CALCEPH Library

Reference manual
version 2.3.2
10 January 2017

M. Gastineau, J. Laskar, A. Fienga, H. Manche
inpop.imcce@obspm.fr

This manual documents how to install and use the CALCEPH Library, version 2.3.2.

Contributed by

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

A. Fienga, Observatoire de la Côte d'Azur

`inpop.imcce@obspm.fr`

Table of Contents

1 CALCEPH Library Copying conditions

Contributed by

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

A. Fienga, Observatoire de la Côte d'Azur

`inpop.imcce@obspm.fr`

2 Introduction to CALCEPH Library

This 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 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, 12, 13, 20, 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, optionnally, the Fortran 77 or 2003 and Python interfaces, an to be called by the application.

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

- Single file access functions

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

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

This library could access to the following ephemeris

- INPOP06 or later
- DE200
- DE403 or later

Although computers have different endianess (order in which integers are stored as bytes in computer memory), the library could handle the binary ephemeris files with any endianess. 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`

3 Installing CALCEPH Library

3.1 Quick instructions for installing on a Unix-like system (Linux, Mac OS X, BSD, cygwin, ...)

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 gcc and gfortran compilers, the steps are :
  1. tar xzf calceph-2.3.2.tar.gz
  2. cd calceph-2.3.2
  3. ./configure –disable-shared CC=gcc FC=gfortran –prefix=/home/mylogin/mydir
  4. make check && make install
- If you use the Intel c++ and fortran compilers, the steps are :
  1. tar xzf calceph-2.3.2.tar.gz
  2. cd calceph-2.3.2
  3. ./configure –disable-shared CC=icc FC=ifort –prefix=/home/mylogin/mydir
  4. make check && make install
- If you use the python interface of the library, the steps are :
  1. tar xzf calceph-2.3.2.tar.gz
  2. cd calceph-2.3.2
  3. ./configure –enable-python=yes –enable-python-package-user=yes –prefix=/home/mylogin/mydir
  4. make check && make install

3.2 Detailed instructions for installing on a Unix-like system (Linux, Mac OS X, BSD, cygwin, ...)

You need a C compiler, such as gcc. A fortran compiler, compliant with the ANSI Fortran 77 specifications, is required to compile the fortran-77/90/95 interface of the library. A fortran compiler, compliant with the Fortran 2003 specifications, is required to compile the fortran-2003 interface of the library. A python interpreter, compliant at least with with the Python 2.6 or Python 3.0 specifications, is required to compile the python 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:

1. '`tar xzf calceph-2.3.2.tar.gz`'
2. '`cd calceph-2.3.2`'
3. '`./configure`'

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

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

`--enable-fortran={yes|no}`
: Enable or disable the fortran-77 and fortran-2003 interface. The default is '`yes`'.

`--enable-python={yes|no}`
: Enable or disable the python interface. The default is '`no`'.

`--enable-python-package-system={yes|no}`
: Enable or disable the installation of the python package to the system site-packages directory (e.g., /usr/lib/python3.4/sites-packages/) . The default is '`no`'.

`--enable-python-package-user={yes|no}`
: Enable or disable the installation of the python package to the user site-packages directory (e.g., ~/.local/lib/python3.4/site-packages/) . The default is '`no`'.

`--enable-thread={yes|no}`
: Enable or disable the thread-safe version of the functions `calceph_sopen` and `calceph_scompute`. The default is '`no`'.

`--disable-shared`
: Disable shared library.

`--disable-static`
: Disable static library.

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

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

The default compilers could be changed using the variable CC for C compiler, FC for the Fortran compiler and PYTHON for the python interpreter. The default compiler flags could be changed using the variable CFLAGS for C compiler and FCFLAGS for the Fortran compiler.

If `--enable-python=yes`, we recommend to set `--enable-python-package-user=yes` (or `--enable-python-package-system=yes` if you have administrative right on the system directory) in order to that the python interpreter finds the CALCEPH python package.

4. '`make`'

   This compiles the CALCEPH Library in the working directory.

5. '`make check`'

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

   If you get error messages, please report them to `inpop.imcce@obspm.fr` (See Chapter 4 [Reporting bugs], page 9, for information on what to include in useful bug reports).

6. '`make install`'

   This will copy the file `calceph.h`, `calceph.mod` and `f90calceph.h` to the directory `/usr/local/include`, the file `libcalceph.a`, `libcalceph.so` to the directory

`/usr/local/lib`, and the file `calceph.info` to the directory `/usr/local/share/info` (or if you passed the '`--prefix`' option to `configure`, using the prefix directory given as argument to '`--prefix`' instead of `/usr/local`). Note: you need write permissions on these directories.

If the python interface is enabled and `enable-python-package-system=yes` or `enable-python-package-user=yes`, the python package will be copied to system or user python site-package.

3.2.1 Other 'make' Targets

There are some other useful make targets:

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

3.3 Installation on Windows system

3.3.1 Using the Windows SDK

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

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

1. Expand the file '`calceph-2.3.2.tar.gz`'
2. Execute the command '`cmd.exe`' from the menu '`Start`' / '`Execute...`'
   This will open a console window
3. '`cd` '*dir*'`\calceph-2.3.2`'
   Go to the directory *dir* where CALCEPH Library has been expanded.

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

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

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

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

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

   If you get error messages, please report them to `inpop.imcce@obspm.fr` (See Chapter 4 [Reporting bugs], page 9, 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 defaut value is '`cl.exe`'
   - FC=*xx* specifies the name of the Fortran compiler. The defaut 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 defaut value is '`0`'.
   - ENABLEF77={0|1} specifies if it must compile the fortran 77/90/95 interface. The defaut value is '`0`'.

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

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

   This command line accepts several options :

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

3.3.2 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. A python interpreter, compliant at least with with the Python 2.6 or Python 3.0 specifications, is required to compile the python interface of the library.

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

1. Expand the file '`calceph-2.3.2.tar.gz`'
2. Execute the command '`MinGW Shell`' from the menu '`Start`'.

   This will open a MinGW Shell console window.
3. '`cd` '*dir*'`\calceph-2.3.2`'

   Go to the directory *dir* where CALCEPH Library has been expanded.
4. '`make -f Makefile.mingw` '

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

   - CC=*xx* specifies the name of the C compiler. The defaut value is '`gcc.exe`'
   - FC=*xx* specifies the name of the Fortran compiler. The defaut value is '`gfortran.exe`'
   - PYTHON=*xx* specifies the name of the Python interpreter. The defaut value is '`python.exe`'
   - ENABLEF2003={0|1} specifies if it must compile the fortran 2003 interface. The defaut value is '`0`'.
   - ENABLEF77={0|1} specifies if it must compile the fortran 77/90/95 interface. The defaut value is '`0`'.
   - ENABLEPYTHON={0|1} specifies if it must compile the python interface. The defaut value is '`0`'.
5. '`make -f Makefile.mingw check`'

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

   If you get error messages, please report them to `inpop.imcce@obspm.fr` (See Chapter 4 [Reporting bugs], page 9, 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 defaut value is '`gcc.exe`'
   - FC=*xx* specifies the name of the Fortran compiler. The defaut value is '`gfortran.exe`'
   - PYTHON=*xx* specifies the name of the Python interpreter. The defaut value is '`python.exe`'
   - ENABLEF2003={0|1} specifies if it must compile and check the fortran 2003 interface. The defaut value is '`0`'.
   - ENABLEF77={0|1} specifies if it must compile and check the fortran 77/90/95 interface. The defaut value is '`0`'.

- ENABLEPYTHON={0|1} specifies if it must compile and check the python interface. The defaut value is '0'.

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

   This will copy the file `calceph.h`, `calceph.mod` and `f90calceph.h` to the directory *dir*, the file `libcalceph.lib` to the directory *dir*`\lib`, the file `calceph.pdf` to the directory *dir*`\doc`.

   If `ENABLEPYTHON=1`, the installation will copy the of the CALCEPH python package to the system python site package (e.g., C:\Python27\Lib\sites-packages\) in order to that the python interpreter finds the CALCEPH module.

   Note: you need write permissions on these directories.

   This command line accepts several options :

   - CC=*xx* specifies the name of the C compiler. The defaut value is '`gcc.exe`'
   - FC=*xx* specifies the name of the Fortran compiler. The defaut value is '`gfortran.exe`'
   - PYTHON=*xx* specifies the name of the Python interpreter. The defaut value is '`python.exe`'
   - ENABLEF2003={0|1} specifies if it must compile and install the fortran 2003 interface. The defaut value is '0'.
   - ENABLEF77={0|1} specifies if it must compile and install the fortran 77/90/95 interface. The defaut value is '0'.
   - ENABLEPYTHON={0|1} specifies if it must compile and install the python interface. The defaut value is '0'.

4 Reporting bugs

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

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

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

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

Send your bug report to: `inpop.imcce@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.

5 CALCEPH Library Interface

5.1 C Usage

5.1.1 Headers and Libraries

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

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

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

5.1.1.1 Compilation on a Unix-like system

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

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

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

5.1.1.2 Compilation on a Windows system

Using the Windows SDK

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

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

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

Using the MinGW

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

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

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

5.1.2 Constants

CALCEPH_MAX_CONSTANTNAME

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

CALCEPH_VERSION_MAJOR

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

CALCEPH_VERSION_MINOR

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

CALCEPH_VERSION_PATCH

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

```
#if   (CALCEPH_VERSION_MAJOR>=2)
  ||  (CALCEPH_VERSION_MAJOR>=3 && CALCEPH_VERSION_MINOR>=2)
...
#endif
```

CALCEPH_VERSION_STRING

This C null-terminated string constant 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:

```
char version[CALCEPH_MAX_CONSTANTNAME];
calceph_getversion_str(version);
if (strcmp (version, CALCEPH_VERSION_STRING)!=0)
  fprintf (stderr, "Warning: header and library do not match\n");
```

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

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

The following constants specify in which units are expressed the output of the computation functions, such as calceph_compute_unit :

- CALCEPH_UNIT_AU This integer defines that the unit of the positions and velocities is expressed in astronomical unit.
- CALCEPH_UNIT_KM This integer defines that the unit of the positions and velocities is expressed in kilometer.
- CALCEPH_UNIT_DAY 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 This integer defines that the unit of the velocities or the quantity TT-TDB or TCG-TCB is expressed in second.
- CALCEPH_UNIT_RAD This integer defines that the unit of the angles is expressed in radian.

CALCEPH_USE_NAIFID

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

5.1.3 Types

`t_calcephbin` [Data type]

This type contains all information to access an ephemeris file.

5.2 Fortran 2003 Usage

5.2.1 Modules and Libraries

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

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

```
use calceph
```

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

5.2.2 Compilation on a Unix-like system

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

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

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

5.2.3 Compilation on a Windows system

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

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

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

5.2.4 Constants

The following constants are defined in the module `calceph.mod`.

CALCEPH_MAX_CONSTANTNAME

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

CALCEPH_VERSION_MAJOR

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

CALCEPH_VERSION_MINOR

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

CALCEPH_VERSION_PATCH

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

CALCEPH_VERSION_STRING

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

CALCEPH_ASTEROID

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

The following constants specify in which units are expressed the output of the computation functions, such as calceph_compute_unit :

- CALCEPH_UNIT_AU This integer defines that the unit of the positions and velocities is expressed in astronomical unit.
- CALCEPH_UNIT_KM This integer defines that the unit of the positions and velocities is expressed in kilometer.
- CALCEPH_UNIT_DAY 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 This integer defines that the unit of the velocities or the quantity TT-TDB or TCG-TCB is expressed in second.
- CALCEPH_UNIT_RAD This integer defines that the unit of the angles is expressed in radian.

CALCEPH_USE_NAIFID

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

5.3 Fortran 77/90/95 Usage

5.3.1 Headers and Libraries

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

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

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

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

5.3.2 Compilation on a Unix-like system

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

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

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

5.3.3 Compilation on a Windows system

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

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

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

5.3.4 Constants

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

CALCEPH_MAX_CONSTANTNAME

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

CALCEPH_VERSION_MAJOR

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

CALCEPH_VERSION_MINOR

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

CALCEPH_VERSION_PATCH

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

CALCEPH_VERSION_STRING

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

CALCEPH_ASTEROID

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

The following constants specify in which units are expressed the output of the computation functions, such as calceph_compute_unit :

- CALCEPH_UNIT_AU This integer defines that the unit of the positions and velocities is expressed in astronomical unit.
- CALCEPH_UNIT_KM This integer defines that the unit of the positions and velocities is expressed in kilometer.
- CALCEPH_UNIT_DAY 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 This integer defines that the unit of the velocities or the quantity TT-TDB or TCG-TCB is expressed in second.

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

CALCEPH_USE_NAIFID

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

5.4 Python 2/3 Usage

5.4.1 Modules

It is designed to work with Python interpreters compliant with the Python 2.6 or later and Python 3.0 or later.

All declarations needed to use CALCEPH Library are collected in the module `calcephpy`. You should import this module:

```
from calcephpy import *
```

If you receive the following message "ImportError: No module named calcephpy" and if the configuration option `enable-python-package-system` and `enable-python-package-user` was not set, the environment variable *PYTHONPATH* should be set to the right location of the CALCEPH python package (e.g., PYTHONPATH=/usr/local/lib64/python3.4/site-packages/:$PYTHONPATH) in your shell initialization file (e.g., ~/.bash_login or ~/.profile), in order that the python interpreter finds the CALCEPH python package.

Relative to C or Fortran interface, the prefixes `calceph_`, `CALCEPH_`, `NAIFID_` are deleted for the naming convention of the functions, constants and NAIF identification numbers.

5.4.2 Types

`calcephpy.CalcephBin` [Data type]
: This type contains all information to access an ephemeris file.

`calcephpy.NaifId` [Data type]
: This type contains the NAIF identification numbers.

`calcephpy.Constants` [Data type]
: This type contains all constants defined in the library, except the NAIF identification numbers.

5.4.3 Constants

The following constants are defined in the class `Constants` (or `calcephpy.Constants`).

VERSION_MAJOR

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

VERSION_MINOR

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

VERSION_PATCH
This integer constant defines the patch level revision of this library. It can be used to distinguish different releases of this library.
VERSION_STRING
This string constant is the version of the library, which can be compared to the result of calceph_getversion_str to check at run time if the header file and library used match.
ASTEROID
This integer defines the offset value for the asteroids that must be used as target or center for the computation functions, such as calceph_compute.
The following constants specify in which units are expressed the output of the computation functions, such as calceph_compute_unit :

- UNIT_AU This integer defines that the unit of the positions and velocities is expressed in astronomical unit.
- UNIT_KM This integer defines that the unit of the positions and velocities is expressed in kilometer.
- UNIT_DAY 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).
- UNIT_SEC This integer defines that the unit of the velocities or the quantity TT-TDB or TCG-TCB is expressed in second.
- UNIT_RAD This integer defines that the unit of the angles is expressed in radian.

USE_NAIFID
This integer defines that the NAIF identification numbers are used as target or center for the computation functions, such as calceph_compute_unit.

5.4.4 A simple example program

The following example program shows the typical usage of the python interface.
Other examples using the python interface can be found in the directory '`examples`' of the library sources.

```
from calcephpy import *
peph = CalcephBin.open("example1.dat")
AU = peph.getconstant("AU")
PV = peph.compute_unit(jd0, dt, NaifId.MOON, NaifId.EARTH,
           Constants.UNIT_KM+Constants.UNIT_SEC+Constants.USE_NAIFID)
print(PV)
peph.close()
```

5.5 Multiple file access functions

The following group of functions should be the preferred method to access to the library. They allow to access to multiple ephemeris files at the same time, even by multiple threads.
When an error occurs, these functions execute error handlers according to the behavior defined by the function `calceph_seterrorhandler` (see Section 5.7 [Error functions], page 47).

5.5.1 Thread notes

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

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

5.5.2 Usage

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

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

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

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

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

5.5.3 Functions

5.5.3.1 calceph_open

`t_calcephbin* calceph_open ( const char *filename )` [C]

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

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

`eph = calcephpy.CalcephBin.open (filename)` [Python 2/3]
<type 'calcephpy.CalcephBin'> eph
<type 'str'> filename

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, 12, 13, 20, 102, 103 and 120.
- meta kernel (KPL/MK) files.
- frame kernel (KPL/FK) files. Only a basic support is provided.

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

Just after the call of `calceph_open`, the function `calceph_prefetch` should be called to accelerate future computations.

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

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

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

5.5.3.2 calceph_open_array

`t_calcephbin* calceph_open_array` (*int* ***n***, *const char* ****array_filename***[]) [C]

`function calceph_open_array` (***n***, ***array_filename***, ***len_filename***) *BIND*(*C*) [Fortran 2003]

```
INTEGER(C_INT), VALUE, intent(in) :: n
CHARACTER(len=1,kind=C_CHAR), dimension(*), intent(in) :: array_filename
INTEGER(C_INT), VALUE, intent(in) :: len_filename
TYPE(C_PTR) :: calceph_open_array
```

`function f90calceph_open_array` (***eph***, ***n***, ***array_filename***, ***len_filename***) [Fortran 77/90/95]

```
INTEGER, intent(in) :: n
CHARACTER(len=*), dimension(*), intent(in) :: filename
INTEGER, intent(in) :: len_filename
INTEGER(8), intent(out) :: eph
INTEGER :: f90calceph_open
```

eph = `calcephpy.CalcephBin.open` (***array_filename***) [Python 2/3]

```
<type 'calcephpy.CalcephBin'> eph
<type 'list'> array_filename
```

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 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, 12, 13, 20, 102, 103 and 120.
- meta kernel (KPL/MK) files.
- frame kernel (KPL/FK) files. Only a basic support is provided.

With the Fortran interfaces, len_filename specifies the number of character of each file's name.

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

Just after the call of `calceph_open_array`, the function `calceph_prefetch` should be called to accelerate future computations.

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

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

```
TYPE(C_PTR) :: peph
character(len=256), dimension (2) :: filear
filear(1) = "../examples/example1.bsp"//C_NULL_CHAR
filear(2) = "../examples/example1.tpc"//C_NULL_CHAR
peph = calceph_open_array(2, filear, 256)
if (C_ASSOCIATED(peph)) then
   calceph_prefetch(peph1);
   calceph_prefetch(peph2);
 ! ...  computation ...
 call calceph_close(peph)
endif
```

5.5.3.3 calceph_prefetch

`int calceph_prefetch ( t_calcephbin* eph )` [C]

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

`function f90calceph_prefetch (eph)` [Fortran 77/90/95]
INTEGER(8), intent(in) :: eph
INTEGER :: f90calceph_prefetch

`eph.prefetch ()` [Python 2/3]
<type 'calcephpy.CalcephBin'> eph

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 `calceph_compute`, `calcephcompute_unit` or `calceph_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 endianess 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.

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

5.5.3.4 calceph_compute

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

`function calceph_compute (`*eph*, *JD0*, *time*, *target*, *center*, *PV*) *BIND*(*C*) [Fortran 2003]

```
TYPE(C_PTR), VALUE, intent(in) :: eph
REAL(C_DOUBLE), VALUE, intent(in) :: JD0
REAL(C_DOUBLE), VALUE, intent(in) :: time
INTEGER(C_INT), VALUE, intent(in) :: target
INTEGER(C_INT), VALUE, intent(in) :: center
REAL(C_DOUBLE), intent(out) :: PV(6)
INTEGER(C_INT) :: calceph_compute
```

`function f90calceph_compute (`*eph*, *JD0*, *time*, *target*, *center*, *PV*) [Fortran 77/90/95]

```
INTEGER(8), intent(in) :: eph
REAL(8), intent(in) :: JD0
REAL(8), intent(in) :: time
INTEGER, intent(in) :: target
INTEGER, intent(in) :: center
REAL(8), intent(out) :: PV(6)
INTEGER :: f90calceph_compute
```

PV = *eph*`.compute (`*JD0*, *time*, *target*, *center*`)` [Python 2/3]

```
<type 'list'> PV
<type 'calcephpy.CalcephBin'> eph
<type 'float'> JD0
<type 'float'> time
<type 'int'> target
<type 'int'> center
```

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

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

The arguments are :

JD0 Integer part of the Julian Date.

time Fraction part of the Julian Date.

target The body or reference point whose coordinates are required (see the list, below).

center The origin of the coordinate system (see the list, below). If *target* is 15, 16 or 17(libration, TT-TDB or TCG-TCB), *center* must be '0'.

PV An array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot).

The position is expressed in Astronomical Unit (au) and the velocity is expressed in Astronomical Unit per day (au/day).

If the target is *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 angles of the librations of the Moon are expressed in radians and their derivatives are expressed in radians per day.

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
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

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

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

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

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

5.5.3.5 calceph_compute_unit

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

`function calceph_compute_unit` (***eph***, ***JD0***, **time**, ***target***, ***center***, **unit**, ***PV***) *BIND*(*C*) [Fortran 2003]

```
TYPE(C_PTR), VALUE, intent(in) :: eph
REAL(C_DOUBLE), VALUE, intent(in) :: JD0
REAL(C_DOUBLE), VALUE, intent(in) :: time
INTEGER(C_INT), VALUE, intent(in) :: target
INTEGER(C_INT), VALUE, intent(in) :: center
INTEGER(C_INT), VALUE, intent(in) :: unit
REAL(C_DOUBLE), intent(out) :: PV(6)
INTEGER(C_INT) :: calceph_compute_unit
```

`function f90calceph_compute_unit` (***eph***, ***JD0***, **time**, ***target***, ***center***, **unit**, ***PV***) [Fortran 77/90/95]

```
INTEGER(8), intent(in) :: eph
REAL(8), intent(in) :: JD0
REAL(8), intent(in) :: time
INTEGER, intent(in) :: target
```

```
INTEGER, intent(in) :: center
INTEGER, intent(in) :: unit
REAL(8), intent(out) :: PV(6)
INTEGER :: f90calceph_compute_unit
```

PV = *eph*.compute_unit (*JD0*, *time*, *target*, *center*, *unit*) [Python 2/3]

```
<type 'list'> PV
<type 'calcephpy.CalcephBin'> eph
<type 'float'> JD0
<type 'float'> time
<type 'int'> target
<type 'int'> center
<type 'int'> unit
```

This function is similar to the function `calceph_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*), from the ephemeris file, previously opened with the function `calceph_open`, for the time *JD0+time* and stores the results to *PV*. 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.

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

The arguments are :

JD0 Integer part of the Julian Date.

time Fraction part of the Julian Date.

target The body or reference point whose coordinates are required. The numbering system depends on the parameter unit.

center The origin of the coordinate system. The numbering system depends on the parameter unit.

unit 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 (see Appendix A [NAIF identification numbers], page 51 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 `calceph_compute`).

PV An array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot).

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 or the derivatives of the angles of the librations of the Moon are expressed in days if unit contains `CALCEPH_UNIT_DAY`. The velocity, 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 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=2451624.5 and at 2451624.9

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

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

  /* compute same quantity as the previous call using NAIF ID */
  calceph_compute_unit(peph, jd0, dt1,
                       NAIFID_MARS_BARYCENTER,
                       NAIFID_SUN,
                       CALCEPH_USE_NAIFID+CALCEPH_UNIT_KM
                       +CALCEPH_UNIT_SEC,
                       PV);
  for(j=0; j<6; j++) printf("%23.16E\n", PV[j]);

 /* the heliocentric coordinates of Mars in AU and AU/day */
  calceph_compute_unit(peph, jd0, dt1, 4, 11,
                       CALCEPH_UNIT_AU+CALCEPH_UNIT_DAY,
                       PV);
  for(j=0; j<6; j++) printf("%23.16E\n", PV[j]);

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

5.5.3.6 calceph_orient_unit

`int calceph_orient_unit (`*t_calcephbin** **`eph`**, *double* ***`JD0`***, *double* **`time`**, *int* **`target`**, *int* **`unit`**, *double* ***`PV`***[*6*]) [C]

`function calceph_orient_unit (`***`eph`***, ***`JD0`***, **`time`**, **`target`**, **`unit`**, ***`PV`***) *BIND*(*C*) [Fortran 2003]

TYPE(C_PTR), VALUE, intent(in) :: eph

```
        REAL(C_DOUBLE), VALUE, intent(in) :: JD0
        REAL(C_DOUBLE), VALUE, intent(in) :: time
        INTEGER(C_INT), VALUE, intent(in) :: target
        INTEGER(C_INT), VALUE, intent(in) :: unit
        REAL(C_DOUBLE), intent(out) :: PV(6)
        INTEGER(C_INT) :: calceph_compute_unit
```

`function f90calceph_orient_unit (`*`eph`*`, `*`JD0`*`, `*`time`*`,` [Fortran 77/90/95]
`  `*`target`*`, `*`unit`*`, `*`PV`*` )`

```
        INTEGER(8), intent(in) :: eph
        REAL(8), intent(in) :: JD0
        REAL(8), intent(in) :: time
        INTEGER, intent(in) :: target
        INTEGER, intent(in) :: unit
        REAL(8), intent(out) :: PV(6)
        INTEGER :: f90calceph_compute_unit
```

PV = *eph*`.orient_unit (`*`JD0`*`, `*`time`*`, `*`target`*`, `*`unit`*`)` [Python 2/3]

```
        <type 'list'> PV
        <type 'calcephpy.CalcephBin'> eph
        <type 'float'> JD0
        <type 'float'> time
        <type 'int'> target
        <type 'int'> unit
```

This function reads, if needed, in the ephemeris file associated to *eph* and interpolates the orientation of a single body (*target*), from the ephemeris file, previously opened with the function `calceph_open`, for the time *JD0*+*time* and stores the results to *PV*. 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.

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

The arguments are :

JD0 Integer part of the Julian Date.

time Fraction part of the Julian Date.

target The body whose orientations are requested. The numbering system depends on the parameter unit.

unit 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 (see Appendix A [NAIF identification numbers], page 51 for the list).

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

PV An array to receive the euler angles and their derivatives for the orientation of the body.

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 of the librations of the Moon are expressed in radians if unit contains `CALCEPH_UNIT_RAD`.

The following example prints the angles of libration of the Moon at time=2451624.5 and at 2451624.9

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

/* open the ephemeris file */
peph = calceph_open("example1.dat");
if (peph)
{
  calceph_prefetch(peph);

  calceph_orient_unit(peph, jd0, dt1, NAIFID_MOON,
                      CALCEPH_USE_NAIFID+CALCEPH_UNIT_RAD
                      +CALCEPH_UNIT_SEC,
                      PV);
  for(j=0; j<6; j++) printf("%23.16E\n", PV[j]);

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

5.5.3.7 calceph_compute_order

`int calceph_compute_order` (*t_calcephbin** **`eph`**, *double* ***`JD0`***, *double* **`time`**, *int* **`target`**, *int* **`center`**, *int* **`unit`**, *int* **`order`**, *double* ****`PVAJ`***) [C]

`function calceph_compute_order` (***`eph`***, ***`JD0`***, **`time`**, **`target`**, ***`center`***, ***`unit`***, ***`order`***, ***`PVAJ`***) *BIND*(*C*) [Fortran 2003]

```
TYPE(C_PTR), VALUE, intent(in) :: eph
REAL(C_DOUBLE), VALUE, intent(in) :: JD0
REAL(C_DOUBLE), VALUE, intent(in) :: time
INTEGER(C_INT), VALUE, intent(in) :: target
INTEGER(C_INT), VALUE, intent(in) :: center
INTEGER(C_INT), VALUE, intent(in) :: unit
```

```
INTEGER(C_INT), VALUE, intent(in) :: order
REAL(C_DOUBLE), intent(out) :: PVAJ(12)
INTEGER(C_INT) :: calceph_compute_order
```

`function f90calceph_compute_order (`*`eph`*`, `*`JD0`*`, `*`time`*`,` *`target`*`, `*`center`*`, `*`unit`*`, `*`order`*`, `*`PVAJ`*` )` [Fortran 77/90/95]

```
INTEGER(8), intent(in) :: eph
REAL(8), intent(in) :: JD0
REAL(8), intent(in) :: time
INTEGER, intent(in) :: target
INTEGER, intent(in) :: center
INTEGER, intent(in) :: unit
INTEGER, intent(in) :: order
REAL(8), intent(out) :: PVAJ(12)
INTEGER :: f90calceph_compute_order
```

PVAJ = *eph*`.compute_order (`*`JD0`*`, `*`time`*`, `*`target`*`, `*`center`*`, `*`unit`*`,` *`order`*`)` [Python 2/3]

```
<type 'list'> PVAJ
<type 'calcephpy.CalcephBin'> eph
<type 'float'> JD0
<type 'float'> time
<type 'int'> target
<type 'int'> center
<type 'int'> unit
<type 'int'> order
```

This function is similar to the function `calceph_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*), from the ephemeris file, previously opened with the function `calceph_open`, for the time *JD0+time* and stores the results to *PVAJ*. The order of the derivatives are specified by *order*. 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.

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

The arguments are :

JD0 Integer part of the Julian Date.

time Fraction part of the Julian Date.

target The body or reference point whose coordinates are required. The numbering system depends on the parameter unit.

center The origin of the coordinate system. The numbering system depends on the parameter unit.

unit 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 (see Appendix A [NAIF identification numbers], page 51 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 `calceph_compute`).

order 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 `calceph_compute_order` is the same as `calceph_compute_unit`.

PVAJ An array to receive the cartesian position (x,y,z) and the derivatives. This array must be large enough to store the results. For the C interface, the size of this array must be equal to 3*(order+1).

- PVAJ[0..2] using the C interface, or PVAJ[1:3] using the fortran interface, contain the position (x,y,z) and is always valid.
- PVAJ[3..5] using the C interface, or PVAJ[4:6] using the fortran interface, contain the velocity (dx/dt,dy/dt,dz/dt) and is only valid if *order* is greater or equal to 1.
- PVAJ[6..8] using the C interface, or PVAJ[7:9] using the fortran interface, 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[9..11] using the C interface, or PVAJ[10:12] using the fortran interface, 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 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*.

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

```
int res;
int j;
double jd0=2451624;
double dt1=0.5E0;
t_calcephbin *peph;
double PVAJ[12];
double P[3];

/* open the ephemeris file */
peph = calceph_open("example1.dat");
if (peph)
{
  /* compute only the heliocentric position of Mars in km */
  calceph_compute_order(peph, jd0, dt1, 4, 11,
                        CALCEPH_UNIT_KM+CALCEPH_UNIT_SEC,
                        0,
                        P);
  for(j=0; j<3; j++) printf("%23.16E\n", P[j]);

  /* compute positions, velocities, accelerations and jerks
  of Mars in km and seconds */
  calceph_compute_order(peph, jd0, dt1,
                        NAIFID_MARS_BARYCENTER,
                        NAIFID_SUN,
                        CALCEPH_USE_NAIFID+CALCEPH_UNIT_KM
                        +CALCEPH_UNIT_SEC,
                        3,
                        PVAJ);
  for(j=0; j<12; j++) printf("%23.16E\n", PVAJ[j]);

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

5.5.3.8 calceph_orient_order

`int calceph_orient_order` (*t_calcephbin** **`eph`**, *double* **`JD0`**, *double* **`time`**, *int* **`target`**, *int* **`unit`**, *int* **`order`**, *double* ***`PVAJ`**) [C]

`function calceph_orient_order (`*eph*, *JD0*, *time*, *target*, *unit*, *order*, *PVAJ* `)` *BIND*(*C*) [Fortran 2003]

```
TYPE(C_PTR), VALUE, intent(in) :: eph
REAL(C_DOUBLE), VALUE, intent(in) :: JD0
REAL(C_DOUBLE), VALUE, intent(in) :: time
INTEGER(C_INT), VALUE, intent(in) :: target
INTEGER(C_INT), VALUE, intent(in) :: unit
INTEGER(C_INT), VALUE, intent(in) :: order
REAL(C_DOUBLE), intent(out) :: PVAJ(12)
INTEGER(C_INT) :: calceph_orient_order
```

`function f90calceph_orient_order (`*eph*, *JD0*, *time*, *target*, *unit*, *order*,*PVAJ* `)` [Fortran 77/90/95]

```
INTEGER(8), intent(in) :: eph
REAL(8), intent(in) :: JD0
REAL(8), intent(in) :: time
INTEGER, intent(in) :: target
INTEGER, intent(in) :: unit
INTEGER, intent(in) :: order
REAL(8), intent(out) :: PVAJ(12)
INTEGER :: f90calceph_orient_order
```

PVAJ = *eph*`.orient_order (`*JD0*, *time*, *target*, *unit*, *order*`)` [Python 2/3]

```
<type 'list'> PVAJ
<type 'calcephpy.CalcephBin'> eph
<type 'float'> JD0
<type 'float'> time
<type 'int'> target
<type 'int'> unit
<type 'int'> order
```

This function is similar to the function `calceph_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*), from the ephemeris file, previously opened with the function `calceph_open`, for the time *JD0*+*time* and stores the results to *PVAJ*. The order of the derivatives are specified by *order*. 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.

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

The arguments are :

JD0 Integer part of the Julian Date.

time Fraction part of the Julian Date.

target The body whose orientations are requested. The numbering system depends on the parameter unit.

unit 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 (see Appendix A [NAIF identification numbers], page 51 for the list).

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

order 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 `calceph_orient_order` is the same as `calceph_orient_unit`.

PVAJ An array to receive the euler angles and their different order of the derivatives for the orientation of the body. This array must be large enough to store the results. For the C interface, the size of this array must be equal to 3*(order+1).

- PVAJ[0..2] using the C interface, or PVAJ[1:3] using the fortran interface, contain the angles and is always valid.
- PVAJ[3..5] using the C interface, or PVAJ[4:6] using the fortran interface, contain the first derivative and is only valid if *order* is greater or equal to 1.
- PVAJ[6..8] using the C interface, or PVAJ[7:9] using the fortran interface, contain the second derivative and is only valid if *order* is greater or equal to 2.
- PVAJ[9..11] using the C interface, or PVAJ[10:12] using the fortran interface, contain the third derivative and is only valid if *order* is equal to 3.

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 of the orientation are expressed in radians if unit contains `CALCEPH_UNIT_RAD`.

The following example prints only the angles of libration of the Moon at time=2451624.5 and at 2451624.9

```
int res;
int j;
double jd0=2451624;
double dt1=0.5E0;
t_calcephbin *peph;
double A[3];

/* open the ephemeris file */
peph = calceph_open("example1.dat");
if (peph)
{
  calceph_prefetch(peph);

  calceph_orient_order(peph, jd0, dt1, NAIFID_MOON,
          CALCEPH_USE_NAIFID+CALCEPH_UNIT_RAD+CALCEPH_UNIT_SEC,
          0,
          A);
  for(j=0; j<3; j++) printf("%23.16E\n", A[j]);

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

5.5.3.9 calceph_getconstant

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

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

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

`value = eph.getconstant (name)` [Python 2/3]
<type 'float'> value
<type 'calcephpy.CalcephBin'> eph
<type 'str'> name

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

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

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

```
double AU;
t_calcephbin *peph;

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

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

5.5.3.10 calceph_getconstantcount

`int calceph_getconstantcount (`*t_calcephbin**` eph` `)` [C]

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

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

eph`.getconstantcount ()` [Python 2/3]
<type 'calcephpy.CalcephBin'> eph

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

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

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

```
int count;
t_calcephbin *peph;

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

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

5.5.3.11 calceph_getconstantindex

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

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

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

`function f90calceph_getconstantindex (`**`eph`**, **`index`**, **`name`**, **`value`**`)` [Fortran 77/90/95]

```
INTEGER(8), intent(in) :: eph
INTEGER(INT), intent(in) :: index
CHARACTER(len=CALCEPH_MAX_CONSTANTNAME), intent(out) ::
name
REAL(8), intent(out) :: value
INTEGER :: f90calceph_getconstantindex
```

`name, value` = **`eph`**`.getconstantindex (`**`index`**`)` [Python 2/3]

```
<type 'str'> name
<type 'float'> value
<type 'calcephpy.CalcephBin'> eph
<type 'int'> index
```

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 the descriptor *eph*. The value of *index* must be between 1 and `calceph_getconstantcount(`*eph*`)`.

If the fortran-77/90/95 interface is used, trailing blanks are added to the name of the constant.

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

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

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

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

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

    ! close the ephemeris file
                  call calceph_close(peph)
                endif
```

5.5.3.12 calceph_close

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

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

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

eph`.close ()` [Python 2/3]
<type 'calcephpy.CalcephBin'> eph

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

5.6 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 `calceph_seterrorhandler` (see Section 5.7 [Error functions], page 47).

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

5.6.1 Thread notes

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

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

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

5.6.2 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 C language is '`csingle.c`'. The example in Fortran 2003 language is '`f2003single.f`'. The example in Fortran 77/90/95 language is '`f77single.f`'.

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

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

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

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

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

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

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

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

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

5.6.3 Functions

5.6.3.1 calceph_sopen

`int calceph_sopen ( const char *filename )` [C]

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

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

This function opens the file whose pathname is the string pointed to by filename, reads the 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, 20 , 102, 103 and 120.
- meta kernel (KPL/MK) files.
- frame kernel (KPL/FK) files. Only a basic support is provided.

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

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

The following example opens the ephemeris file example1.dat

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

5.6.3.2 calceph_scompute

`int calceph_scompute ( double JD0, double time, int target, int center, double PV[6] )` [C]

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

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

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

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

The arguments are :

JD0 Integer part of the Julian Date.

time Fraction part of the Julian Date.

target The body or reference point whose coordinates are required (see the list, below).

center The origin of the coordinate system (see the list, below). If *target* is 15, 16 or 17 (libration, TT-TDB or TCG-TCB), *center* must be '0'.

PV An array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot).

The position is expressed in Astronomical Unit (au) and the velocity is expressed in Astronomical Unit per day (au/day).

If the target is *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 angles of the librations of the Moon are expressed in radians and their derivatives are expressed in radians per day.

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
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

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

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

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

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

5.6.3.3 calceph_sgetconstant

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

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

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

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

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

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

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

5.6.3.4 calceph_sgetconstantcount

`int calceph_sgetconstantcount ( )` [C]

`function calceph_sgetconstantcount ()` *BIND*(*C*) [Fortran 2003]
INTEGER(C_INT) :: calceph_sgetconstantcount

`function f90calceph_sgetconstantcount ()` [Fortran 77/90/95]
INTEGER :: f90calceph_sgetconstantcount

This function returns the number of constants available in the header of the ephemeris file. The function `calceph_sopen` must be previously called before. On exit, it returns 0 if an error occurs, otherwise the return value is a non-zero value.

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

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

5.6.3.5 calceph_sgetconstantindex

`int calceph_sgetconstantindex (`*int* `index`, *char* `name`[*CALCEPH_MAX_CONSTANTNAME*], *double* *`value`) [C]

`function calceph_sgetconstantindex (index, name, value)` [Fortran 2003]
BIND(*C*)
INTEGER(C_INT), VALUE, intent(in) :: index
CHARACTER(len=1,kind=C_CHAR),
dimension(CALCEPH_MAX_CONSTANTNAME), intent(out) :: name
REAL(C_DOUBLE), intent(out) :: value
INTEGER(C_INT) :: calceph_sgetconstantindex

`function f90calceph_sgetconstantindex (index, name, value)` [Fortran 77/90/95]
INTEGER(INT), intent(in) :: index

```
CHARACTER(len=CALCEPH_MAX_CONSTANTNAME), intent(out) :: name
REAL(8), intent(out) :: value
INTEGER :: f90calceph_sgetconstantindex
```

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

If the fortran-77/90/95 interface is used, trailing blanks are added to the name of the constant.

The function `calceph_sopen` must be previously called before.

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

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

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

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

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


    ! close the ephemeris file
               call calceph_sclose
```

5.6.3.6 calceph_sclose

`void calceph_sclose ( )` [C]

`subroutine calceph_sclose ( )` [Fortran 2003]

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

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

5.7 Error functions

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

5.7.1 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 C language is '`cerror.c`'. The example in Fortran 2003 language is '`f2003error.f`'. The example in Fortran 77/90/95 language is '`f77error.f`'.

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

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

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

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

            stop
            end
```

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

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

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

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


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

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

        ....

     stop
     end
```

5.7.2 calceph_seterrorhandler

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

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

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

`calcephpy.seterrorhandler (typehandler,` *userfunc*`)` [Python 2/3]
<type 'int'> typehandler
<type 'function'> userfunc

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

The possible values for *typehandler* are :

value	**meaning**
1	The library displays a message and continues the execution. The functions return an error code. The python interface raises 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 `exit`.
3	The library calls the user function *userfunc* with the message.

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

The function *userfunc* must be defined as

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

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

`def userfunc (`***msg***`)` [Python 2/3]
<type 'str'> msg

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

5.8 Miscellaneous functions

5.8.1 calceph_getversion_str

`void calceph_getversion_str ( char` [C]
`version`[*CALCEPH_MAX_CONSTANTNAME*])

`subroutine calceph_getversion_str (` *`version`*`) BIND(C)` [Fortran 2003]
CHARACTER(len=1,kind=C_CHAR),
dimension(CALCEPH_MAX_CONSTANTNAME), intent(out) :: name

`function f90calceph_getversion_str (`*`version`*`)` [Fortran 77/90/95]
CHARACTER(len=CALCEPH_MAX_CONSTANTNAME), intent(out) :: name

`version` `= calcephpy.getversion_str ()` [Python 2/3]
<type 'str'> *version*

This function returns the version of the CALCEPH Library, as a null-terminated string.
If the fortran interface is used, trailing blanks are added to the name version.

```
from calcephpy import *
print('version=', getversion_str())
```

Appendix A NAIF identification numbers

The following predefined values must be used as the target body and origin of the coordinate system with the functions `calceph_compute_unit`, `calceph_orient_unit` `calceph_compute_order` or `calceph_orient_order` if and only if the value `CALEPH_USE_NAIFID` has been set in the parameter `unit`.

This list is already predefined in :

- interface file `calceph.h` for the C interface.
- interface file `f90calceph.h` for the Fortran 77/90/95 interface.
- module file `calceph.mod` for the Fortran 2003 interface.
- class `NaifId` of the module `calcephpy` for the Python 2/3 interface. Relative to C or Fortran interface, the prefix `NAIFID_` is deleted for the following numbers.

A.1 NAIF identification numbers for the 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

A.2 NAIF identification numbers for the 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[3]

A.3 NAIF identification numbers for the planet centers and satellites

Predefined Macros	**NAIF ID**	**Name**
NAIFID_MERCURY	199	Mercury

[1] These values must only be used as a center body

[2] These values must only be used as a target body

[3] These values must only be used as a target body

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_PASIPHAE	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_ADRASTEA	515	Adrastea
NAIFID_METIS	516	Metis
NAIFID_CALLIRRHOE	517	Callirrhoe
NAIFID_THEMISTO	518	Themisto
NAIFID_MAGACLITE	519	Magaclite
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_ORTHOSIE	535	Orthosie
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_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
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
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

A.4 NAIF identification numbers for the 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	DDrrest
NAIFID_DANIEL	1000018	Daniel
NAIFID_DE_VICO_SWIFT	1000019	De Vico-Swift
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
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_SHAJN_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
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

Appendix B Release notes

- Version 1.0.0
  Initial release.

- 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 1.0.2
  Fix memory leaks in the fortran-90 interface.

- Version 1.0.3
  Support the JPL ephemeris file DE423.

- Version 1.1.0
  Add the function calceph_seterrorhandler for the custom error handlers.

- Version 1.1.1
  Fix a compilation error in util.h and a warning with the sun studio compilers.

- 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 1.2.0
  Change the licensing : triple licenses to support integration in BSD software.
  Remove explicit dependencies on the record size for DExxx.

- 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 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 2.2.0
  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 2.2.1
  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 2.2.2
  Support the compilation in the standard C89.

- 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 2.2.4
  Update the version number of the dynamic library.

- 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 2.3.0
  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 2.3.1
  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 2.3.2
  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.