CALCEPH - Python language

Release 5.0.0

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

Jul 06, 2026

CONTENTS

This manual documents how to install and use the CALCEPH Library using the Python interface.

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

CHAPTER

ONE

INTRODUCTION

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

- text Planetary Constants Kernel (KPL/PCK) files
- binary PCK (DAF/PCK) files.
- binary SPK (DAF/SPK) files containing segments of type 1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 19, 20, 21, 102, 103 and 120.
- meta kernel (KPL/MK) files.
- frame kernel (KPL/FK) files. Only a basic support is provided.
- leapseconds clock kernel (KPL/LSK) files.
- instrument kernel (KPL/IK) files.
- spacecraft clock kernel (KPL/SCLK) files.

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

This library could access to the following ephemeris

- INPOP06 or later
- DE200
- DE403 or later
- EPM2011 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 https://www.imcce.fr/inpop/inpop_file_format_2_0.pdf

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

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

CHAPTER

TWO

INSTALLATION

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

2.1 Instructions

The python interface of the library should be installed using the package management system **pip** on all operating systems : Windows and Unix-like system (Linux, macOS, BSD, Cygwin, ...).

A python interpreter, compliant at least with Python 3.0 specifications, and the package Cython, setuptools and numpy are required to install the python interface of the library.

Some Linux distributions require the installation development tools of the python software. The name of this package may change on other Linux distributions or operating systems :

- Debian or Ubuntu distributions : package **python3-dev** and **python-dev-is-python3** are required.
- Opensuse distribution : package **python3-dev** is required.
- Fedora or RedHat distributions : **python3-devel** is required.

In addition you need the software *CMake* and a C compiler :

- On Unix-like operating systems (Linux, MacOS X, ...), you should install gcc or clang.
- On windows operating system, you need the Microsoft Visual C++ compiler. If you don't have a C compiler already installed, you can download the community edition of visual studio *https://visualstudio.microsoft.com/fr/vs/features/cplusplus/*. Before the execution of the next steps, you should execute the following line in the same terminal. You may have to adjust the path according to your version of the Visual studio compiler.

  ```
  "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\
  ↪Build\vcvars64.bat"
  ```

2.1.1 Using pip

Depending on your python installation, the command **pip** may be replaced by **pip3**. If you use the distribution *anaconda*, you should prefer to use the instruction from the Anaconda section.

The steps are :

- Install the requirements

  ```
  pip install Cython setuptools numpy
  ```

- Install the library

```
pip install calcephpy
```

2.1.2 Using Anaconda

Depending on your anaconda installation, the command **pip** may be replaced by **pip3**.

The steps are :

- Install the gcc compiler from the Anaconda compiler tools :

  See the instruction on *https://docs.conda.io/projects/conda-build/en/latest/resources/compiler-tools.html*

  For example, on the operating system Linux, it will be :

  ```
  conda install gcc_linux-64 cmake make
  ```

- Install the other requirements

  ```
  conda install Cython setuptools numpy
  ```

- Install the library

  ```
  pip install calcephpy
  ```

CHAPTER

THREE

LIBRARY INTERFACE

3.1 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")
jd0 = 2451542
dt = 0.5
PV = peph.compute_unit(jd0, dt, NaifId.MOON, NaifId.EARTH,
                       Constants.UNIT_KM+Constants.UNIT_SEC+Constants.USE_NAIFID)
print(PV)
peph.close()
```

3.2 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.

3.3 Types

`calcephpy.`**`CalcephBin`**

This type contains all information to access a single ephemeris file or a set of ephemeris files to be used together.

calcephpy.**NaifId**

This type contains the NAIF identification numbers.

calcephpy.**Constants**

This type contains all constants defined in the library, except the NAIF identification numbers.

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

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

ASTEROID

This integer defines the offset value for the asteroids, starting at 2E6, that must be used as target or center for the computation functions, such as `calcephpy.CalcephBin.compute()`.

ASTEROID_8

This integer defines the offset value for the asteroids, starting at 2E8, that must be used as target or center for the computation functions, such as `calcephpy.CalcephBin.compute()`.

The following constants specify in which units are expressed the output of the computation functions, such as `calcephpy.CalcephBin.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.

`OUTPUT_EULERANGLES`

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

`OUTPUT_NUTATIONANGLES`

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

`USE_NAIFID`

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

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

`SEGTYPE_ORIG_0`

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

`SEGTYPE_SPK_1`

`SEGTYPE_SPK_2`

`SEGTYPE_SPK_3`

`SEGTYPE_SPK_5`

`SEGTYPE_SPK_8`

`SEGTYPE_SPK_9`

`SEGTYPE_SPK_12`

`SEGTYPE_SPK_13`

`SEGTYPE_SPK_14`

`SEGTYPE_SPK_17`

`SEGTYPE_SPK_18`

`SEGTYPE_SPK_19`

`SEGTYPE_SPK_20`

`SEGTYPE_SPK_21`

`SEGTYPE_SPK_102`

`SEGTYPE_SPK_103`

`SEGTYPE_SPK_120`

This integer defines the type of segments (1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 19, 20, 21, 102, 103 and 120) for the SPICE Kernel files.

The following constants specify the time scale used for time conversion functions :

`TDB`

`TCB`

`UTC`

`TAI`

`TT`

`TIMESCALE_FROM_STR`

These integers define the timescales.

CHAPTER

FOUR

READING/EVALUATION 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 *calcephpy.seterrorhandler()*.

4.1 Time notes

The functions *calcephpy.CalcephBin.compute()*, *calcephpy.CalcephBin.compute_unit()*, *calcephpy.CalcephBin.compute_order()*, *calcephpy.CalcephBin.orient_unit()*, ... only accept a date expressed in the same time scale as the ephemeris files, which can be retrieved using the function *calcephpy.CalcephBin.gettimescale()*. Ephemeris files are generally expressed using the time scale TDB. If a date, expressed in the TT (Terrestrial Time) time scale, is supplied to them, these functions will return an erroneous position of the order of several tens of meters for the planets. If a date, expressed in the Coordinated Universal Time (UTC), is supplied to them, these functions will return a very large erroneous position over several thousand kilometers for the planets.

4.2 Thread notes

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

It's not safe for two threads to call the functions with the same object of type `CalcephBin` if and only if the function *calcephpy.CalcephBin.isthreadsafe()* returns a non-zero value. A previous call to the function *calcephpy.CalcephBin.prefetch()* is required for the function *calcephpy.CalcephBin.isthreadsafe()* to return a non-zero value.

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

4.3 Usage

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

The example in Python language is `pymultiple.py`.

4.4 Functions

4.4.1 calcephpy.CalcephBin.open

`calcephpy.CalcephBin.open(filename)` → eph

Parameters
filename (*str*) -- pathname of the file

Returns
ephemeris descriptor

Return type
calcephpy.CalcephBin

This function opens the file whose pathname is the string pointed to by filename, reads the two header blocks of this file and returns an ephemeris descriptor associated to it. This file must be compliant to the format specified by the 'original JPL binary' , 'INPOP 2.0 binary' or 'SPICE' ephemeris file. At the moment, supported SPICE files are the following :

- text Planetary Constants Kernel (KPL/PCK) files
- binary PCK (DAF/PCK) files.
- binary SPK (DAF/SPK) files containing segments of type 1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 19, 20, 21, 102, 103 and 120.
- meta kernel (KPL/MK) files.
- frame kernel (KPL/FK) files. Only a basic support is provided.

Just after the call of `calcephpy.CalcephBin.open()`, the function `calcephpy.CalcephBin.prefetch()` should be called to accelerate future computations.

The function `calcephpy.CalcephBin.close()` must be called to free allocated memory by this function.

The following example opens the ephemeris file example1.dat

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")

# ...  computation ...

peph.close()
```

noindex

`calcephpy.CalcephBin.open(array_filename)` → eph

Parameters
array_filename (*list*) -- array of pathname of the files

Returns
ephemeris descriptor

Return type
calcephpy.CalcephBin

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

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

- text Planetary Constants Kernel (KPL/PCK) files
- binary PCK (DAF/PCK) files.
- binary SPK (DAF/SPK) files containing segments of type 1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 19, 20, 21, 102, 103 and 120.
- meta kernel (KPL/MK) files.
- frame kernel (KPL/FK) files. Only a basic support is provided.

The single descriptor internally maintains a table to respond to the following queries for computing the position-velocity vector between bodies present in these different files.

Just after the call of `calcephpy.CalcephBin.open()`, the function `calcephpy.CalcephBin.prefetch()` should be called to accelerate future computations.

The function `calcephpy.CalcephBin.close()` must be called to free allocated memory by this function.

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

```
from calcephpy import CalcephBin
peph = CalcephBin.open(['example1.bsp', 'example1.tpc'])

# ... computation ...

peph.close()
```

4.4.2 calcephpy.CalcephBin.prefetch

`calcephpy.CalcephBin.prefetch()`

This function prefetches to the main memory all files associated to the ephemeris descriptor. This prefetching operation will accelerate the further computations performed with `calcephpy.CalcephBin.compute()`, `calcephpy.CalcephBin.compute_unit()`, `calcephpy.CalcephBin.compute_order()`, `calcephpy.CalcephBin.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.

4.4.3 calcephpy.CalcephBin.isthreadsafe

`calcephpy.CalcephBin.isthreadsafe()`

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

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

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

4.4.4 calcephpy.CalcephBin.compute

`calcephpy.CalcephBin.compute`(*JD0*, *time*, *target*, *center*) → PV

Parameters

- **JD0** (*float/list/numpy.ndarray*) -- Integer part of the Julian date (TDB or TCB)
- **time** (*float/list/numpy.ndarray*) -- Fraction part of the Julian date (TDB or TCB)
- **target** (*int*) -- The body or reference point whose coordinates are required (see the list, below).
- **center** (*int*) -- The origin of the coordinate system (see the list, below). If *target* is 14, 15, 16 or 17 (nutation, libration, TT-TDB or TCG-TCB), *center* must be *0*.

Returns

Depending on the target value, an array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot), or a time scale transformation value, or the angles of the librations of the Moon and their derivatives, or the nutation angles and their derivatives.

Return type

list

This function reads, if needed, in the ephemeris file *self* and interpolates a single object, usually the position and velocity of one body (*target*) relative to another (*center*) for the time *JD0+time* and stores the results to *PV*. The ephemeris file *self* must have been previously opened with the function `calcephpy.CalcephBin.open()`.

The returned array *PV* has the following properties

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

If *JD0* and *time* are list or NumPy's array (1D) of double-precision floating-point values, the returned array *PV* is a list of 6 arrays. Each array contain a single component of position or velocity (e.g., PV[0] contains the coordinate X, PV[1] contains the coordinate Y, ...) .

The date (JD0, time) should be expressed in the same time scale as the ephemeris files, which can be retrieved using the function `calcephpy.CalcephBin.gettimescale()`. To get the best numerical precision for the interpolation, the time is splitted in two floating-point numbers. The argument *JD0* should be an integer and *time* should be a fraction of the day. But you may call this function with *time=0* and *JD0*, the desired time, if you don't take care about numerical precision.

> **Warning**
>
> If a date, expressed in the Coordinated Universal Time (UTC), is supplied to this function, a very large erroneous position will be returned. The function `calcephpy.CalcephBin.time_cal_utc_to_jd_tdb()` can convert the UTC calendar date to the Julian Day expressed in the time scale TDB.

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

value	meaning
1	Mercury Barycenter
2	Venus Barycenter
3	Earth
4	Mars Barycenter
5	Jupiter Barycenter
6	Saturn Barycenter
7	Uranus Barycenter
8	Neptune Barycenter
9	Pluto Barycenter
10	Moon
11	Sun
12	Solar Sytem barycenter
13	Earth-moon barycenter
14	Nutation angles
15	Librations
16	TT-TDB
17	TCG-TCB
asteroid number + CALCEPH_ASTEROID	asteroid (2E6+...)
asteroid number + CALCEPH_ASTEROID_8	asteroid (2E8+...)

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

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

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

```
from calcephpy import *

def printcoord(PV,name):
    print('{0} :\n{1}\n'.format(name,PV))

jd0=2442457
dt1=0.5E0
dt2=0.9E0

peph = CalcephBin.open("example1.dat")

PV1 = peph.compute(jd0, dt1, 4, 11)
printcoord(PV1,"heliocentric coordinates of Mars")

PV2 = peph.compute(jd0, dt2, 4, 11)
printcoord(PV2,"heliocentric coordinates of Mars")

peph.close()
```

4.4.5 calcephpy.CalcephBin.compute_unit

`calcephpy.CalcephBin.compute_unit`(*JD0*, *time*, *target*, *center*, *unit*) → PV

Parameters

- **JD0** (*float/list/numpy.ndarray*) -- Integer part of the Julian date (TDB or TCB)
- **time** (*float/list/numpy.ndarray*) -- Fraction part of the Julian date (TDB or TCB)
- **target** (*int*) -- The body or reference point whose coordinates are required. The numbering system depends on the parameter unit.
- **center** (*int*) -- The origin of the coordinate system. The numbering system depends on the parameter unit.
- **unit** (*int*) --

  The units of PV.

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

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

  If the unit doesnot contain *USE_NAIFID*, the old number system is used for the target and the center (see the list in the function `calcephpy.CalcephBin.compute()`).

Returns

Depending on the target value, an array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot), or a time scale transformation value, or the angles of the librations of the Moon and their derivatives, or the nutation angles and their derivatives.

Return type

list

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

This function reads, if needed, in the ephemeris file *self* and interpolates a single object, usually the position and velocity of one body (*target*) relative to another (*center*) for the time *JD0+time* and stores the results to *PV*. The ephemeris file *self* must have been previously opened with the function `calcephpy.CalcephBin.open()`. The output values are expressed in the units specified by *unit*.

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

The date (JD0, time) should be expressed in the same time scale as the ephemeris files, which can be retrieved using the function `calcephpy.CalcephBin.gettimescale()`.

> **Warning**
>
> If a date, expressed in the Coordinated Universal Time (UTC), is supplied to this function, a very large erroneous position will be returned. The function `calcephpy.CalcephBin.time_cal_utc_to_jd_tdb()` can convert the UTC calendar date to the Julian Day expressed in the time scale TDB.

The returned array *PV* has the following properties

- If the target is the time scale transformation TT-TDB, only the first element of this array will get the result.
- If the target is the time scale transformation *TCG-TCB*, only the first element of this array will get the result.
- If the target is *Librations*, the array contains the angles of the librations of the Moon and their derivatives.

- If the target is *Nutations*, the array contains the nutation angles and their derivatives.
- Otherwise the returned value is the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot).

If *JD0* and *time* are list or NumPy's array (1D) of double-precision floating-point values, the returned array *PV* is a list of 6 arrays. Each array contain a single component of position or velocity (e.g., PV[0] contains the coordinate X, PV[1] contains the coordinate Y, ...) .

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

- The position and velocity are expressed in Astronomical Unit (au) if unit contains *UNIT_AU*.
- The position and velocity are expressed in kilometers if unit contains *UNIT_KM*.
- The velocity, TT-TDB, TCG-TCB, the derivatives of the angles of the nutation, or the derivatives of the librations of the Moon or are expressed in days if unit contains *UNIT_DAY*.
- The velocity, TT-TDB, TCG-TCB, the derivatives of the angles of the nutation, or the derivatives of the librations of the Moon are expressed in seconds if unit contains *UNIT_SEC*.
- The angles of the librations of the Moon or the nutation angles are expressed in radians if unit contains *UNIT_RAD*.

For example, to get the position and velocities expressed in kilometers and kilometers/seconds, the unit must be set to *UNIT_KM* + *UNIT_SEC*.

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

```
from calcephpy import *

def printcoord(PV,name):
    print('{0} :\n{1}\n'.format(name,PV))

jd0=2442457
dt=0.5E0

peph = CalcephBin.open("example1.dat")

PV1 = peph.compute_unit(jd0, dt, 4, 11, Constants.UNIT_KM+Constants.UNIT_SEC)
printcoord(PV1,"heliocentric coordinates of Mars")

PV2 = peph.compute_unit(jd0, dt, NaifId.MARS_BARYCENTER, NaifId.SUN,
                        Constants.UNIT_KM+Constants.UNIT_SEC+Constants.USE_NAIFID)
printcoord(PV2,"heliocentric coordinates of Mars")

peph.close()
```

4.4.6 calcephpy.CalcephBin.orient_unit

`calcephpy.CalcephBin.orient_unit`(*JD0*, *time*, *target*, *unit*) → PV

Parameters

- **JD0** (*float/list/numpy.ndarray*) -- Integer part of the Julian date (TDB or TCB)
- **time** (*float/list/numpy.ndarray*) -- Fraction part of the Julian date (TDB or TCB)
- **target** (*int*) -- The body whose orientations are requested. The numbering system depends on the parameter unit.
- **unit** (*int*) --

The units of PV.

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

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

If the unit does not contain *USE_NAIFID*, the old number system is used for the target (see the list in the function `calcephpy.CalcephBin.compute()`).

Returns

An array to receive the euler angles, or nutation angles, and their derivatives for the orientation of the body.

Return type

list

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

The date (JD0, time) should be expressed in the same time scale as the ephemeris files, which can be retrieved using the function `calcephpy.CalcephBin.gettimescale()`.

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

The returned array *PV* has the following properties

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

If *JD0* and *time* are list or NumPy's array (1D) of double-precision floating-point values, the returned array *PV* is a list of 6 arrays. Each array contain a single component of orientation.

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

- The derivatives of the angles are expressed in days if unit contains *UNIT_DAY*.
- The derivatives of the angles are expressed in seconds if unit contains *UNIT_SEC*.
- The angles and their derivatives are expressed in radians if unit contains *UNIT_RAD*.

For example, to get the nutation angles of the Earth and their derivatives expressed in radian and radian/seconds using the NAIF identification numbering system, the target must be set to NAIFID_EARTH and the unit must be set to *OUTPUT_NUTATIONANGLES* + *UNIT_RAD* + *UNIT_SEC*.

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

```
from calcephpy import *

jd0=2442457
dt=0.5E0

peph = CalcephBin.open("example1.dat")

PV = peph.orient_unit(jd0, dt, NaifId.MOON,
                      Constants.USE_NAIFID+Constants.UNIT_RAD+Constants.UNIT_SEC)
print(PV)
```

(continues on next page)

(continued from previous page)

```
peph.close()
```

4.4.7 calcephpy.CalcephBin.rotangmom_unit

calcephpy.CalcephBin.**rotangmom_unit**(*JD0*, *time*, *target*, *unit*) → PV

Parameters

- **JD0** (*float*) -- Integer part of the Julian date (TDB or TCB)
- **time** (*float*) -- Fraction part of the Julian date (TDB or TCB)
- **target** (*int*) -- The body whose orientations are requested. The numbering system depends on the parameter unit.
- **unit** (*int*) --

  The units of PV.

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

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

  If the unit does not contain *USE_NAIFID*, the old number system is used for the target (see the list in the function *calcephpy.CalcephBin.compute()*).

Returns

An array to receive the angular momentum due to its rotation, divided by the product of the mass and of the square of the radius, and the derivatives, of the body.

Return type

list

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

The date (JD0, time) should be expressed in the same time scale as the ephemeris files, which can be retrieved using the function *calcephpy.CalcephBin.gettimescale()*.

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

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

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

- The angular momentum and its derivative are expressed in days if unit contains *UNIT_DAY*.
- The angular momentum and its derivative are expressed in seconds if unit contains *UNIT_SEC*.

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

```
from calcephpy import *

jd0=2451419
dt=0.5E0
```

(continues on next page)

(continued from previous page)

```
peph = CalcephBin.open("example2_rotangmom.dat")

G = peph.rotangmom_unit(jd0, dt, NaifId.EARTH,
                        Constants.USE_NAIFID+Constants.UNIT_SEC)
print(G)

peph.close()
```

4.4.8 calcephpy.CalcephBin.compute_order

`calcephpy.CalcephBin.`**`compute_order`**(*JD0*, *time*, *target*, *center*, *unit*, *order*) → PVAJ

Parameters

- **JD0** (*float/list/numpy.ndarray*) -- Integer part of the Julian date (TDB or TCB)
- **time** (*float/list/numpy.ndarray*) -- Fraction part of the Julian date (TDB or TCB)
- **target** (*int*) -- The body or reference point whose coordinates are required. The numbering system depends on the parameter unit.
- **center** (*int*) -- The origin of the coordinate system. The numbering system depends on the parameter unit.
- **unit** (*int*) --

  The units of PVAJ.

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

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

  If the unit doesnot contain *USE_NAIFID*, the old number system is used for the target and the center (see the list in the function *calcephpy.CalcephBin.compute()*).
- **order** (*int*) -- 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 *calcephpy.CalcephBin.compute_order()* is the same as *calcephpy.CalcephBin.compute_unit()*.

Returns

Depending on the target value, an array to receive the cartesian position (x,y,z), the velocity (xdot, ydot, zdot), the acceleration and the jerk, or a time scale transformation value, or the angles of the librations of the Moon and their successive derivatives, or the nutation angles and their successive derivatives.

Return type

list

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

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

The returned array *PVAJ* has the following properties

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

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

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

If *JD0* and *time* are list or NumPy's array (1D) of double-precision floating-point values, the returned array *PVAJ* is a list of 3*(order+1) arrays. Each array contain a single component of position, velocity ... (e.g., PV[0] contains the coordinate X, PV[1] contains the coordinate Y, ...) .

The date (JD0, time) should be expressed in the same time scale as the ephemeris files, which can be retrieved using the function `calcephpy.CalcephBin.gettimescale()`.

Warning

If a date, expressed in the Coordinated Universal Time (UTC), is supplied to this function, a very large erroneous position will be returned. The function `calcephpy.CalcephBin.time_cal_utc_to_jd_tdb()` can convert the UTC calendar date to the Julian Day expressed in the time scale TDB.

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

- The position, velocity, acceleration and jerk are expressed in Astronomical Unit (au) if unit contains `UNIT_AU`.
- The position, velocity, acceleration and jerk are expressed in kilometers if unit contains `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 `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 `UNIT_SEC`.
- The angles of the librations of the Moon are expressed in radians if unit contains `UNIT_RAD`.

For example, to get the positions, velocities, accelerations and jerks expressed in kilometers and kilometers/seconds, the unit must be set to *UNIT_KM* + *UNIT_SEC*.

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

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

```
from calcephpy import *

jd0=2442457
dt=0.5E0

peph = CalcephBin.open("example1.dat")

# compute only the heliocentric position of Mars in km
P = peph.compute_order(jd0, dt, NaifId.MARS_BARYCENTER, NaifId.SUN,
                        Constants.UNIT_KM+Constants.UNIT_SEC+Constants.USE_NAIFID, 0)
print(P)

# compute positions, velocities, accelerations and jerks of Mars in km and seconds
PVAJ = peph.compute_order(jd0, dt, NaifId.MARS_BARYCENTER, NaifId.SUN,
                        Constants.UNIT_KM+Constants.UNIT_SEC+Constants.USE_NAIFID, 3)
print(PVAJ)

peph.close()
```

4.4.9 calcephpy.CalcephBin.orient_order

`calcephpy.CalcephBin.`**`orient_order`**(*JD0*, *time*, *target*, *unit*, *order*) → PVAJ

Parameters

- **JD0** (*float/list/numpy.ndarray*) -- Integer part of the Julian date (TDB or TCB)
- **time** (*float/list/numpy.ndarray*) -- Fraction part of the Julian date (TDB or TCB)
- **target** (*int*) -- The body whose orientations are requested. The numbering system depends on the parameter unit.
- **unit** (*int*) --

  The units of PV.
  This integer is a sum of some unit constants (CALCEPH_UNIT_???) and/or the constant *USE_NAIFID*.
  If the unit contains *USE_NAIFID*, the NAIF identification numbering system is used for the target (*NAIF identification numbers* for the list).
  If the unit does not contain *USE_NAIFID*, the old number system is used for the target (see the list in the function *calcephpy.CalcephBin.compute()*).
- **order** (*int*) -- 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 `calcephpy.CalcephBin.orient_order()` is the same as `calcephpy.CalcephBin.orient_unit()`.

Returns
An array to receive the euler angles, or nutation angles, and their derivatives for the orientation of the body.

Return type
list

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

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

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

The returned array *PVAJ* has the following properties

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

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

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

If *JD0* and *time* are list or NumPy's array (1D) of double-precision floating-point values, the returned array *PVAJ* is a list of 3*(order+1) arrays. Each array contain a single component of the orientation.

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

- The derivatives of the angles are expressed in days if unit contains `UNIT_DAY`.
- The derivatives of the angles are expressed in seconds if unit contains `UNIT_SEC`.
- The angles and their derivatives are expressed in radians if unit contains `UNIT_RAD`.

The date (JD0, time) should be expressed in the same time scale as the ephemeris files, which can be retrieved using the function `calcephpy.CalcephBin.gettimescale()`.

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

```
from calcephpy import *

jd0=2442457
dt=0.5E0

peph = CalcephBin.open("example1.dat")
```

(continues on next page)

(continued from previous page)

```
P = peph.orient_order(jd0, dt, NaifId.MOON,
                      Constants.USE_NAIFID+Constants.UNIT_RAD+Constants.UNIT_SEC, 0)
print(P)

peph.close()
```

4.4.10 calcephpy.CalcephBin.rotangmom_order

calcephpy.CalcephBin.**rotangmom_order**(*JD0*, *time*, *target*, *unit*, *order*) → PVAJ

Parameters

- **JD0** (*float*) -- Integer part of the Julian date (TDB or TCB)
- **time** (*float*) -- Fraction part of the Julian date (TDB or TCB)
- **target** (*int*) -- The body whose orientations are requested. The numbering system depends on the parameter unit.
- **unit** (*int*) --

  The units of PV.

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

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

  If the unit does not contain *USE_NAIFID*, the old number system is used for the target (see the list in the function *calcephpy.CalcephBin.compute()*).
- **order** (*int*) -- The order of derivatives.
  - = 0 , only the angular momentum is computed. The first three numbers of PVAJ are valid for the results.
  - = 1 , only the angular momentum and the first derivative are computed. The first six numbers of PVAJ are valid for the results.
  - = 2 , only the angular momentum and the first and second derivatives are computed. The first nine numbers of PVAJ are valid for the results.
  - = 3 , the angular momentum and the first, second and third derivatives are computed. The first twelve numbers of PVAJ are valid for the results.

  If order equals to 1, the behavior of *calcephpy.CalcephBin.rotangmom_order()* is the same as *calcephpy.CalcephBin.rotangmom_unit()*.

Returns
An array to receive the angular momentum due to its rotation, divided by the product of the mass and of the square of the radius, and their different order of the derivatives, of the body.

Return type
list

This function is similar to the function *calcephpy.CalcephBin.orient_unit()*, except that the order of the computed derivatives is specified.

This function reads, if needed, in the ephemeris file *self* and interpolates the angular momentum vector due to the rotation of the body, divided by the product of the mass m and of the square of the radius R, of a single body (*target*) for the time *JD0+time* and stores the results to *PVAJ*. The angular momentum L , due to the rotation of the body, is defined as the product of the inertia matrix I by the angular velocity vector ω. So the returned value is $L/(mR^2) = (I\omega)/(mR^2)$ The order of the derivatives are specified by *order*. The ephemeris file *self* must have been previously opened with the function `calcephpy.CalcephBin.open()`. The output values are expressed in the units specified by *unit*.

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

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

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

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

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

The date (JD0, time) should be expressed in the same time scale as the ephemeris files, which can be retrieved using the function `calcephpy.CalcephBin.gettimescale()`.

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

```
from calcephpy import *

jd0=2451419
dt=0.5E0

peph = CalcephBin.open("example2_rotangmom.dat")

G = peph.rotangmom_order(jd0, dt, NaifId.EARTH,
                         Constants.USE_NAIFID+Constants.UNIT_SEC, 0)
print(G)

peph.close()
```

4.4.11 calcephpy.CalcephBin.getconstant

`calcephpy.CalcephBin.getconstant`(*name*) → value

Parameters
name (*str*) -- name of the constant

Returns
first value of the constant

Return type
float

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

This function is the same function as `calcephpy.CalcephBin.getconstantsd()`.

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
AU = peph.getconstant("AU")
print(AU)
peph.close()
```

4.4.12 calcephpy.CalcephBin.getconstantsd

calcephpy.CalcephBin.**getconstantsd**(*name*) → value

Parameters
: **name** (*str*) -- name of the constant

Returns
: first value of the constant

Return type
: float

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

This function is the same function as *calcephpy.CalcephBin.getconstant()*.

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
AU = peph.getconstantsd("AU")
print(AU)
peph.close()
```

4.4.13 calcephpy.CalcephBin.getconstantvd

calcephpy.CalcephBin.**getconstantvd**(*name*) → arrayvalue

Parameters
: **name** (*str*) -- name of the constant

Returns
: array of values for the constant

Return type
: list

This function returns, as floating-point numbers, all values associated to the constant *name* in the header of the ephemeris file *self*.

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

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
```

(continues on next page)

(continued from previous page)

```
radii = peph.getconstantvd("BODY399_RADII")
print(radii)
peph.close()
```

4.4.14 calcephpy.CalcephBin.getconstantss

`calcephpy.CalcephBin.`**`getconstantss`**`(`*name*`)` → value

Parameters
: **name** (*str*) -- name of the constant

Returns
: first value of the constant

Return type
: str

This function returns, as a string of character, the value associated to the constant *name* in the header of the ephemeris file *self*. Only the first value is returned if multiple values are associated to a constant, such as a list of values. The values must be strings or dates or numbers. For the original INPOP/DE files, the floating-points numbers are expressed using a scientific notation in base 10.

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
UNIT = peph.getconstantss("UNIT")
print(UNIT)
peph.close()
```

4.4.15 calcephpy.CalcephBin.getconstantvs

`calcephpy.CalcephBin.`**`getconstantvs`**`(`*name*`)` → arrayvalue

Parameters
: **name** (*str*) -- name of the constant

Returns
: array of values for the constant

Return type
: list

This function returns, as strings of characters, all values associated to the constant *name* in the header of the ephemeris file *self*.

The values must be strings or dates or numbers. For the original INPOP/DE files, the floating-points numbers are expressed using a scientific notation in base 10.

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
mission_units = peph.getconstantvs("MISSION_UNITS")
```

(continues on next page)

(continued from previous page)

```
print(mission_units)
peph.close()
```

4.4.16 calcephpy.CalcephBin.getconstantcount

calcephpy.CalcephBin.**getconstantcount**()

Returns
number of constants

Return type
int

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

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
n = peph.getconstantcount()
print("number of constants", n)
peph.close()
```

4.4.17 calcephpy.CalcephBin.getconstantindex

calcephpy.CalcephBin.**getconstantindex**(*index*) → name, value

Parameters
index (*int*) -- index of the constant, between 1 and *calcephpy.CalcephBin.getconstantcount()*

Returns
name of the constant, first value of the constant

Return type
str, float

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

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

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
n = peph.getconstantcount()
for j in range(1, n+1):
    name, value = peph.getconstantindex(j)
    print(name, value)

peph.close()
```

4.4.18 calcephpy.CalcephBin.getfileversion

calcephpy.CalcephBin.**getfileversion**()

Returns
version of the ephemeris file

Return type
str

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

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

The following example prints the version of the ephemeris file.

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
version = peph.getfileversion()
print(version)
peph.close()
```

4.4.19 calcephpy.CalcephBin.getidbyname

calcephpy.CalcephBin.**getidbyname**(*name*, *unit*) → id

Parameters

- **name** (*str*) -- name of the body
- **unit** (*int*) --

  The unit of id.

  This integer is 0 or the constant *USE_NAIFID*.

  If the unit equals *USE_NAIFID*, the NAIF identification numbering system is used for the id (*NAIF identification numbers* for the list).

  If the unit equals 0, the old number system is used for the idr (see the list in the function `calcephpy.CalcephBin.compute()`).

Returns
identification number of the body

Return type
int or None

This function returns the identification number of the body associated to the given name. If no such name exists in the mapping, then the value *None* is returned.

The function is not sensitive to the case of the characters of the string *name*. Leading and trailing spaces are ignored. At least one space character is required between the words. For example, 'Saturn Barycenter' is the same as ' Saturn Barycenter ', but is different from 'SaturnBarycenter'.

The library has a default mapping listed in the list *NAIF identification numbers*. The mapping name/id may be overriden by the text constants *NAIF_BODY_CODE* and *NAIF_BODY_NAME* in the 'SPICE' ephemeris files.

The following example prints the identification number of the Moon.

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
# print the id of the Moon using the old numbering system
moon = peph.getidbyname('Moon', 0)
print(moon)
# print the id of the Moon using the NAIF identification numbering system
moon = peph.getidbyname('Moon', Constants.USE_NAIFID)
print(moon)
print(NaifId.MOON)
peph.close()
```

4.4.20 calcephpy.CalcephBin.getnamebyidss

`calcephpy.CalcephBin.`**`getnamebyidss`**`(`*id*, *unit*`)` → name

Parameters

- **id** (*int*) -- identification number of the body
- **unit** (*int*) --

  The unit of id.

  This integer is 0 or the constant *USE_NAIFID*.

  If the unit equals *USE_NAIFID*, the NAIF identification numbering system is used for the id (*NAIF identification numbers* for the list).

  If the unit equals 0, the old number system is used for the idr (see the list in the function *calcephpy.CalcephBin.compute()*).

Returns
name of the body

Return type
str or None

This function returns the first given name of the body associated to the identification number *id*. If no such name exists in the mapping, then the value *None* is returned.

The library has a default mapping listed in the list *NAIF identification numbers*. The mapping name/id may be overriden by the text constants *NAIF_BODY_CODE* and *NAIF_BODY_NAME* in the 'SPICE' ephemeris files.

The following example prints the name of the Earth-Moon barycenter.

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
#  print the name of the Earth-Moon barycenter using the old numbering system
emb = peph.getnamebyidss(13, 0)
print(emb)
# print the name of the Earth-Moon barycenter using the NAIF identification numbering␣
↪system
emb = peph.getnamebyidss(3, Constants.USE_NAIFID)
print(emb)
peph.close()
```

4.4.21 calcephpy.CalcephBin.gettimescale

`calcephpy.CalcephBin.gettimescale()`

Returns
time scale of the ephemeris file

Return type
int

This function returns the time scale of the ephemeris file *self* :

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

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
timescale = peph.gettimescale()
print(timescale)
peph.close()
```

4.4.22 calcephpy.CalcephBin.gettimespan

`calcephpy.CalcephBin.gettimespan()` → firsttime, lasttime, continuous

Returns
first and last available time, availability of the quantities of the bodies over the time span

Return type
float, float, int

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

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

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

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
firsttime, lasttime, continuous = peph.gettimespan()
print(firsttime, lasttime, continuous)
peph.close()
```

4.4.23 calcephpy.CalcephBin.getpositionrecordcount

calcephpy.CalcephBin.**getpositionrecordcount**()

Returns
number of position's records

Return type
int

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

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
n = peph.getpositionrecordcount()
print("number of position's record", n)
peph.close()
```

4.4.24 calcephpy.CalcephBin.getpositionrecordindex

calcephpy.CalcephBin.**getpositionrecordindex**(*index*) → target, center, firsttime, lasttime, frame

Parameters
index (*int*) -- index of the position's record, between 1 and *calcephpy.CalcephBin.getpositionrecordcount()*

Returns

target : the target body
center : the origin body
firsttime : julian date of the first time
lasttime : julian date of the last time
frame : reference frame (see the list, below)

Return type
int, int, float, float, int

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

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

value	Name
1	ICRF

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
n = peph.getpositionrecordcount()
for j in range(1, n+1):
    itarget, icenter, firsttime, lasttime, iframe = peph.getpositionrecordindex(j)
    print(itarget, icenter, firsttime, lasttime, iframe)

peph.close()
```

4.4.25 calcephpy.CalcephBin.getpositionrecordindex2

`calcephpy.CalcephBin.`**`getpositionrecordindex2`**(*index*) → target, center, firsttime, lasttime, frame, segid

Parameters

index (*int*) -- index of the position's record, between 1 and *calcephpy.CalcephBin.getpositionrecordcount()*

Returns

target : the target body
center : the origin body
firsttime : julian date of the first time
lasttime : julian date of the last time
frame : reference frame (see the list, below)
segid : segment type (see the details, below)

Return type

int, int, float, float, int, int

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

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

value	Name
1	ICRF

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

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
n = peph.getpositionrecordcount()
for j in range(1, n+1):
    itarget, icenter, firsttime, lasttime, iframe, iseg = peph.getpositionrecordindex2(j)
```

(continues on next page)

(continued from previous page)

```
    print(itarget, icenter, firsttime, lasttime, iframe, iseg)

peph.close()
```

4.4.26 calcephpy.CalcephBin.getorientrecordcount

calcephpy.CalcephBin.**getorientrecordcount**()

Returns
number of orientation's records

Return type
int

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

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
n = peph.getorientrecordcount()
print("number of orientation's record", n)
peph.close()
```

4.4.27 calcephpy.CalcephBin.getorientrecordindex

calcephpy.CalcephBin.**getorientrecordindex**(*index*) → target, firsttime, lasttime, frame

Parameters
index (*int*) -- index of the orientation's record, between 1 and *calcephpy.CalcephBin.getorientrecordcount()*

Returns

target : the target body

firsttime : julian date of the first time

lasttime : julian date of the last time

frame : reference frame (see the list, below)

Return type
int, float, float, int

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

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

value	Name
1	ICRF

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
n = peph.getorientrecordcount()
for j in range(1, n+1):
    itarget, firsttime, lasttime, iframe = peph.getorientrecordindex(j)
    print(itarget, firsttime, lasttime, iframe)

peph.close()
```

4.4.28 calcephpy.CalcephBin.getorientrecordindex2

`calcephpy.CalcephBin.`**`getorientrecordindex2`**`(`*index*`)` → target, firsttime, lasttime, frame, segid

Parameters
index (*int*) -- index of the orientation's record, between 1 and *calcephpy.CalcephBin.getorientrecordcount()*

Returns

target : the target body
firsttime : julian date of the first time
lasttime : julian date of the last time
frame : reference frame (see the list, below)
segid : segment type (see the details, below)

Return type
int, float, float, int, int

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

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

value	Name
1	ICRF

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

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

```
from calcephpy import *

peph = CalcephBin.open("example1.dat")
n = peph.getorientrecordcount()
for j in range(1, n+1):
    itarget, firsttime, lasttime, iframe, iseg = peph.getorientrecordindex2(j)
    print(itarget, firsttime, lasttime, iframe, iseg)

peph.close()
```

4.4.29 calcephpy.CalcephBin.close

`calcephpy.CalcephBin.close()`

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

CHAPTER

FIVE

TIME CONVERSION FUNCTIONS

The following group of functions allow to convert dates between different timescales (UTC, TAI, TT, TDB, TCB) and formats (Julian date, calendar date). Some of these functions accept time strings as input, which are parsed internally.

When an error occurs, these functions execute error handlers according to the behavior defined by the function `calcephpy.seterrorhandler()`.

5.1 Timescale notes

Conversions involving the Coordinated Universal Time (UTC) rely on the history of leap seconds. Consequently, these functions require an ephemeris file containing the time constants (usually DELTET or DELTA_AT) to be opened and provided to the function . These time constants are provided in the leap second spice kernel file (*.tls).

Conversions the Barycentric Dynamical Time (TDB) and the Terrestrial Time (TT) are performed using the model chosen with the `calcephpy.CalcephBin.time_set_relationship_tt_tdb()` function.

5.2 Input string formats

For functions accepting time strings as input, the library supports various formats, including ISO 8601, explicit Julian dates, and custom calendar strings.

If the time scale is specified within the string (e.g., "1996-01-01 UTC"), it is automatically detected. If the time scale is not present in the string, it is considered as UTC time scale.

The following strings show some examples of supported strings

```
"2006-01-15T12:01:10.00 UTC"
"2006-01-15T12:01:10.00 TDB"
"1995 December 31  15:59:59.5  (UTC)"
"1988 June 13, 12:29:48 TDB"
"1988 June 13, 12:29:48 TT"
"JD 2453751.0008101849816740 (TAI)"
"JD 2453751.0011826851405203 (TT)"
"JD 2453751.0013471459969878 (TCB)"
"2451515.2981 (JD)"
```

5.3 Functions

5.3.1 calcephpy.CalcephBin.time_jd_to_cal

calcephpy.CalcephBin.**time_jd_to_cal**(*timescale*, *jd0*, *jdfrac*) → yy, month, day, ihh, min, sec

Parameters

- **timescale** (*int*) -- timescale of the input date and result (see *Timescale constant* for the available timescales)
- **jd0** (*int*) -- Integer part of Julian Date
- **jdfrac** (*int*) -- Fractional part of Julian Date

Returns
Year, Month [1–12], Day [1–31], Hour [0–23], Minute [0–59], seconds [0.0–61.0[

Return type
int, int, int, int, int, float

This function converts a Julian Date, expressed as two double-precision numbers, into a Gregorian calendar date (year, month, day) and time (hour, minute, second).

Depending on the value of the timescale argument, the function behaves as follows:

- *UTC*: The function handles leap seconds. It requires that the ephemeris file *self* contains leap second history (constants *DELTET* or *DELTA_AT*).
- Other timescales (e.g., *TT*, *TDB*): The time is treated as continuous. The seconds field is always in the range [0, 60[.

If the timescale is *UTC* and the ephemeris file does not contain the required leap second constants, the function returns an error.

The following example converts a JD to a calendar date in TDB:

```
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
peph.time_set_relationship_tt_tdb(1)
# convert the julian day 2457754.499994212962963 UTC (leap second) to the calendar date␣
↪UTC
yy, mo, dd, hh,mi, ss = peph.time_jd_to_cal(Constants.UTC, 2457754, 0.499994212962963)
print(yy, mo, dd, hh,mi, ss) # print 2016 12 31 23 59 60.49999421295723
peph.close()
```

5.3.2 calcephpy.CalcephBin.time_cal_to_jd

calcephpy.CalcephBin.**time_cal_to_jd**(*timescale*, *yy*, *month*, *day*, *hh*, *min*, *sec*) → jd0, jdfrac

Parameters

- **timescale** (*int*) -- timescale of the input date and result (see *Timescale constant* for the available timescales)
- **yy** (*int*) -- Year
- **month** (*int*) -- Month [1–12]
- **day** (*int*) -- Day [1–31]
- **hh** (*int*) -- Hour [0–23]
- **min** (*int*) -- Minute [0–59]
- **sec** (*float*) -- Seconds [0.0–61.0[

Returns
Integer part of Julian Date, Fractional part of Julian Date

Return type
float, float

This function converts a Gregorian calendar date (year, month, day) and time (hour, minute, second) into a Julian Date, expressed as two double-precision floating-point numbers.

The resulting Julian Date is the sum of the two returned values jd0 and jdfrac. This split representation preserves numerical precision.

Depending on the value of the timescale argument, the function performs the conversion as follows:

- *UTC*: The function accounts for leap seconds. It calculates the Julian Date by checking the leap second history stored in the ephemeris file (eph).
  - If the input date corresponds to a leap second (e.g., 23:59:60), the function correctly computes the JD for that instant.
  - The function requires the ephemeris file to contain standard time constants (*DELTET* or *DELTA_AT*).
- Other timescales (e.g., *TT*, *TDB*): The time is treated as continuous (without leap seconds). The minutes are assumed to always contain 60 seconds.

If timescale is *UTC* and the necessary leap second data is missing from the ephemeris file, the function returns an error.

The following example converts a calendar date to a Julian Date:

```
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
# convert 2025-01-03T22:59:50.300 UTC to the julian day UTC
jd0, jdfrac = peph.time_cal_to_jd(Constants.UTC, 2025, 1, 3, 22, 59, 50.3)
print(jd0, jdfrac) # print 2460679.0 0.458221064814814 84
peph.close()
```

5.3.3 calcephpy.CalcephBin.time_str_to_jd

calcephpy.CalcephBin.**time_str_to_jd**(*timescale*, *str*) → jd0, jdfrac

Parameters

- **timescale** -- timescale of the input date and result (see *Timescale constant* for the available timescales)
- **str** -- Input time string to be parsed

Returns
Integer part of Julian Date, Fractional part of Julian Date

Return type
float, float

This function parses a time string and converts it into a Julian Date, returned as two double-precision numbers (jd0 and jdfrac).

This function supports various input string formats, including ISO 8601, explicit Julian Dates (prefixed with "JD"), and standard calendar formats.

The interpretation of the date depends on the timescale argument:

- Specific Timescale (e.g., *UTC*, *TT*): The function interprets the date components parsed from the string as being in this specific timescale, regardless of any timescale suffix present in the string itself.

- *TIMESCALE_FROM_STR*: The function attempts to detect the timescale from the string (e.g., "2000-01-01 TDB").
  - If a timescale is found in the string, it is used for the conversion.
  - If no timescale is found, the function defaults to UTC.

If the resulting conversion requires UTC (either explicitly requested or detected), the ephemeris handle *self* must contain leap second data.

The following example parses a string and converts it to a Julian Date:

```
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
peph.time_set_relationship_tt_tdb(1)
# convert the calendar date TAI to the julian day TAI
jd0, jdfrac = peph.time_str_to_jd(Constants.TAI, "2025-01-03T22:59:50.300")
print(jd0, jdfrac) # print 2460679.0 0.45822106481481484
peph.close()
```

5.3.4 calcephpy.CalcephBin.time_set_relationship_tt_tdb

calcephpy.CalcephBin.**time_set_relationship_tt_tdb**(*model*)

Parameters

model --

Relationship model

0: use of *calcephpy.CalcephBin.compute_unit()*

1: use the model based on TLS file

This function sets the mathematical model or data source used to convert dates between the Terrestrial Time (TT) and Barycentric Dynamical Time (TDB) scales.

The selected model affects the behavior of the conversion functions *calcephpy.CalcephBin.time_jd_tdb_to_jd_tt()* and *calcephpy.CalcephBin.time_jd_tt_to_jd_tdb()*.

The supported values for the model argument are:

- 0: The conversion uses *calcephpy.CalcephBin.compute_unit()* to retrieve the time difference. The difference TT-TDB should be available in the ephemeris file.
- 1: The conversion uses a default model based on the loaded TLS file.

If an invalid model is specified, the function returns an error.

The following example sets the relationship model to use the binary ephemeris data:

```
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
peph.time_set_relationship_tt_tdb(1)
jd0, jdfrac = peph.time_jd_tt_to_jd_tdb(2457754.0, 0.5)
peph.close()
```

5.3.5 calcephpy.CalcephBin.time_jd_tdb_to_jd_tt

calcephpy.CalcephBin.**time_jd_tdb_to_jd_tt**(*jd0_tdb*, *jdfrac_tdb*) → jd0_tt, jdfrac_tt

Parameters

- **jd0_tdb** -- Integer part of Julian Date (TDB)

- **jdfrac_tdb** -- Fractional part of Julian Date (TDB)

Returns
Integer part of Julian Date (TT), Fractional part of Julian Date (TT)

Return type
float, float

This function converts a Julian Date from the Barycentric Dynamical Time (TDB) scale to the Terrestrial Time (TT) scale.

The conversion relies on the model previously set by the function *calcephpy.CalcephBin.time_set_relationship_tt_tdb()*.

The input Julian Date is provided as two double-precision numbers (jd0_tdb and jdfrac_tdb) to maintain precision. The result is returned in the output variables jd0_tt and jdfrac_tt.

Depending on the selected model:

- Model 0: Retrieves the offset with *calcephpy.CalcephBin.compute_unit()*.
- Model 1: Uses the internal default model from the TLS file.

The following example converts a TDB date to TT:

```
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
peph.time_set_relationship_tt_tdb(1)
# convert the julian day 2460679.45902180578559637 TDB to the julian day TT
jd0, jdfrac = peph.time_jd_tdb_to_jd_tt(2460679.0, 0.45902180578559637)
print(jd0, jdfrac) # print 2460679.0 0.4590218056322423
peph.close()
```

5.3.6 calcephpy.CalcephBin.time_jd_tt_to_jd_tdb

`calcephpy.CalcephBin.`**`time_jd_tt_to_jd_tdb`**`(`*jd0_tt*, *jdfrac_tt*`)` → jd0_tdb, jdfrac_tdb

Parameters

- **jd0_tt** -- Integer part of Julian Date (TT)
- **jdfrac_tt** -- Fractional part of Julian Date (TT)

Returns
Integer part of Julian Date (TDB), Fractional part of Julian Date (TDB)

Return type
float, float

This function converts a Julian Date from the Terrestrial Time (TT) scale to the Barycentric Dynamical Time (TDB) scale.

This is the inverse operation of *calcephpy.CalcephBin.time_jd_tdb_to_jd_tt()*. It uses the model configured with the function *calcephpy.CalcephBin.time_set_relationship_tt_tdb()*.

The input Julian Date is provided as two double-precision numbers (jd0_tt and jdfrac_tt). The converted date is stored in jd0_tdb and jdfrac_tdb.

If the ephemeris handle is NULL or if the model configuration is invalid, the function returns an error.

The following example converts a TT date to TDB:

```python
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
peph.time_set_relationship_tt_tdb(1)
# convert the julian day 2460679.4590218056322423 TDB to the julian day TT
jd0, jdfrac = peph.time_jd_tt_to_jd_tdb(2460679.0, 0.4590218056322423)
print(jd0, jdfrac) # print 2460679.0 0.45902180578559637
peph.close()
```

5.3.7 calcephpy.CalcephBin.time_jd_tdb_to_jd_tcb

calcephpy.CalcephBin.**time_jd_tdb_to_jd_tcb**(*jd0_tdb*, *jdfrac_tdb*) → jd0_tcb, jdfrac_tcb

Parameters

- **jd0_tdb** -- Integer part of Julian Date (TDB)
- **jdfrac_tdb** -- Fractional part of Julian Date (TDB)

Returns
Integer part of Julian Date (TCB), Fractional part of Julian Date (TCB)

Return type
float, float

This function converts a Julian Date from the Barycentric Dynamical Timescale (TDB) to the Barycentric Coordinate Timescale (TCB).

The conversion relies on the relation between TCB and TDB defined by the IAU 2006 Resolution B3 : "Re-definition of Barycentric Dynamical Time, TDB".

The input Julian Date is provided as two double-precision numbers (jd0_tdb and jdfrac_tdb) to maintain precision. The result is returned in the output variables jd0_tcb and jdfrac_tcb.

The following example converts a TDB date to TCB:

```python
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
# convert the julian day 2450083.1267361110076308 TDB to the julian day TCB
jd0, jdfrac = peph.time_jd_tdb_to_jd_tcb(2450083.0, 0.1267361110076308)
print(jd0, jdfrac) # print 2450083.0 0.1268436964601278
peph.close()
```

5.3.8 calcephpy.CalcephBin.time_jd_tcb_to_jd_tdb

calcephpy.CalcephBin.**time_jd_tcb_to_jd_tdb**(*jd0_tcb*, *jdfrac_tcb*) → jd0_tdb, jdfrac_tdb

Parameters

- **jd0_tcb** -- Integer part of Julian Date (TCB)
- **jdfrac_tcb** -- Fractional part of Julian Date (TCB)

Returns
Integer part of Julian Date (TDB), Fractional part of Julian Date (TDB)

Return type
float, float

This function converts a Julian Date from the Barycentric Coordinate Timescale (TCB) to the Barycentric Dynamical Timescale (TDB) .

This is the inverse operation of *calcephpy.CalcephBin.time_jd_tdb_to_jd_tcb()*.

The conversion relies on the relation between TCB and TDB defined by the IAU 2006 Resolution B3 : "Re-definition of Barycentric Dynamical Time, TDB".

The input Julian Date is provided as two double-precision numbers (jd0_tcb and jdfrac_tcb). The converted date is stored in jd0_tdb and jdfrac_tdb.

If the ephemeris handle is NULL, the function returns an error.

The following example converts a TCB date to TDB:

```
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
# convert the julian day 2450083.1268436964601278 TCB to the julian day TDB
jd0, jdfrac = peph.time_jd_tcb_to_jd_tdb(2450083.0, 0.1268436964601278)
print(jd0, jdfrac) # print 22450083.0  0.1267361110076308
peph.close()
```

5.3.9 calcephpy.CalcephBin.time_cal_utc_to_jd_tdb

`calcephpy.CalcephBin.`**`time_cal_utc_to_jd_tdb`**(*yy*, *month*, *day*, *hh*, *min*, *sec*) → jd0_tdb, jdfrac_tdb

param yy
Year

param month
Month

param day
Day

param hh
Hour

param min
Minute

param sec
Second

return
Integer part of Julian Date (TDB), Fractional part of Julian Date (TDB)

rtype
float, float

Returns
0 on success, 1 on error

This function converts a Coordinated Universal Time (UTC) calendar date (year, month, day, hour, minute, second) into a Barycentric Dynamical Time (TDB) Julian Date.

The result is returned as two double-precision floating-point numbers (jd0_tdb and jdfrac_tdb) to preserve precision.

This function performs the following transformation chain internally:

1. Converts the UTC calendar date to a UTC Julian Date.

2. Computes the difference between UTC and TT (Terrestrial Time) using leap second data.
3. Converts TT to TDB using *calcephpy.CalcephBin.time_jd_tt_to_jd_tdb()*.

This function requires that the ephemeris file *self* contains both leap second constants (for UTC → TT) and the necessary data for the TT → TDB transformation that can be initialized with *calcephpy.CalcephBin.time_set_relationship_tt_tdb()*.

The following example converts a UTC calendar date to TDB:

```
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
peph.time_set_relationship_tt_tdb(1)
# convert 2025-01-03T22:59:50.300 UTC to the julian day TDB
jd0, jdfrac = peph.time_cal_utc_to_jd_tdb(2025, 1, 3, 22, 59, 50.3)
print(jd0, jdfrac) # print 2460679.0 0.459021805785596 37
peph.close()
```

5.3.10 calcephpy.CalcephBin.time_jd_tdb_to_cal_utc

calcephpy.CalcephBin.**time_jd_tdb_to_cal_utc**(*jd0_tdb*, *jdfrac_tdb*) → yy, month, day, hh, min, sec

Parameters

- **jd0_tdb** -- Integer part of the TDB Julian Date
- **jdfrac_tdb** -- Fractional part of the TDB Julian Date

Returns

Year, Month [1–12], Day [1–31], Hour [0–23], Minute [0–59], seconds [0.0–61.0[

Return type

int, int, int, int, int, float

This function converts a TDB Julian Date into a UTC calendar date (year, month, day, hour, minute, second).

This is the inverse operation of *calcephpy.CalcephBin.time_cal_utc_to_jd_tdb()*.

The transformation chain performed is:

1. Converts TDB to TT using *calcephpy.CalcephBin.time_jd_tdb_to_jd_tt()*.
2. Computes the difference between TT and UTC (using leap seconds).
3. Converts the resulting UTC Julian Date into calendar components.

The function requires that the ephemeris file *self* contains the necessary time constants.

The following example converts a TDB Julian Date to a UTC calendar date:

```
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
peph.time_set_relationship_tt_tdb(1)
# convert the julian day 2460679.45902180578559637 TDB to the calendar date UTC
yy, mo, dd, hh,mi, ss = peph.time_jd_tdb_to_cal_utc(2460679.0, 0.45902180578559637)
print(yy, mo, dd, hh,mi, ss) # print 2025 1 3 22 59 50.30000662574196
peph.close()
```

5.3.11 calcephpy.CalcephBin.time_str_utc_to_jd_tdb

`calcephpy.CalcephBin.`**`time_str_utc_to_jd_tdb`**(*str*) → jd0_tdb, jdfrac_tdb

Parameters
`str` -- UTC time string to parse

Returns
Integer part of Julian Date (TDB), Fractional part of Julian Date (TDB)

Return type
float, float

This function parses a time string representing a UTC date and converts it directly to a TDB Julian Date.

The function forces the interpretation of the input string as UTC, ignoring any potential timescale suffix present in the string. It then performs the full conversion chain (UTC → TAI → TT → TDB).

The following example converts a UTC string to TDB:

```
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
peph.time_set_relationship_tt_tdb(1)
# convert the calendar date UTC to the julian day TDB
jd0, jdfrac = peph.time_str_utc_to_jd_tdb("2025-01-03T22:59:50.300")
print(jd0, jdfrac) # print 2460679.0 0.45902180578559637
peph.close()
```

5.3.12 calcephpy.CalcephBin.time_str_any_to_jd_tdb

`calcephpy.CalcephBin.`**`time_str_any_to_jd_tdb`**(*str*) → jd0_tdb, jdfrac_tdb

Parameters
`str` -- Time string to parse

Returns
Integer part of Julian Date (TDB), Fractional part of Julian Date (TDB)

Return type
float, float

This function parses a time string, identifies its native time scale, and converts the date directly to the TDB scale.

The function first parses the string to extract the date and the time scale (e.g., "2000-01-01 TAI"). Based on the detected time scale, it performs the necessary conversions to reach TDB.

The supported input timescales and their internal conversion paths are:

- *`TDB`*: No conversion needed.
- *`TT`*: Performs the transformation TT → TDB.
- *`TAI`*: Performs the transformation chain TAI → TT → TDB.
- *`UTC`*: Performs the transformation chain UTC → TAI → TT → TDB.

If the input string does not specify a time scale, the function defaults to *`UTC`* before converting to TDB.

This function requires that the ephemeris file *self* contains the necessary data for the requested conversions (leap seconds for UTC/TAI, and relationship model for TT/TDB that can be initialized with *`calcephpy.CalcephBin.time_set_relationship_tt_tdb()`*).

The following example converts a TAI string to TDB:

```python
from calcephpy import *
peph = CalcephBin.open("example_lsk.tls")
peph.time_set_relationship_tt_tdb(1)
# convert the calendar date UTC to the julian day TDB
jd0, jdfrac = peph.time_str_any_to_jd_tdb("2025-01-03T22:59:50.300 UTC")
print(jd0, jdfrac) # print 2460679.0 0.45902180578559637
peph.close()
```

5.3.13 calcephpy.CalcephBin.time_str_spacecraft_clock_to_jd_tdb

calcephpy.CalcephBin.**time_str_spacecraft_clock_to_jd_tdb**(*target*, *strdate*) → jd0_tdb, jdfrac_tdb

Parameters

- **target** -- NAIF ID of the spacecraft
- **strdate** -- Spacecraft clock string (e.g., "1/1234:56")

Returns
Integer part of Julian Date (TDB), Fractional part of Julian Date (TDB)

Return type
float, float

This function converts a spacecraft clock string (SCLK) to the Barycentric Dynamical Time (TDB) scale.

The input spacecraft clock string is provided in the parameter *str*. The converted date is stored in *jd0_tdb* and *jdfrac_tdb*.

It is strictly required to load a **Spacecraft Clock kernel** (usually `.tsc`) to define the clock partitions and coefficients.

Since this function performs an internal conversion between Terrestrial Time (TT) and Barycentric Dynamical Time (TDB), additional kernels are required depending on the configuration set by *calcephpy.CalcephBin.time_set_relationship_tt_tdb()*:

- If the relationship mode is **0** (default), an **Ephemeris kernel** (usually `.bsp`) containing the ntime tranformation TT-TDB must be loaded.
- If the relationship mode is **1**, a **Leap Seconds kernel** (usually `.tls`) must be loaded.

If the required kernels are missing or if the string format is invalid, then the function returns an error.

The following example converts a spacecraft clock string to TDB:

```python
from calcephpy import *
peph = CalcephBin.open(['example_lsk.tls', 'example_sclk.tsc'])
peph.time_set_relationship_tt_tdb(1)
jd0, jdfrac = peph.time_str_spacecraft_clock_to_jd_tdb(28, "1/0707109102:11026")
print(jd0, jdfrac) # print 2459728.0 0.633957315236 33
peph.close()
```

5.3.14 calcephpy.CalcephBin.time_jd_tdb_to_str_spacecraft_clock

calcephpy.CalcephBin.**time_jd_tdb_to_str_spacecraft_clock**(*eph*, *target*, *jd0_tdb*, *jdfrac_tdb*)

Parameters

- **eph** -- ephemeris descriptor
- **target** -- NAIF ID of the spacecraft
- **jd0_tdb** -- Integer part of Julian Date (TDB)

- `jdfrac_tdb` -- Fractional part of Julian Date (TDB)

Returns
Output spacecraft clock time string

Return type
str

Returns
returns 0 or an exception if an error occurs, otherwise a non-zero value.

This function computes the spacecraft clock string corresponding to a given Barycentric Dynamical Time (TDB).

This is the inverse operation of *calcephpy.CalcephBin.time_str_spacecraft_clock_to_jd_tdb()*. The input Julian Date is provided as two double-precision numbers (*jd0_tdb* and *jdfrac_tdb*).

It is strictly required to load a **Spacecraft Clock kernel** (usually `.tsc`) to define the clock partitions and coefficients.

Since this function performs an internal conversion between Barycentric Dynamical Time (TDB) and Terrestrial Time (TT), additional kernels are required depending on the configuration set by *calcephpy.CalcephBin.time_set_relationship_tt_tdb()*:

- If the relationship mode is **0** (default), an **Ephemeris kernel** (usually `.bsp`) containing the ntime tranformation TT-TDB must be loaded.
- If the relationship mode is **1**, a **Leap Seconds kernel** (usually `.tls`) must be loaded.

If the time falls outside the range covered by the SCLK coefficients, then the function returns an error.

The following example converts a TDB date to a spacecraft clock string:

```
from calcephpy import *
peph = CalcephBin.open(['example_lsk.tls', 'example_sclk.tsc'])
peph.time_set_relationship_tt_tdb(1)
strdate = peph.time_jd_tdb_to_str_spacecraft_clock(28, 2459728, 0.63395731542095745681)
print(strdate) # print "1/0707109102:11026"
peph.close()
```

CHAPTER

SIX

INSTRUMENT FUNCTIONS

The instrument functions provide access to the description and the properties of instruments defined in the ephemeris files.

6.1 calcephpy.CalcephBin.getfov

`calcephpy.CalcephBin.getfov(instrumentid)` → shape, frame, vector, arraybounds

Parameters
  instrumentid (*int*) -- Instrument identifier

Returns
  FOV shape, Reference frame name, Boresight vector, Array of boundary vectors (3 components)

Return type
  int, str, np.array(3,dtype=float64), np.array(3, nbounds, dtype=float64)

This function retrieves the instrument Field Of View (FOV) description from an instrument kernel (IK) of the ephemeris file *self*, and computes the boundary vectors expressed in the instrument frame.

Depending on the FOV specification found in the IK, the function reads a subset of the following keywords (example names use the NAIF convention *INS<id>_...*):

- *INS#_FOV_SHAPE* — shape: *CIRCLE*, *ELLIPSE*, *RECTANGLE*, *POLYGON*.
- *INS#_FOV_FRAME* — name of the reference frame used for the boresight and boundary vectors.
- *INS#_BORESIGHT* — the boresight vector (3 components).
- *INS#_FOV_CLASS_SPEC* — either *ANGLES* or *CORNERS*.

If *FOV_CLASS_SPEC* = *ANGLES* additional keywords are expected:

- *INS#_FOV_REF_VECTOR* — reference vector defining the reference axis in the boresight plane.
- *INS#_FOV_REF_ANGLE* — half angular extent along the reference axis.
- *INS#_FOV_CROSS_ANGLE* — half angular extent along the axis orthogonal to the reference axis.
- *INS#_FOV_ANGLE_UNITS* — angle units (*DEGREES*, *RADIANS*).

If *FOV_CLASS_SPEC* = *CORNERS* the IK must provide explicit boundary corner vectors:

- *INS#_FOV_BOUNDARY_CORNERS* — array of 3-component vectors giving the boundary corners.

Difference between ANGLES and CORNERS:

- **ANGLES**: The IK describes the FOV by angular half-extents and a reference vector. Boundary vectors are computed from these values. Returned vectors share the same magnitude as the boresight.

- **CORNERS**: The IK provides explicit boundary vectors, which are returned exactly as given. Magnitudes are preserved.

Coordinate conventions and units:

- All vectors are expressed in the frame *INS#_FOV_FRAME*.
- Angles may be in *DEGREES* or *RADIANS*; converted internally to radians.
- For *ANGLES*, the boundary vectors preserve boresight magnitude. For *CORNERS*, the magnitudes are taken as-is.

For the **ANGLES** specification, the following formula is used internally to compute FOV boundary vectors. It describes how to rotate a vector u inside the plane defined by two vectors u and v, starting from u and turning it by a given angle θ:

$$\mathbf{u}' = \|\mathbf{u}\| \left(\cos\theta \, \frac{\mathbf{u}}{\|\mathbf{u}\|} + \sin\theta \, \mathbf{Z} \right)$$

where $\mathbf{Z}$ is the normalized projection of v onto the plane orthogonal to u.

$$\mathbf{Z} = \frac{\mathbf{v} - \dfrac{(\mathbf{v} \cdot \mathbf{u})}{\|\mathbf{u}\|^2} \mathbf{u}}{\left\| \mathbf{v} - \dfrac{(\mathbf{v} \cdot \mathbf{u})}{\|\mathbf{u}\|^2} \mathbf{u} \right\|}$$

This rotation is the core operation used when computing the boundary vectors of a FOV.

For the **CORNERS** specification, vectors are returned as given in the IK.

The possible values for **shape** are :

value	meaning
1	Polygon
2	Rectangle
3	Circle
4	Ellipse

The following example prints the shape, the frame, and the boundary vector for a circular FOV

```
from calcephpy import *
peph = CalcephBin.open("example_ik.ti")
instrumentid = -42550
shape, frame, boresight_vector, boundary_vectors = peph.getfov(instrumentid)
print("shape=",shape) # print shape= 3
print("frame=",frame) # print frame= EXAMPLE_CIRCLE
print("boresight_vector=",boresight_vector) # print boresight_vector= [0.0, 0.0, 25.0]
print("boundary_vectors=",boundary_vectors) # print boundary_vectors= [-3.
↪046733585128687, 0.0, 24.813653791033305]
peph.close()
```

CHAPTER SEVEN

ERROR FUNCTIONS

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

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 Python language is `pyerror.py`.

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

```
from calcephpy import *

#set the  error handler to stop on error
seterrorhandler(2, 0);

# open the ephemeris file
peph = CalcephBin.open("example1.dat")
```

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

```
from calcephpy import *

#-----------------------------------------------------------------
# custom error handler
#-----------------------------------------------------------------
def myhandler(msg):
    print("The calceph calls the function myhandler");
    print("The message contains {0} characters\n".format(len(msg)))
    print("The error message is :")
    print("----------------------")
    print(msg)
    print("----------------------")
    print("The error handler returns")

# set the  error handler to use my own callback
seterrorhandler(3, myhandler)

# open the ephemeris file
peph = CalcephBin.open("example1.dat")
```

7.2 calcephpy.seterrorhandler

calcephpy.**seterrorhandler**(*typehandler*, *userfunc*)

Parameters

- **typehandler** (*int*) -- type of handler
- **userfunc** (*function*) -- user function

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

The possible values for *typehandler* are :

value	meaning
1	The library displays a message and continues the execution. The functions return an error code. The python and Octave/Matlab interfaces raise an exception. This is the default behavior of the library.
2	The library displays a message and terminates the execution with a system call to the function *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 *0*.

The function *userfunc* must be defined as

```
def userfunc (msg)
# parameter msg is of type str
```

CHAPTER

EIGHT

MISCELLANEOUS FUNCTIONS

8.1 calcephpy.getmaxsupportedorder

calcephpy.**getmaxsupportedorder**(*segid*)

Parameters
segid (*int*) -- type of the segment.

Returns
version of the library

Return type
str

This function returns the maximal order of the derivatives computed by the functions *calcephpy.CalcephBin.compute_order()*, *calcephpy.CalcephBin.orient_order()*, ... for the segment type *segid*. If the segment type is unknown by the library, the function returns -1.

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

```
from calcephpy import *
maxorder = getmaxsupportedorder(Constants.SEGTYPE_SPK_2)
print('maximal order is ', maxorder)
```

8.2 calcephpy.getversion_str

calcephpy.**getversion_str**()

Returns
version of the library

Return type
str

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

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

CHAPTER

NINE

NAIF IDENTIFICATION NUMBERS

The following predefined values must be used as the target body and origin of the coordinate system with the functions `calcephpy.CalcephBin.compute_unit()`, `calcephpy.CalcephBin.orient_unit()` `calcephpy.CalcephBin.compute_order()` or `calcephpy.CalcephBin.orient_order()` if and only if the value `USE_NAIFID` has been set in the parameter *unit*.

This list is already predefined in the class NaifId (*Types*) of the module calcephpy (*Modules*) for the Python 2/3 interface. Relative to C or Fortran interface, the prefix *NAIFID_* is deleted for the following numbers.

9.1 Sun and planetary barycenters

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

9.2 Coordinate Time ephemerides

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

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

9.3 Planet centers and satellites

Predefined Macros	NAIF ID	Name
NAIFID_MERCURY	199	Mercury
NAIFID_VENUS	299	Venus
NAIFID_EARTH	399	Earth
NAIFID_MOON	301	Moon
NAIFID_MARS	499	Mars
NAIFID_PHOBOS	401	Phobos
NAIFID_DEIMOS	402	Deimos
NAIFID_JUPITER	599	Jupiter
NAIFID_IO	501	Io
NAIFID_EUROPA	502	Europa
NAIFID_GANYMEDE	503	Ganymede
NAIFID_CALLISTO	504	Callisto
NAIFID_AMALTHEA	505	Amalthea
NAIFID_HIMALIA	506	Himalia
NAIFID_ELARA	507	Elara
NAIFID_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_MEGACLITE	519	Megaclite
NAIFID_TAYGETE	520	Taygete
NAIFID_CHALDENE	521	Chaldene
NAIFID_HARPALYKE	522	Harpalyke
NAIFID_KALYKE	523	Kalyke
NAIFID_IOCASTE	524	Iocaste
NAIFID_ERINOME	525	Erinome
NAIFID_ISONOE	526	Isonoe
NAIFID_PRAXIDIKE	527	Praxidike
NAIFID_AUTONOE	528	Autonoe
NAIFID_THYONE	529	Thyone
NAIFID_HERMIPPE	530	Hermippe
NAIFID_AITNE	531	Aitne
NAIFID_EURYDOME	532	Eurydome
NAIFID_EUANTHE	533	Euanthe
NAIFID_EUPORIE	534	Euporie
NAIFID_ORTHOSIE	535	Orthosie
NAIFID_SPONDE	536	Sponde

continues on next page

Table 1 – continued from previous page

Predefined Macros	NAIF ID	Name
NAIFID_KALE	537	Kale
NAIFID_PASITHEE	538	Pasithee
NAIFID_HEGEMONE	539	Hegemone
NAIFID_MNEME	540	Mneme
NAIFID_AOEDE	541	Aoede
NAIFID_THELXINOE	542	Thelxinoe
NAIFID_ARCHE	543	Arche
NAIFID_KALLICHORE	544	Kallichore
NAIFID_HELIKE	545	Helike
NAIFID_CARPO	546	Carpo
NAIFID_EUKELADE	547	Eukelade
NAIFID_CYLLENE	548	Cyllene
NAIFID_KORE	549	Kore
NAIFID_HERSE	550	Herse
NAIFID_DIA	553	Dia
NAIFID_SATURN	699	Saturn
NAIFID_MIMAS	601	Mimas
NAIFID_ENCELADUS	602	Enceladus
NAIFID_TETHYS	603	Tethys
NAIFID_DIONE	604	Dione
NAIFID_RHEA	605	Rhea
NAIFID_TITAN	606	Titan
NAIFID_HYPERION	607	Hyperion
NAIFID_IAPETUS	608	Iapetus
NAIFID_PHOEBE	609	Phoebe
NAIFID_JANUS	610	Janus
NAIFID_EPIMETHEUS	611	Epimetheus
NAIFID_HELENE	612	Helene
NAIFID_TELESTO	613	Telesto
NAIFID_CALYPSO	614	Calypso
NAIFID_ATLAS	615	Atlas
NAIFID_PROMETHEUS	616	Prometheus
NAIFID_PANDORA	617	Pandora
NAIFID_PAN	618	Pan
NAIFID_YMIR	619	Ymir
NAIFID_PAALIAQ	620	Paaliaq
NAIFID_TARVOS	621	Tarvos
NAIFID_IJIRAQ	622	Ijiraq
NAIFID_SUTTUNGR	623	Suttungr
NAIFID_KIVIUQ	624	Kiviuq
NAIFID_MUNDILFARI	625	Mundilfari
NAIFID_ALBIORIX	626	Albiorix
NAIFID_SKATHI	627	Skathi
NAIFID_ERRIAPUS	628	Erriapus
NAIFID_SIARNAQ	629	Siarnaq
NAIFID_THRYMR	630	Thrymr
NAIFID_NARVI	631	Narvi
NAIFID_METHONE	632	Methone
NAIFID_PALLENE	633	Pallene

continues on next page

Table 1 – continued from previous page

Predefined Macros	NAIF ID	Name
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

continues on next page

Table 1 – continued from previous page

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

9.4 Comets

Predefined Macros	NAIF ID	Name
NAIFID_AREND	1000001	Arend
NAIFID_AREND_RIGAUX	1000002	Arend-Rigaux
NAIFID_ASHBROOK_JACKSON	1000003	Ashbrook-Jackson
NAIFID_BOETHIN	1000004	Boethin
NAIFID_BORRELLY	1000005	Borrelly
NAIFID_BOWELL_SKIFF	1000006	Bowell-Skiff
NAIFID_BRADFIELD	1000007	Bradfield
NAIFID_BROOKS_2	1000008	Brooks 2
NAIFID_BRORSEN_METCALF	1000009	Brorsen-Metcalf
NAIFID_BUS	1000010	Bus
NAIFID_CHERNYKH	1000011	Chernykh
NAIFID_CHURYUMOV_GERASIM	1000012	Churyumov-Gerasimenko
NAIFID_CIFFREO	1000013	Ciffreo
NAIFID_CLARK	1000014	Clark
NAIFID_COMAS_SOLA	1000015	Comas Sola
NAIFID_CROMMELIN	1000016	Crommelin
NAIFID_D__ARREST	1000017	D''Drrest
NAIFID_DANIEL	1000018	Daniel
NAIFID_DE_VICO_SWIFT	1000019	De Vico-Swift
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_DEL	1000023	Dutoit-Neujmin-Delporte

continues on next page

Table 2 – continued from previous page

Predefined Macros	NAIF ID	Name
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_PAJDU:	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

continues on next page

Table 2 – continued from previous page

Predefined Macros	NAIF ID	Name
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_WACH	1000079	Schwassmann-Wachmann 1
NAIFID_SCHWASSMANN_WACH	1000080	Schwassmann-Wachmann 2
NAIFID_SCHWASSMANN_WACH	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_KR	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_IKEM	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_CROCK	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

continues on next page

Table 2 – continued from previous page

Predefined Macros	NAIF ID	Name
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_BREWINGTO	1000128	Metcalf-Brewington
NAIFID_LEVY	1000129	Levy
NAIFID_SHOEMAKER_LEVY_9	1000130	Shoemaker-Levy 9
NAIFID_HYAKUTAKE	1000131	Hyakutake
NAIFID_HALE_BOPP	1000132	Hale-Bopp
NAIFID_SIDING_SPRING	1003228	Siding Spring

CHAPTER

TEN

FRAME NUMBERS

value	Name
1	ICRF

CHAPTER

ELEVEN

RELEASE NOTES

- **Version 5.0.0**

  Support the Leapseconds Clock Kernel files (spice files .tls).

  Support the Scpacecraft Clock Kernel files (spice files .tsc).

  Support the Instrument Kernels files (spice files .ik).

  Add the set of functions to convert dates (julian days or calendar dates) between different timescales (TCB, TDB, TT, TAI, UTC) : calceph_time... .

  Add the set of functions to write binary SPK and PCK SPICE kernel ephemeris files in TDB or TCB timescales.

  Add the calceph_getfov to retrieve data about the field of view of the instruments.

  Add the documentation of the deprecated functions : Single file access functions.

  Add a new define for asteroids on 8-digit NAIF ID : CALCEPH_ASTEROID_8.

  Fix the compilation with android-ndk if API level >=23.

  Fix the returned value of calceph_gettimescale with SPICE kernel files containing segments of type 102.

  Fix a buffer overflow in the function calceph_getfileversion in the Fortran 2003 interface. Change the array size of the function calceph_getfileversion from the value CALCEPH_MAX_CONSTANTNAME to CALCEPH_MAX_CONSTANTVALUE in the header file and documentation, to conform to the examples and to other interfaces. Only Fortran 2003 are affected by this change : previous header file and documentation contains an invalid declaration of the function.

  Fix a buffer overflow in the function calceph_getconstantss in the Fortran 2003 interface. Change the array size of the last argument of the function calceph_getconstantss from the value CALCEPH_MAX_CONSTANTNAME to CALCEPH_MAX_CONSTANTVALUE in the header file and documentation, to conform to the examples and to other interfaces. Only Fortran 2003 are affected by this change : previous header file and documentation contains an invalid declaration of the function.

  Fix the documentation of the functions f90calceph_getconstantss and f90calceph_getconstantvs in the Fortran 77/90/95 interface. Change the array size of the last argument of the function f90calceph_getconstantss from the value CALCEPH_MAX_CONSTANTNAME to CALCEPH_MAX_CONSTANTVALUE, to conform to the examples and to other interfaces. Only Fortran 77/90/95 are affected by this change : possible buffer overflow occurs with previous documentation.

  Fix the documentation and the behavior with numbers of the function calceph_getconstantvs and calceph_getconstantss.

  Fix the compilation with vcpkg on the operating system arm64/linux.

  Remove the undocumented function calceph_ and calcephinit_.

- **Version 4.0.5**

  Support the segment 19 in the SPICE kernel file.

  Fix the installation with Cython>=3.1.0.

  Fix the compilation of the documentation with Cmake 4.0 or later.

- **Version 4.0.4**

  Fix the calceph_gettimespan function result with non-integer time bounds in SPICE kernel files.

  Support for parallel execution of the python and octave tests.

- **Version 4.0.3**

  Fix a crash of the function calceph_open and calceph_open_array with some invalid SPICE kernel files (files .tpc and .tf) on arm32v6 processors.

  Add the man pages for the tools.

  Replace the calls to sprintf by the calls to the fortified function snprintf.

  Version 4.0.2 skipped due to an archive problem.

- **Version 4.0.1**

  Fix the function calceph_open and calceph_open_array with some SPICE kernel files (files .bsp and .spk) if the segment descriptor does not contain any segments.

- **Version 4.0.0**

  Use cmake to compile the library. Binary ABI may be broken by this major change in the compilation process. The library remains compatible at the source level: the existing application therefore only needs a recompilation.

  Add the function calceph_getidbyname and calceph_getnamebyidss.

  Fix potential crash of fortran-90 interface only, on 32-bit OS, if ephemeris file does not exist.

  Support frame name with any character except tab, space, comma, parenthese, equal, in the frame SPICE kernel file (files .tf).

  Fix system error messages to be thread-safe (remove strerror).

  Fix memory buffer overflow if the number of constants is greater than 400 in the original JPL files.

- **Version 3.5.5**

  Fix memory leaks when files are closed.

  Fix heap buffer overflow (read only operation) with text SPICE kernel files.

  Fix memory crash if the SPICE kernel files contain segments with a size of the directory greater than 100.

- **Version 3.5.4**

  Fix the documentation about the unsupported segment 19.

  Fix the window size if requested time is near a boundary for SPICE kernel files containing segments of type 18, compliant with the SPICE specifications.

  Fix incorrect results for SPICE kernel files containing segments of type 20.

  Support the segment 14 in the SPICE kernel file.

- **Version 3.5.3**

  Fix compilation error with Cython 3.0 or later.

  Fix compilation error with Python 3.10 or later on some linux distributions.

  Fix the tests suite with octave, due to a change of the syntax parsing by octave.

- **Version 3.5.2**

  Fix a possible buffer overflow in the python interface of getfileversion.

  Fix the undefined external symbol rint with visual C compiler 10.0.

  Fix the redefinition of vasprintf with the environnment MinGW.

- **Version 3.5.1**

  Support the incremental assignment in the text SPICE kernel file (files .tpc).

  Support the relative frames in the frame SPICE kernel file (files .tf).

Fix the returned values of calceph_getconstantvs with large SPICE kernels.

Fix random crash or double-free error when meta-kernel SPICE kernel files are closed (files .mk).

Raise an error inside the function calceph_compute if center != 0 for the target (14,15,16,17), to conform to the documentation.

Fix the object name Megaclite (519) from NAIFID_MAGACLITE to NAIFID_MEGACLITE.

- **Version 3.5.0**

  Add the function calceph_getmaxsupportedorder, calceph_getpositionrecordindex2 and calceph_getorientrecordindex2.

  Update config.sub and config.guess to support new processors.

  Fix an incorrect result of calceph_(s)getconstant and calceph_getconstantvd with SPICE text kernel

  if the current locale has a decimal separator different than the decimal point (e.g. comma).

  Fix compilation warnings with gcc 11 about pointers and arrays.

  Fix the installation with python 3.10 or later (PEP 632).

  Fix compilation errors with the android SDK API level < 24.

  Reduce the stack size of the arrays for the evaluation with C99 compilers.

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

  to CALCEPH_MAX_CONSTANTVALUE, to conform to the documentation and examples.

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

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

- **Version 3.4.7**

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

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

- **Version 3.4.6**

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

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

- **Version 3.4.5**

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

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

- **Version 3.4.4**

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

- **Version 3.4.3**

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

  Fix the installation of python package under Anaconda.

- **Version 3.4.2**

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

  Support SPICE kernels larger than 4GBytes.

- **Version 3.4.1**

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

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

- **Version 3.4.0**

  Add the function calceph_isthreadsafe.

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

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

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

  Fix the MinGW Makefiles if the variable MAKE contains spaces.

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

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

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

  Add the utilities calceph_queryposition and calceph_queryorientation.

- **Version 3.3.1**

  Fix the installation with python 3.7.0 or later.

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

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

- **Version 3.3.0**

  Add the functions calceph_getfileversion.

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

  Fix a compiler warning in the file util.c.

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

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

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

- **Version 3.2.0**

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

  Fix the returned value of the functions f90calceph_getconstantvd and f90calceph_getconstantvs.

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

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

  Support the IAU 1980 Nutation Angles of the JPL files.

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

  Add the option installnodoc to the make command.

- **Version 3.1.0**

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

  Add the functions calceph_getconstantsd, calceph_getconstantvd and calceph_getconstantss and calceph_getconstantvs.

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

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

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

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

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

- **Version 3.0.0**

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

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

  Add the function calceph_gettimescale and calceph_gettimespan.

  Add the function calceph_getpositionrecordcount and calceph_getpositionrecordindex.

  Add the function calceph_getorientrecordcount and calceph_getorientrecordindex.

  Add the function calceph_sgettimescale and calceph_sgettimespan.

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

  Use sphinx-doc to produce the documentation.

- **Version 2.3.2**

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

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

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

- **Version 2.3.1**

  Fix the compilation warnings with the Pelles compiler.

  Fix the compilation warnings with the C89 standard.

  Fix the compilation warnings with the GNU C compilers.

  Fix the documentation for the constant CALCEPH_VERSION_STRING.

- **Version 2.3.0**

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

  Add the preprocessor macro CALCEPH_VERSION_STRING.

  Add the function calceph_getversion_str.

  Add the function calceph_compute_order and calceph_orient_order.

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

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

- **Version 2.2.5**

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

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

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

- **Version 2.2.4**

  Update the version number of the dynamic library.

- **Version 2.2.3**

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

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

- **Version 2.2.2**

  Support the compilation in the standard C89.

- **Version 2.2.1**

  Remove debug informations that are printed when errors occur in calceph_?compute_???.
  Support the Portland compilers.
  Fix the info documentation.
  Report an error if no asteroid is available in an ephemeris file with the INPOP file format (instead of a crash).

- **Version 2.2.0**

  Support the new segments 20, 102, 103 and 120 in the SPICE kernel file.
  Support the NAIF identification numbers.
  Add the functions calceph_orient_unit and calceph_prefetch.

- **Version 2.1.0**

  Fix a bug in calceph_getconstant and calceph_sgetconstant with an invalid name
  Remove the null character in the name of the constant returned by the function (f90)calceph_(s)getconstantindex when the Fortran interface is used.

- **Version 2.0.0**

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

- **Version 1.2.0**

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

- **Version 1.1.2**

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

- **Version 1.1.1**

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

- **Version 1.1.0**

  Add the function calceph_seterrorhandler for the custom error handlers.

- **Version 1.0.3**

  Support the JPL ephemeris file DE423.

- **Version 1.0.2**

  Fix memory leaks in the fortran-90 interface.

- **Version 1.0.1**

  Support the large ephemeris files (>2GB) on 32-bit operating systems.

Fix the documentation of the function f90calceph_sopen.

Fix an invalid open mode on Windows operating systems.

Report accurately the I/O errors.

- **Version 1.0.0**

  Initial release.

CHAPTER

TWELVE

REPORTING BUGS

If you think you have found a bug in the CALCEPH Library, first have a look on the CALCEPH Library web page https://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: Mickael.Gastineau@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.

CHAPTER

THIRTEEN

CALCEPH LIBRARY COPYING CONDITIONS

Contributed by

Gastineau M. , Laskar J., Manche H., Astronomie et Systèmes Dynamiques, LTE, Observatoire de Paris, CNRS, PSL Research University, Sorbonne Universite

Fienga A. , Observatoire de la Côte d'Azur