function*), **33**
 f90calceph_getfileversion() (*fortran function*), **35**
 f90calceph_getmaxsupportedorder() (*fortran subroutine*), **57**
 f90calceph_getorientrecordcount() (*fortran function*), **39**
 f90calceph_getpositionrecordcount() (*fortran function*), **37**
 f90calceph_getpositionrecordindex() (*fortran function*), **37, 40**
 f90calceph_getpositionrecordindex2() (*fortran function*), **38, 40**
 f90calceph_gettimescale() (*fortran function*),
36
 f90calceph_gettimespan() (*fortran function*),
36
 f90calceph_getversion_str() (*fortran subroutine*), **57**
 f90calceph_isthreadsafe() (*fortran function*),
18
 f90calceph_open() (*fortran function*), **16**
 f90calceph_open_array() (*fortran function*), **17**
 f90calceph_orient_order() (*fortran function*),
27

f90calceph_orient_unit() (*fortran function*),
[22](#)
f90calceph_prefetch() (*fortran function*), [18](#)
f90calceph_rotangmom_order() (*fortran function*), [29](#)
f90calceph_rotangmom_unit() (*fortran function*), [23](#)
f90calceph_sclose() (*fortran subroutine*), [51](#)
f90calceph_scompute() (*fortran function*), [46](#)
f90calceph_seterrorhandler() (*fortran subroutine*), [54](#)
f90calceph_sgetconstant() (*fortran function*),
[48](#)
f90calceph_sgetconstantcount() (*fortran function*), [48](#)
f90calceph_sgetconstantindex() (*fortran function*), [49](#)
f90calceph_sgetfileversion() (*fortran function*), [49](#)
f90calceph_sgettimescale() (*fortran function*), [50](#)
f90calceph_sgettimespan() (*fortran function*),
[50](#)
f90calceph_sopen() (*fortran function*), [45](#)