
CALCEPH - Fortran 2003 language

Release 5.0.1

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

Sep 21, 2026

CONTENTS

1	Introduction	3
2	Library interface	5
2.1	A simple example program	5
2.2	Modules and Libraries	5
2.2.1	Compilation on a Unix-like system	6
2.2.2	Compilation on a Windows system	6
2.3	Constants	6
3	Reading/Evaluation functions	9
3.1	Time notes	9
3.2	Thread notes	9
3.3	Usage	9
3.4	Functions	9
3.4.1	calceph_open	9
3.4.2	calceph_open_array	10
3.4.3	calceph_prefetch	11
3.4.4	calceph_isthreadsafe	12
3.4.5	calceph_compute	12
3.4.6	calceph_compute_unit	14
3.4.7	calceph_orient_unit	16
3.4.8	calceph_rotangmom_unit	17
3.4.9	calceph_compute_order	19
3.4.10	calceph_orient_order	21
3.4.11	calceph_rotangmom_order	23
3.4.12	calceph_getconstant	25
3.4.13	calceph_getconstantsd	25
3.4.14	calceph_getconstantvd	26
3.4.15	calceph_getconstantss	27
3.4.16	calceph_getconstantvs	27
3.4.17	calceph_getconstantcount	28
3.4.18	calceph_getconstantindex	29
3.4.19	calceph_getfileversion	30
3.4.20	calceph_getidbyname	30
3.4.21	calceph_getnamebyidss	31
3.4.22	calceph_gettimescale	32
3.4.23	calceph_gettimespan	33
3.4.24	calceph_getpositionrecordcount	34
3.4.25	calceph_getpositionrecordindex	34
3.4.26	calceph_getpositionrecordindex2	35

3.4.27	calceph_getorientrecordcount	36
3.4.28	calceph_getorientrecordindex	37
3.4.29	calceph_getorientrecordindex2	38
3.4.30	calceph_close	39
4	Time conversion functions	41
4.1	Timescale notes	41
4.2	Input string formats	41
4.3	Functions	41
4.3.1	calceph_time_jd_to_cal	41
4.3.2	calceph_time_cal_to_jd	43
4.3.3	calceph_time_str_to_jd	44
4.3.4	calceph_time_set_relationship_tt_tdb	45
4.3.5	calceph_time_jd_tdb_to_jd_tt	46
4.3.6	calceph_time_jd_tt_to_jd_tdb	46
4.3.7	calceph_time_jd_tdb_to_jd_tcb	47
4.3.8	calceph_time_jd_tcb_to_jd_tdb	48
4.3.9	calceph_time_cal_utc_to_jd_tdb	49
4.3.10	calceph_time_jd_tdb_to_cal_utc	50
4.3.11	calceph_time_str_utc_to_jd_tdb	51
4.3.12	calceph_time_str_any_to_jd_tdb	52
4.3.13	calceph_time_str_spacecraft_clock_to_jd_tdb	53
4.3.14	calceph_time_jd_tdb_to_str_spacecraft_clock	54
5	Instrument functions	57
5.1	calceph_getfov	57
6	Error functions	61
6.1	Usage	61
6.2	calceph_seterrorhandler	62
7	Miscellaneous functions	65
7.1	calceph_getmaxsupportedorder	65
7.2	calceph_getversion_str	65
8	Writing functions	67
8.1	Thread notes	68
8.2	Usage	68
8.3	SPK Functions	68
8.3.1	writeph_spk_create	68
8.3.2	writeph_spk_open	69
8.3.3	writeph_close	69
8.3.4	writeph_dump	70
8.3.5	writeph_restore	71
8.3.6	writeph_comment	72
8.3.7	writeph_spk2_seq_write	72
8.3.8	writeph_spk2_par_reserve	74
8.3.9	writeph_spk2_par_write	76
8.3.10	writeph_spk3_seq_write	77
8.3.11	writeph_spk3_par_reserve	79
8.3.12	writeph_spk3_par_write	81
8.3.13	writeph_spk8_seq_write	82
8.3.14	writeph_spk8_par_reserve	84
8.3.15	writeph_spk8_par_write	86
8.3.16	writeph_spk9_seq_write	87

8.3.17	writeph_spk9_par_reserve	88
8.3.18	writeph_spk9_par_write	90
8.3.19	writeph_spk12_seq_write	91
8.3.20	writeph_spk12_par_reserve	93
8.3.21	writeph_spk12_par_write	95
8.3.22	writeph_spk13_seq_write	96
8.3.23	writeph_spk13_par_reserve	98
8.3.24	writeph_spk13_par_write	100
8.3.25	writeph_spk14_begin	101
8.3.26	writeph_spk14_add	102
8.3.27	writeph_spk14_end	103
8.3.28	writeph_spk102_seq_write	103
8.3.29	writeph_spk102_par_reserve	105
8.3.30	writeph_spk102_par_write	107
8.3.31	writeph_spk103_seq_write	108
8.3.32	writeph_spk103_par_reserve	110
8.3.33	writeph_spk103_par_write	112
8.4	PCK Functions	113
8.4.1	writeph_pck_create	113
8.4.2	writeph_pck_open	114
8.4.3	writeph_pck2_seq_write	114
8.4.4	writeph_pck2_par_reserve	116
8.4.5	writeph_pck2_par_write	118
8.4.6	writeph_pck3_seq_write	119
8.4.7	writeph_pck3_par_reserve	121
8.4.8	writeph_pck3_par_write	123
8.4.9	writeph_pck102_seq_write	124
8.4.10	writeph_pck102_par_reserve	126
8.4.11	writeph_pck102_par_write	128
8.4.12	writeph_pck103_seq_write	129
8.4.13	writeph_pck103_par_reserve	131
8.4.14	writeph_pck103_par_write	133
9	Single file access functions	135
9.1	Time notes	135
9.2	Thread notes	136
9.3	Usage	136
9.4	Functions	137
9.4.1	calceph_sopen	137
9.4.2	calceph_scompute	138
9.4.3	calceph_sgetconstant	140
9.4.4	calceph_sgetconstantcount	140
9.4.5	calceph_sgetconstantindex	141
9.4.6	calceph_sgetfileversion	141
9.4.7	calceph_sgettimescale	142
9.4.8	calceph_sgettimespan	142
9.4.9	calceph_sclose	143
10	NAIF identification numbers	145
10.1	Sun and planetary barycenters	145
10.2	Coordinate Time ephemerides	145
10.3	Planet centers and satellites	145
10.4	Comets	149

11	Frame numbers	153
12	Release notes	155
13	Reporting bugs	163
14	CALCEPH Library Copying conditions	165

This manual documents how to install and use the CALCEPH Library using the Fortran 2003 interface.

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

INTRODUCTION

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

- text Planetary Constants Kernel (KPL/PCK) files
- binary PCK (DAF/PCK) files.
- binary SPK (DAF/SPK) files containing segments of type 1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 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 endianness (order in which integers are stored as bytes in computer memory), the library could handle the binary ephemeris files with any endianness. This library automatically swaps the bytes when it performs read operations on the ephemeris file.

The library is able also to write binary SPK and PCK SPICE kernel ephemeris files in the TDB and TCB timescales.

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

LIBRARY INTERFACE

2.1 A simple example program

The following example program shows the typical usage of the Fortran 2003 interface.

Other examples using the Fortran 2003 interface can be found in the directory *examples* of the library sources.

```
program f2003multiple
  USE, INTRINSIC :: ISO_C_BINDING
  use calceph
  integer res
  real(8) AU
  TYPE(C_PTR) :: peph

  peph = calceph_open("example1.dat"//C_NULL_CHAR)
  if (C_ASSOCIATED(peph)) then

    if (calceph_getconstant(peph, "AU"//C_NULL_CHAR, AU).eq.1) then
      write (*,*) "AU=", AU
    endif

    call calceph_close(peph)
  endif
stop
end
```

2.2 Modules and Libraries

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

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

```
use calceph
```

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

2.2.1 Compilation on a Unix-like system

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

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

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

2.2.2 Compilation on a Windows system

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

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

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

2.3 Constants

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

CALCEPH_MAX_CONSTANTNAME [*integer*]

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

CALCEPH_MAX_CONSTANTVALUE [*integer*]

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

CALCEPH_VERSION_MAJOR [*integer*]

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

CALCEPH_VERSION_MINOR [*integer*]

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

CALCEPH_VERSION_PATCH [*integer*]

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

CALCEPH_VERSION_STRING [*character(len=*)*]

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

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

CALCEPH_ASTEROID *[integer]*

This integer defines the offset value for the asteroids, starting at 2E6, that must be used as target or center for the computation functions, such as [calceph_compute\(\)](#).

CALCEPH_ASTEROID_8 *[integer]*

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 [calceph_compute\(\)](#).

The following constants specify in which units are expressed the output of the computation functions, such as [calceph_compute_unit\(\)](#) :

CALCEPH_UNIT_AU *[integer]*

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

CALCEPH_UNIT_KM *[integer]*

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

CALCEPH_UNIT_DAY *[integer]*

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

CALCEPH_UNIT_SEC *[integer]*

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

CALCEPH_UNIT_RAD *[integer]*

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

CALCEPH_OUTPUT_EULERANGLES *[integer]*

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

CALCEPH_OUTPUT_NUTATIONANGLES *[integer]*

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

CALCEPH_USE_NAIFID *[integer]*

This integer defines that the NAIF identification numbers are used as target or center for the computation functions, such as [calceph_compute_unit\(\)](#).

The following constants specify the type of segments for the functions, such as [calceph_getmaxsupportedorder\(\)](#) :

CALCEPH_SEGTYPE_ORIG_0 *[integer]*

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

CALCEPH_SEGTYPE_SPK_1 *[integer]*

CALCEPH_SEGTYPE_SPK_2 *[integer]*

CALCEPH_SEGTYPE_SPK_3 *[integer]*

CALCEPH_SEGTYPE_SPK_5 *[integer]*

CALCEPH_SEGTYPE_SPK_8 *[integer]*

CALCEPH_SEGTYPE_SPK_9 *[integer]*

CALCEPH_SEGTYPE_SPK_12 *[integer]*

CALCEPH_SEGTYPE_SPK_13 *[integer]*

CALCEPH_SEGTYPE_SPK_14 *[integer]*

CALCEPH_SEGTYPE_SPK_17 *[integer]*

CALCEPH_SEGTYPE_SPK_18 *[integer]*

CALCEPH_SEGTYPE_SPK_19 *[integer]*

CALCEPH_SEGTYPE_SPK_20 *[integer]*

CALCEPH_SEGTYPE_SPK_21 *[integer]*

CALCEPH_SEGTYPE_SPK_102 *[integer]*

CALCEPH_SEGTYPE_SPK_103 *[integer]*

CALCEPH_SEGTYPE_SPK_120 *[integer]*

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

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

CALCEPH_TDB *[integer]*

CALCEPH_TCB *[integer]*

CALCEPH_UTC *[integer]*

CALCEPH_TAI *[integer]*

CALCEPH_TT *[integer]*

CALCEPH_TIMESCALE_FROM_STR *[integer]*

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 `calceph_seterrorhandler()`.

3.1 Time notes

The functions `calceph_compute()`, `calceph_compute_unit()`, `calceph_compute_order()`, `calceph_orient_unit()`, ... only accept a date expressed in the same time scale as the ephemeris files, which can be retrieved using the function `calceph_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.

3.2 Thread notes

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

It's safe for two threads to call the functions with the same handle of ephemeris object if and only if the function `calceph_isthreadsafe()` returns a non-zero value. A previous call to the function `calceph_prefetch()` is required for the function `calceph_isthreadsafe()` to return a non-zero value.

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

3.3 Usage

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

The example in Fortran 2003 language is `f2003multiple.f`.

3.4 Functions

3.4.1 calceph_open

function `calceph_open`(*filename*) *BIND(C)*

Parameters

filename [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: pathname of the file.

Return

calceph_open [*TYPE(C_PTR)*] :: ephemeris descriptor. This value is C_NULL_PTR if an error occurs, otherwise non-C_NULL_PTR value.

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

- text Planetary Constants Kernel (KPL/PCK) files
- binary PCK (DAF/PCK) files.
- binary SPK (DAF/SPK) files containing segments of type 1, 2, 3, 5, 8, 9, 12, 13, 14, 17, 18, 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 `calceph_open()`, the function `calceph_prefetch()` should be called to accelerate future computations.

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

The following example opens the ephemeris file example1.dat

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    ! ... computation ...

endif
call calceph_close(peph)
```

3.4.2 calceph_open_array

function `calceph_open_array`(*n, array_filename, len_filename*) *BIND(C)*

Parameters

- **n** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of files.
- **array_filename** [*CHARACTER(len=1,kind=C_CHAR), dimension(*), intent(in)*] :: array of pathname of the files.
- **len_filename** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of characters of each file's name.

Return

calceph_open_array [*TYPE(C_PTR)*] :: ephemeris descriptor. This value is C_NULL_PTR if an error occurs, otherwise non-C_NULL_PTR value.

This function opens *n* files whose pathnames are the string pointed to by `array_filename`, reads the header blocks of these files and returns 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 `calceph_open_array()`, the function `calceph_prefetch()` should be called to accelerate future computations.

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

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

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

3.4.3 calceph_prefetch

function `calceph_prefetch`(*eph*) *BIND(C)*

Parameters

eph [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor.

Return

calceph_prefetch [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function prefetches to the main memory all files associated to the ephemeris descriptor *eph*. This prefetching operation will accelerate the further computations performed with `calceph_compute()`, `calceph_compute_unit()`, `calceph_compute_order()`, `calceph_orient_unit()`,

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

3.4.4 calceph_isthreadsafe

function `calceph_isthreadsafe`(*eph*) *BIND(C)*

Parameters

eph [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor.

Return

calceph_isthreadsafe [*INTEGER(C_INT)*] :: returns 1 if multiple threads can access the same ephemeris ephemeris descriptor, otherwise 0.

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

A previous call to the function `calceph_prefetch()` is required, and the library should be compiled with **--enable-thread=yes** on Unix-like operating system, for the function `calceph_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 `calceph_compute()`, It allows to use the same object for the parallel loops.

3.4.5 calceph_compute

function `calceph_compute`(*eph*, *JD0*, *time*, *target*, *center*, *PV*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **JD0** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: Fraction part of the Julian date (TDB or TCB)
- **target** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The body or reference point whose coordinates are required (see the list, below).
- **center** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The origin of the coordinate system (see the list, below). If *target* is 14, 15, 16 or 17 (nutaton, libration, TT-TDB or TCG-TCB), *center* must be 0.
- **PV** [*REAL(C_DOUBLE)*, *dimension(1:6)*, *intent(out)*] :: Depending on the target value, an array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot), or a time scale transformation value, or the angles of the librations of the Moon and their derivatives, or the nutaton angles and their derivatives.

Return

calceph_compute [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

The returned array *PV* has the following properties

- If the target is *TT-TDB*, only the first element of this array will get the result. The time scale transformation TT-TDB is expressed in seconds.
- If the target is *TCG-TCB*, only the first element of this array will get the result. The time scale transformation TCG-TCB is expressed in seconds.

- If the target is *Librations*, the array contains the angles of the librations of the Moon and their derivatives. The angles of the librations of the Moon are expressed in radians and their derivatives are expressed in radians per day.
- If the target is *Nutations*, the array contains the nutation angles and their derivatives. The nutation angles are expressed in radians and their derivatives are expressed in radians per day.
- Otherwise the returned values is the cartesian position (x,y,z), expressed in Astronomical Unit (au), and the velocity (xdot, ydot, zdot), expressed in Astronomical Unit per day (au/day).

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

```
integer*8 peph
integer res
```

(continues on next page)

(continued from previous page)

```

real(8) jd0
real(8) dt1, dt2
real(8) PV(6)
TYPE(C_PTR) :: peph

jd0 = 2442457
dt1 = 0.5D0
dt2 = 0.9D0
peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

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

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

    call calceph_close(peph)
endif

```

3.4.6 calceph_compute_unit

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

Parameters

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

The units of PV.

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

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

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

- **PV** [*REAL(C_DOUBLE), dimension(1:6), intent(out)*] :: Depending on the target value, an array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot), or a time scale transformation value, or the angles of the librations of the Moon and their derivatives, or the nutation angles and their derivatives.

Return

calceph_compute_unit [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function is similar to the function [calceph_compute\(\)](#), except that the units of the output are specified.

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

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

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

For example, to get the position and velocities expressed in kilometers and kilometers/seconds, the unit must be set to [CALCEPH_UNIT_KM](#) + [CALCEPH_UNIT_SEC](#).

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

```
integer res
real(8) jd0
real(8) dt1
real(8) PV(6)
TYPE(C_PTR) :: peph

jd0 = 2442457
dt1 = 0.5D0
```

(continues on next page)

(continued from previous page)

```

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    ! the heliocentric coordinates of Mars in km and km/s
    res = calceph_compute_unit(peph,jd0, dt1, 4, 11,
&                                CALCEPH_UNIT_KM+CALCEPH_UNIT_SEC,
&                                PV)
    write(*,*) PV

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

    call calceph_close(peph)
endif

```

3.4.7 calceph_orient_unit

function `calceph_orient_unit`(*eph*, *JD0*, *time*, *target*, *unit*, *PV*) *BIND(C)*

Parameters

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

The units of PV.

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

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

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

- **PV** [*REAL(C_DOUBLE)*, *dimension(1:6)*, *intent(out)*] :: An array to receive the euler angles, or nutation angles, and their derivatives for the orientation of the body.

Return

`calceph_orient_unit` [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

The returned array *PV* has the following properties

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

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

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

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

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

```
integer res
real(8) jd0
real(8) dt1
real(8) PV(6)
TYPE(C_PTR) :: peph

jd0 = 2442457
dt1 = 0.5D0
peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

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

    call calceph_close(peph)
endif
```

3.4.8 calceph_rotangmom_unit

function `calceph_rotangmom_unit`(*eph*, *JD0*, *time*, *target*, *unit*, *PV*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **JD0** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: Fraction part of the Julian date (TDB or TCB)

- **target** [*INTEGER(C_INT), VALUE, intent(in)*] :: The body whose orientations are requested. The numbering system depends on the parameter unit.
- **unit** [*INTEGER(C_INT), VALUE, intent(in)*] ::
The units of PV.
This integer is a sum of some unit constants (CALCEPH_UNIT_???) and/or the constant *CALCEPH_USE_NAIFID*.
If the unit contains *CALCEPH_USE_NAIFID*, the NAIF identification numbering system is used for the target (*NAIF identification numbers* for the list).
If the unit does not contain *CALCEPH_USE_NAIFID*, the old number system is used for the target (see the list in the function *calceph_compute()*).
- **PV** [*REAL(C_DOUBLE), dimension(1:6), intent(out)*] :: An array to receive the angular momentum due to its rotation, divided by the product of the mass and of the square of the radius, and the derivatives, of the body.

Return

calceph_rotangmom_unit [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function reads, if needed, in the ephemeris file associated to *eph* and interpolates the angular momentum vector due to the rotation of the body, divided by the product of the mass m and of the square of the radius R , of a single body (*target*) for the time $JD0+time$ and stores the results to *PV*. The ephemeris file associated to *eph* must have been previously opened with the function *calceph_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 *calceph_gettimescale()*.

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

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

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

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

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

```
integer res
real(8) jd0
real(8) dt1
real(8) G(6)
TYPE(C_PTR) :: peph

jd0 = 2451419
dt1 = 0.5D0
peph = calceph_open("example2_rotangmom.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

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

(continues on next page)

(continued from previous page)

```

call calceph_close(peph)
endif

```

3.4.9 calceph_compute_order

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

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **JD0** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: Fraction part of the Julian date (TDB or TCB)
- **target** [*INTEGER(C_INT), VALUE, intent(in)*] :: The body or reference point whose coordinates are required (see the list, below).
- **center** [*INTEGER(C_INT), VALUE, intent(in)*] :: The origin of the coordinate system (see the list, below). If *target* is 14, 15, 16 or 17 (nutaton, libration, TT-TDB or TCG-TCB), *center* must be 0.
- **unit** [*INTEGER(C_INT), VALUE, intent(in)*] ::

The units of PVAJ.

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

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

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

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

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

- **PVAJ** [*REAL(C_DOUBLE), dimension(1:12), intent(out)*] :: Depending on the target value, an array to receive the cartesian position (x,y,z), the velocity (xdot, ydot, zdot), the acceleration and the jerk, or a time scale transformation value, or the angles of the librations of the Moon and their successive derivatives, or the nutation angles and their successive derivatives.

Return

`calceph_compute_order` [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

The returned array *PVAJ* has the following properties

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

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

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

The date (JD0, time) should be expressed in the same time scale as the ephemeris files, which can be retrieved using the function `calceph_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 `calceph_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 `CALCEPH_UNIT_AU`.
- The position, velocity, acceleration and jerk are expressed in kilometers if unit contains `CALCEPH_UNIT_KM`.
- The velocity, acceleration, jerk, TT-TDB, TCG-TCB or the derivatives of the angles of the librations of the Moon are expressed in days if unit contains `CALCEPH_UNIT_DAY`.
- The velocity, acceleration, jerk, TT-TDB, TCG-TCB or the derivatives of the angles of the librations of the Moon are expressed in seconds if unit contains `CALCEPH_UNIT_SEC`.
- The angles of the librations of the Moon are expressed in radians if unit contains `CALCEPH_UNIT_RAD`.

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

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

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

```
integer res
real(8) jd0
real(8) dt1
real(8) P(3)
real(8) PVAJ(12)
TYPE(C_PTR) :: peph

jd0 = 2442457
dt1 = 0.5D0
peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

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

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

    call calceph_close(peph)
endif
```

3.4.10 calceph_orient_order

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

Parameters

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

The units of PVAJ.

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

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

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

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

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

- **PVAJ** [`REAL(C_DOUBLE)`, `dimension(1:12)`, `intent(out)`] :: An array to receive the euler angles, or nutation angles, and their derivatives for the orientation of the body.

Return

calceph_compute_order [`INTEGER(C_INT)`] :: 0 if an error occurs, otherwise non-zero value.

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

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

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

The returned array *PVAJ* has the following properties

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

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

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

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

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

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

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

```

integer res
real(8) jd0
real(8) dt1
real(8) P(3)
TYPE(C_PTR) :: peph

jd0 = 2442457
dt1 = 0.5D0
peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    res = calceph_orient_order(peph,jd0, dt1, NAIFID_MOON,
&                                CALCEPH_USE_NAIFID+CALCEPH_UNIT_RAD+CALCEPH_UNIT_SEC,
&                                0, P)
    write(*,*) P

    call calceph_close(peph)
endif

```

3.4.11 calceph_rotangmom_order

function `calceph_rotangmom_order`(*eph*, *JD0*, *time*, *target*, *unit*, *order*, *PVAJ*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **JD0** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: Fraction part of the Julian date (TDB or TCB)
- **target** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The body whose orientations are requested. The numbering system depends on the parameter unit.
- **unit** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] ::
 The units of PVAJ.
 This integer is a sum of some unit constants (`CALCEPH_UNIT_???`) and/or the constant `CALCEPH_USE_NAIFID`.
 If the unit contains `CALCEPH_USE_NAIFID`, the NAIF identification numbering system is used for the target and the center (*NAIF identification numbers* for the list).
 If the unit doesnot contain `CALCEPH_USE_NAIFID`, the old number system is used for the target and the center (see the list in the function `calceph_compute()`).
- **order** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The order of derivatives.
 - = 0 , only the angular momentum is computed. The first three numbers of PVAJ are valid for the results.
 - = 1 , only the angular momentum and the first derivative are computed. The first six numbers of PVAJ are valid for the results.
 - = 2 , only the angular momentum and the first and second derivatives are computed. The first nine numbers of PVAJ are valid for the results.

- = 3 , the angular momentum and the first, second and third derivatives are computed. The first twelve numbers of PVAJ are valid for the results.

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

- **PVAJ** [*REAL(C_DOUBLE), dimension(1:12), intent(out)*] :: An array to receive the angular momentum due to its rotation, divided by the product of the mass and of the square of the radius, and their different order of the derivatives, of the body.

Return

calceph_rotangmom_order [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

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

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

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

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

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

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

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

```
integer res
real(8) jd0
real(8) dt1
real(8) G(3)
TYPE(C_PTR) :: peph

jd0 = 2451419
dt1 = 0.5D0
peph = calceph_open("example2_rotangmom.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    res = calceph_rotangmom_order(peph,jd0, dt1, NAIFID_EARTH,
&                                CALCEPH_USE_NAIFID+CALCEPH_UNIT_SEC,
```

(continues on next page)

(continued from previous page)

```

&                                G)
  write(*,*) G

  call calceph_close(peph)
endif

```

3.4.12 calceph_getconstant

function `calceph_getconstant`(*eph*, *name*, *value*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor.
- **name** [*CHARACTER(len=1,kind=C_CHAR)*, *intent(in)*] :: name of the constant.
- **value** [*REAL(C_DOUBLE)*, *intent(out)*] :: first value of the constant.

Return

calceph_getconstant [*INTEGER(C_INT)*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

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

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

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

```

integer res
real(8) AU
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
  ! print the value of AU
  if (calceph_getconstant(peph, "AU"//C_NULL_CHAR, AU).eq.1) then
    write (*,*) "AU=", AU
  endif
endif

call calceph_close(peph)
endif

```

3.4.13 calceph_getconstantsd

function `calceph_getconstantsd`(*eph*, *name*, *value*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor.
- **name** [*CHARACTER(len=1,kind=C_CHAR)*, *intent(in)*] :: name of the constant.
- **value** [*REAL(C_DOUBLE)*, *intent(out)*] :: first value of the constant.

Return

calceph_getconstantsd [*INTEGER(C_INT)*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

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

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

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

```
integer res
real(8) AU
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
    ! print the value of AU
    if (calceph_getconstantsd(peph, "AU"//C_NULL_CHAR, AU).eq.1) then
        write (*,*) "AU=", AU
    endif
endif

call calceph_close(peph)
endif
```

3.4.14 calceph_getconstantvd

function `calceph_getconstantvd(eph, name, arrayvalue, nvalue) BIND(C)`

Parameters

- **eph** [`TYPE(C_PTR)`, *VALUE*, *intent(in)*] :: ephemeris descriptor.
- **name** [`CHARACTER(len=1,kind=C_CHAR)`, *intent(in)*] :: name of the constant.
- **value** [`REAL(C_DOUBLE)`, *dimension(1:nvalue)*, *intent(out)*] :: array of values for the constant.
- **nvalue** [`INTEGER(C_INT)`, *VALUE*, *intent(in)*] :: number of elements of the array

Return

`calceph_getconstantvd` [`INTEGER(C_INT)`] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

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

The required value *nvalue* to store all values can be determinated with the previous call to `calceph_getconstantsd`.

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

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

```
integer res, nvalue
real(8) svalue
real(8), allocatable :: radii
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
    ! get the number of values
```

(continues on next page)

(continued from previous page)

```

nvalue = calceph_getconstantsd(peph, "BODY399_RADII"//C_NULL_CHAR, svalue)
! fill the array
allocate(radii(1:nvalue))
res = calceph_getconstantvd(peph, "BODY399_RADII"//C_NULL_CHAR, radii, nvalue)
write(*,*) radii

call calceph_close(peph)
endif

```

3.4.15 calceph_getconstantss

function calceph_getconstantss(*eph, name, value*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **name** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: name of the constant.
- **value** [*CHARACTER(len=1,kind=C_CHAR), dimension(CALCEPH_MAX_CONSTANTVALUE), intent(out)*] :: first value of the constant.

Return

calceph_getconstantss [*INTEGER(C_INT)*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

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

Trailing blanks are added to each value.

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

```

character(len=CALCEPH_MAX_CONSTANTVALUE, kind=C_CHAR) UNIT
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
! print the value of UNIT
if (calceph_getconstantss(peph, "UNIT"//C_NULL_CHAR, UNIT).eq.1) then
write (*,*) "UNIT=", trim(UNIT)
endif

call calceph_close(peph)
endif

```

3.4.16 calceph_getconstantvts

function calceph_getconstantvts(*eph, name, arrayvalue, nvalue*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **name** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: name of the constant.

- **value** [*CHARACTER(len=1,kind=C_CHAR), dimension(1:nvalue), intent(out)*] :: array of values for the constant.
- **nvalue** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of elements of the array

Return

calceph_getconstantvs [*INTEGER(C_INT)*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

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

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

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.

Trailing blanks are added to each value.

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

```
integer res, nvalue
character(len=CALCEPH_MAX_CONSTANTVALUE, kind=C_CHAR) svalue
character(len=CALCEPH_MAX_CONSTANTVALUE, kind=C_CHAR), allocatable :: mission_units
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
    ! get the number of values
    nvalue = calceph_getconstantss(peph, "MISSION_UNITS"//C_NULL_CHAR, svalue)
    ! fill the array
    allocate(mission_units(1:nvalue))
    res = calceph_getconstantvs(peph, "MISSION_UNITS"//C_NULL_CHAR, mission_units,
    ↪nvalue)
    write(*,*) mission_units
    deallocate(mission_units)

    call calceph_close(peph)
endif
```

3.4.17 calceph_getconstantcount

function **calceph_getconstantcount** (*eph*) *BIND(C)*

Parameters

eph [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor

Return

calceph_getconstantcount [*INTEGER(C_INT)*] :: number of constants. 0 if an error occurs, otherwise non-zero value.

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

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

```
integer n
TYPE(C_PTR) :: peph
```

(continues on next page)

(continued from previous page)

```

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
  n = calceph_getconstantcount(peph)
  write (*,*) "number of constants", n
  call calceph_close(peph)
endif

```

3.4.18 calceph_getconstantindex

function `calceph_getconstantindex`(*eph*, *index*, *name*, *value*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the constant, between 1 and `calceph_getconstantcount()`
- **name** [*CHARACTER(len=1,kind=C_CHAR)*, *dimension(CALCEPH_MAX_CONSTANTNAME)*, *intent(out)*] :: name of the constant.
- **value** [*REAL(C_DOUBLE)*, *intent(out)*] :: first value of the constant

Return

`calceph_getconstantindex` [*INTEGER(C_INT)*] :: returns 0 if an error occurs, otherwise the number of values associated to the constant.

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

```

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

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

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

```

(continues on next page)

(continued from previous page)

```

    call calceph_close(peph)
endif

```

3.4.19 calceph_getfileversion

function calceph_getfileversion(*eph, version*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **version** [*CHARACTER(len=1,kind=C_CHAR), dimension(CALCEPH_MAX_CONSTANTVALUE), intent(out)*] :: version of the file

Return

calceph_getfileversion [*INTEGER(C_INT)*] :: returns 0 if the file version was not found, otherwise non-zero value.

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

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

The following example prints the version of the ephemeris file.

```

integer res
character(len=CALCEPH_MAX_CONSTANTVALUE) version
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
    res = calceph_getfileversion(peph, version)
    write (*,*) "The version of the file is ", version

    call calceph_close(peph)
endif

```

3.4.20 calceph_getidbyname

function calceph_getidbyname(*eph, name, unit, id*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **name** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: name of the body.
- **unit** [*INTEGER(C_INT), VALUE, intent(in)*] ::

The unit of id.

This integer is 0 or the constant *CALCEPH_USE_NAIFID*.

If the unit equals *CALCEPH_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 *calceph_compute()*).

- **id** [*INTEGER(C_INT), intent(out)*] :: id of the body. The numbering system depends on the parameter unit.

Return

calceph_getidbyname [*INTEGER(C_INT)*] :: returns 0 if the name of the body was not found, otherwise non-zero value.

This function returns, in the parameter *id*, the identification number of the body associated to the given name.

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 overridden 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.

```
integer res
integer moon
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
  ! print the id of the Moon using the old numbering system
  if (calceph_getidbyname(peph, "Moon"//C_NULL_CHAR, 0, moon).eq.1) then
    write (*,*) "Moon=", moon
  endif
  ! print the id of the Moon using the NAIF identification numbering system
  if (calceph_getidbyname(peph, "Moon"//C_NULL_CHAR, CALCEPH_USE_NAIFID, moon).eq.1)
    then
    write (*,*) "Moon=", moon
  endif
  call calceph_close(peph)
endif
```

3.4.21 calceph_getnamebyidss

function **calceph_getnamebyidss**(*eph, id, unit, name*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **id** [*INTEGER(C_INT), intent(out)*] :: id of the body. The numbering system depends on the parameter unit.
- **unit** [*INTEGER(C_INT), VALUE, intent(in)*] ::
The unit of id.
This integer is 0 or the constant *CALCEPH_USE_NAIFID*.
If the unit equals *CALCEPH_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 *calceph_compute()*).
- **name** [*CHARACTER(len=1,kind=C_CHAR), dimension(CALCEPH_MAX_CONSTANTVALUE), intent(out)*] :: name of the body

Return

calceph_getnamebyidss [*INTEGER(C_INT)*] :: returns 0 if the name of the body was not found, otherwise non-zero value.

This function returns, in the parameter *name*, the first given name of the body associated to the identification number *id*.

To remove the C null character at the end of the name, you could use the following statement : `name = name(1:index(name, C_NULL_CHAR)-1)`

The library has a default mapping listed in the list [NAIF identification numbers](#). The mapping name/id may be overridden 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.

```
integer res
character(len=CALEPH_MAX_CONSTANTVALUE, kind=C_CHAR) emb
character(len=CALEPH_MAX_CONSTANTVALUE, kind=C_CHAR) name_c
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
    ! print the name of the Earth-Moon barycenter using the old numbering system
    ! remove the c null character
    if (calceph_getidbyname(peph, 13, 0, emb).eq.1) then
        emb = name_c(1:index(fname, C_NULL_CHAR)-1)
        write (*,*) "EMB=", emb
    endif
    ! print the name of the Earth-Moon barycenter using the NAIF identification_
    ↪ numbering system
    ! remove the c null character
    if (calceph_getidbyname(peph, 3, CALEPH_USE_NAIFID, emb).eq.1) then
        emb = name_c(1:index(fname, C_NULL_CHAR)-1)
        write (*,*) "EMB=", emb
    endif
    call calceph_close(peph)
endif
```

3.4.22 calceph_gettimescale

function **calceph_gettimescale**(*eph*) *BIND(C)*

Parameters

eph [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor

Return

calceph_gettimescale [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

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

```
integer res
integer timescale
```

(continues on next page)

(continued from previous page)

```

TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
  ! print the time scale
  timescale = calceph_gettimescale(peph)
  write (*,*) "timescale=", timescale

  call calceph_close(peph)
endif

```

3.4.23 calceph_gettimespan

function `calceph_gettimespan`(*eph*, *firsttime*, *lasttime*, *continuous*) *BIND(C)*

Parameters

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

Return

calceph_gettimespan [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

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

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

```

integer res
integer :: continuous
real(8) :: firsttime, lasttime
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
  if (calceph_gettimespan(peph, firsttime, lasttime, continuous).eq.1) then
    write (*,*) firsttime, lasttime, continuous
  endif

  call calceph_close(peph)
endif

```

3.4.24 calceph_getpositionrecordcount

function calceph_getpositionrecordcount(*eph*) BIND(C)

Parameters

eph [TYPE(C_PTR), VALUE, intent(in)] :: ephemeris descriptor

Return

calceph_getpositionrecordcount [INTEGER(C_INT)] :: number of position's records. 0 if an error occurs, otherwise non-zero value.

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

```
integer res
integer n
TYPE(C_PTR) :: peph

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
  n = calceph_getpositionrecordcount(peph)
  write (*,*) "number of position's record", n
  call calceph_close(peph)
endif
```

3.4.25 calceph_getpositionrecordindex

function calceph_getpositionrecordindex(*eph*, *index*, *target*, *center*, *firsttime*, *lasttime*, *frame*) BIND(C)

Parameters

- **eph** [TYPE(C_PTR), VALUE, intent(in)] :: ephemeris descriptor
- **index** [INTEGER(C_INT), VALUE, intent(int)] :: index of the position's record, between 1 and [calceph_getpositionrecordcount\(\)](#)
- **target** [INTEGER(C_INT), intent(out)] :: The target body
- **center** [INTEGER(C_INT), intent(out)] :: The origin body
- **firsttime** [REAL(C_DOUBLE), intent(out)] :: Julian date of the first time
- **lasttime** [REAL(C_DOUBLE), intent(out)] :: Julian date of the last time
- **frame** [INTEGER(C_INT), intent(out)] :: reference frame (see the list, below)

Return

calceph_getpositionrecordindex [INTEGER(C_INT)] :: 0 if an error occurs, otherwise non-zero value.

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

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
implicit none
integer res
integer j, itarget, icenter, iframe
real(C_DOUBLE) firsttime, lasttime
TYPE(C_PTR) :: peph

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

! print the list of positionrecords
do j=1, calceph_getpositionrecordcount(peph)
  res = calceph_getpositionrecordindex(peph,j,itarget, icenter, firsttime, lasttime,
  → iframe)
  write (*,*) itarget, icenter, firsttime, lasttime, iframe
enddo

  call calceph_close(peph)
endif

```

3.4.26 calceph_getpositionrecordindex2

function `calceph_getpositionrecordindex2`(*eph, index, target, center, firsttime, lasttime, frame, segid*)
BIND(C)

Parameters

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

Return

`calceph_getpositionrecordindex2` [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function returns the target and origin bodies, the first and last time, the reference frame, and the segment type available at the specified index for the position's records of the ephemeris file associated to *eph*. The NAIF identification

numbering system is used for the target and center integers (*NAIF identification numbers* for the list). The Julian date for the first and last time are expressed in the time scale returned by `calceph_gettimescale()`.

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

value	Name
1	ICRF

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

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

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
implicit none
integer res
integer j, itarget, icenter, iframe, iseg
real(C_DOUBLE) firsttime, lasttime
TYPE(C_PTR) :: peph

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

! print the list of positionrecords
do j=1, calceph_getpositionrecordcount(peph)
  res = calceph_getpositionrecordindex2(peph,j,itarget, icenter, firsttime,
↪lasttime, iframe, iseg)
  write (*,*) itarget, icenter, firsttime, lasttime, iframe, iseg
enddo

call calceph_close(peph)
endif

```

3.4.27 calceph_getorientrecordcount

function `calceph_getorientrecordcount(eph) BIND(C)`

Parameters

eph [`TYPE(C_PTR)`, `VALUE`, `intent(in)`] :: ephemeris descriptor

Return

calceph_getorientrecordcount [`INTEGER(C_INT)`] :: number of orientation's records. 0 if an error occurs, otherwise non-zero value.

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

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

```

integer res
integer n
TYPE(C_PTR) :: peph

```

(continues on next page)

(continued from previous page)

```

peph = calceph_open("example1.dat"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
  n = calceph_getorientrecordcount(peph)
  write (*,*) "number of orientation's record", n
  call calceph_close(peph)
endif

```

3.4.28 calceph_getorientrecordindex

function calceph_getorientrecordindex(*eph*, *index*, *target*, *firsttime*, *lasttime*, *frame*) *BIND(C)*

Parameters

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

Return

calceph_getorientrecordindex [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
implicit none
integer res
integer j, itarget, iframe
real(C_DOUBLE) firsttime, lasttime
TYPE(C_PTR) :: peph

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

```

(continues on next page)

(continued from previous page)

```

! print the list of orientation records
do j=1, calceph_getorientrecordcount(peph)
  res = calceph_getorientrecordindex(peph,j,itarget, firsttime, lasttime, iframe)
  write (*,*) itarget, firsttime, lasttime, iframe
enddo

call calceph_close(peph)
endif

```

3.4.29 calceph_getorientrecordindex2

function `calceph_getorientrecordindex2`(*eph, index, target, firsttime, lasttime, frame, segid*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **index** [*INTEGER(C_INT), VALUE, intent(int)*] :: index of the orientation's record, between 1 and `calceph_getorientrecordcount()`
- **target** [*INTEGER(C_INT), intent(out)*] :: The target body
- **firsttime** [*REAL(C_DOUBLE), intent(out)*] :: Julian date of the first time
- **lasttime** [*REAL(C_DOUBLE), intent(out)*] :: Julian date of the last time
- **frame** [*INTEGER(C_INT), intent(out)*] :: reference frame (see the list, below)
- **segid** [*INTEGER(C_INT), intent(out)*] :: type of the segment.

Return

`calceph_getorientrecordindex2` [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

value	Name
1	ICRF

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

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

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
implicit none
integer res
integer j, itarget, iframe, iseg
real(C_DOUBLE) firsttime, lasttime
TYPE(C_PTR) :: peph

```

(continues on next page)

(continued from previous page)

```

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

! print the list of orientation records
do j=1, calceph_getorientrecordcount(peph)
  res = calceph_getorientrecordindex2(peph,j,itarget, firsttime, lasttime, iframe,
↪iseg)
  write (*,*) itarget, firsttime, lasttime, iframe, iseg
enddo

call calceph_close(peph)
endif

```

3.4.30 calceph_close

subroutine `calceph_close`(*eph*) *BIND(C)*

Parameters

eph [*TYPE*(*C_PTR*), *VALUE*, *intent(in)*] :: ephemeris descriptor

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

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 `calceph_seterrorhandler()`.

4.1 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 `calceph_time_set_relationship_tt_tdb()` function.

4.2 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)"
```

4.3 Functions

4.3.1 `calceph_time_jd_to_cal`

function calceph_time_jd_to_cal(*eph, timescale, jd0, jdfrac, yy, month, day, hh, min, sec*) BIND(C)

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **timescale** [*INTEGER(C_INT), VALUE, intent(in)*] :: timescale of the input date and result (see *Timescale constant* for the available timescales)
- **jd0** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: Integer part of Julian Date
- **jdfrac** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: Fractional part of Julian Date
- **yy** [*INTEGER(C_INT), intent(out)*] :: Year
- **month** [*INTEGER(C_INT), intent(out)*] :: Month [1-12]
- **day** [*INTEGER(C_INT), intent(out)*] :: Day [1-31]
- **hh** [*INTEGER(C_INT), intent(out)*] :: Hour [0-23]
- **min** [*INTEGER(C_INT), intent(out)*] :: Minute [0-59]
- **sec** [*REAL(C_DOUBLE), intent(out)*] :: Seconds [0-60]

Return

calceph_time_jd_to_cal [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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:

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

If the timescale is **CALCEPH.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 UTC:

```
integer res, yy, mo, dd, hh,mi
real(8) ss
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    res = calceph_time_set_relationship_tt_tdb(peph, 1)
    ! convert the julian day 2457754.499994212962963 UTC to the calendar date UTC
    res = calceph_time_jd_to_cal(peph, 2460679.0, 0.45902180578559637, yy, mo, dd, hh,mi,
    ↪ ss)
    write(*,*) yy, mo, dd, hh,mi, ss ! print 2016 12 31 23 59 60.49999421295723 (leap_
    ↪ second)

    call calceph_close(peph)
endif
```

4.3.2 calceph_time_cal_to_jd

function `calceph_time_cal_to_jd`(*eph, timescale, yy, month, day, hh, min, sec, jd0, jdfrac*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **timescale** [*INTEGER(C_INT), VALUE, intent(in)*] :: timescale of the input date and result (see *Timescale constant* for the available timescales)
- **yy** [*INTEGER(C_INT), VALUE, intent(in)*] :: Year
- **month** [*INTEGER(C_INT), VALUE, intent(in)*] :: Month [1-12]
- **day** [*INTEGER(C_INT), VALUE, intent(in)*] :: Day [1-31]
- **hh** [*INTEGER(C_INT), VALUE, intent(in)*] :: Hour [0-23]
- **min** [*INTEGER(C_INT), VALUE, intent(in)*] :: Minute [0-59]
- **sec** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: Seconds [0.0-60.0]
- **jd0** [*REAL(C_DOUBLE), intent(out)*] :: Integer part of Julian Date
- **jdfrac** [*REAL(C_DOUBLE), intent(out)*] :: Fractional part of Julian Date

Return

`calceph_time_cal_to_jd` [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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:

- **CALCEPH_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., **CALCEPH_TT**, **CALCEPH_TDB**): The time is treated as continuous (without leap seconds). The minutes are assumed to always contain 60 seconds.

If `timescale` is **CALCEPH_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:

```
integer res
real(8) jd0, jdfrac
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    ! convert 2025-01-03T22:59:50.300 UTC to the julian day UTC
    res = calceph_time_cal_to_jd(peph, CALCEPH_UTC, 2025, 1, 3, 22, 59, 50.3, jd0,
```

(continues on next page)

(continued from previous page)

```

→ jdfraction
  write(*,*) jd0, jdfraction ! print 2460679.0 0.45822106481481484

  call calceph_close(peph)
endif

```

4.3.3 calceph_time_str_to_jd

function `calceph_time_str_to_jd`(*eph, timescale, const char *str, jd0, jdfraction*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **timescale** [*INTEGER(C_INT), VALUE, intent(in)*] :: timescale of the input date and result (see *Timescale constant* for the available timescales)
- **str** [*CHARACTER(len=*, intent(in))*] :: Input time string to be parsed
- **jd0** [*REAL(C_DOUBLE), intent(out)*] :: Integer part of Julian Date
- **jdfraction** [*REAL(C_DOUBLE), intent(out)*] :: Fractional part of Julian Date

Return

calceph_time_str_to_jd [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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., *CALCEPH_UTC*, *CALCEPH_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.
- *CALCEPH_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 associated to *eph* must contain leap second data.

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

```

integer res
real(8) jd0, jdfraction
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

  res = calceph_time_set_relationship_tt_tdb(peph, 1)

```

(continues on next page)

(continued from previous page)

```

! convert the calendar date TAI to the julian day TAI
res = calceph_time_str_to_jd(peph, CALCEPH_TAI, "2025-01-03T22:59:50.300"//C_NULL_
↪CHAR, jd0, jdfrac)
write(*,*) jd0, jdfrac ! print 2460679.0 0.45822106481481484

call calceph_close(peph)
endif

```

4.3.4 calceph_time_set_relationship_tt_tdb

function `calceph_time_set_relationship_tt_tdb`(*eph, model*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **model** [*INTEGER(C_INT), VALUE, intent(in)*] ::
Relationship model
0: use of `calceph_compute_unit()`
1: use the model based on TLS file

Return

calceph_time_set_relationship_tt_tdb [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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 `calceph_time_jd_tdb_to_jd_tt()` and `calceph_time_jd_tt_to_jd_tdb()`.

The supported values for the model argument are:

- 0: The conversion uses `calceph_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:

```

integer res
real(8) jd0, jdfrac
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    res = calceph_time_set_relationship_tt_tdb(peph, 1)
    res = calceph_time_jd_tt_to_jd_tdb(peph, 2460679.0, 0.4590218056322423, jd0, jdfrac)

    call calceph_close(peph)
endif

```

4.3.5 calceph_time_jd_tdb_to_jd_tt

```
function calceph_time_jd_tdb_to_jd_tt(eph, jd0_tdb, jdfrac_tdb, jd0_tt, jdfrac_tt) BIND(C)
```

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **jd0_tdb** [*REAL(C_DOUBLE), intent(in)*] :: Integer part of Julian Date (TDB)
- **jdfrac_tdb** [*REAL(C_DOUBLE), intent(in)*] :: Fractional part of Julian Date (TDB)
- **jd0_tt** [*REAL(C_DOUBLE), intent(out)*] :: Integer part of Julian Date (TT)
- **jdfrac_tt** [*REAL(C_DOUBLE), intent(out)*] :: Fractional part of Julian Date (TT)

Return

calceph_time_jd_tdb_to_jd_tt [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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 `calceph_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 `calceph_compute_unit()`.
- Model 1: Uses the internal default model from the TLS file.

The following example converts a TDB date to TT:

```
integer res
real(8) jd0, jdfrac
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    res = calceph_time_set_relationship_tt_tdb(peph, 1)
    ! convert the julian day 2460679.45902180578559637 TDB to the julian day TT
    res = calceph_time_jd_tdb_to_jd_tt(peph, 2460679.0, 0.45902180578559637, jd0, jdfrac)
    write(*,*) jd0, jdfrac ! print 2460679.0 0.4590218056322423

    call calceph_close(peph)
endif
```

4.3.6 calceph_time_jd_tt_to_jd_tdb

```
function calceph_time_jd_tt_to_jd_tdb(eph, jd0_tt, jdfrac_tt, jd0_tdb, jdfrac_tdb) BIND(C)
```

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **jd0_tt** [*REAL(C_DOUBLE), intent(in)*] :: Integer part of Julian Date (TT)
- **jdfrac_tt** [*REAL(C_DOUBLE), intent(in)*] :: Fractional part of Julian Date (TT)

- **jd0_tdb** [*REAL(C_DOUBLE), intent(out)*] :: Integer part of Julian Date (TDB)
- **jdfrac_tdb** [*REAL(C_DOUBLE), intent(out)*] :: Fractional part of Julian Date (TDB)

Return

calceph_time_jd_tt_to_jd_tdb [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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 *calceph_time_jd_tdb_to_jd_tt()*. It uses the model configured with the function *calceph_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:

```
integer res
real(8) jd0, jdfrac
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    res = calceph_time_set_relationship_tt_tdb(peph, 1)
    ! convert the julian day 2460679.4590218056322423 TDB to the julian day TT
    res = calceph_time_jd_tt_to_jd_tdb(peph, 2460679.0, 0.4590218056322423, jd0, jdfrac)
    write(*,*) jd0, jdfrac ! print 2460679.0 0.45902180578559637

    call calceph_close(peph)
endif
```

4.3.7 calceph_time_jd_tdb_to_jd_tcb

```
function calceph_time_jd_tdb_to_jd_tcb(eph, jd0_tdb, jdfrac_tdb, jd0_tcb, jdfrac_tcb)
    BIND(C)
```

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **jd0_tdb** [*REAL(C_DOUBLE), intent(in)*] :: Integer part of Julian Date (TDB)
- **jdfrac_tdb** [*REAL(C_DOUBLE), intent(in)*] :: Fractional part of Julian Date (TDB)
- **jd0_tcb** [*REAL(C_DOUBLE), intent(out)*] :: Integer part of Julian Date (TCB)
- **jdfrac_tcb** [*REAL(C_DOUBLE), intent(out)*] :: Fractional part of Julian Date (TCB)

Return

calceph_time_jd_tdb_to_jd_tcb [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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:

```
integer res
real(8) jd0, jdfrac
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    ! convert the julian day 2450083.1267361110076308 TDB to the julian day TCB
    res = calceph_time_jd_tdb_to_jd_tcb(peph,2450083.0, 0.1267361110076308D0, jd0,
    ↪jdfrac)
    write(*,*) jd0, jdfrac ! print 2450083.0 0.1268436964601278

    call calceph_close(peph)
endif
```

4.3.8 calceph_time_jd_tcb_to_jd_tdb

```
function calceph_time_jd_tcb_to_jd_tdb(eph, jd0_tcb, jdfrac_tcb, jd0_tdb, jdfrac_tdb)
    BIND(C)
```

Parameters

- **eph** [TYPE(C_PTR), VALUE, intent(in)] :: ephemeris descriptor.
- **jd0_tcb** [REAL(C_DOUBLE), intent(in)] :: Integer part of Julian Date (TCB)
- **jdfrac_tcb** [REAL(C_DOUBLE), intent(in)] :: Fractional part of Julian Date (TCB)
- **jd0_tdb** [REAL(C_DOUBLE), intent(out)] :: Integer part of Julian Date (TDB)
- **jdfrac_tdb** [REAL(C_DOUBLE), intent(out)] :: Fractional part of Julian Date (TDB)

Return

calceph_time_jd_tcb_to_jd_tdb [INTEGER(C_INT)] :: 0 if an error occurs, otherwise non-zero value.

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

This is the inverse operation of [calceph_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:

```
integer res
real(8) jd0, jdfrac
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    ! convert the julian day 2450083.1268436964601278 TCB to the julian day TDB
    res = calceph_time_jd_tcb_to_jd_tdb(peph, 450083.0D0, 0.1268436964601278D0, jd0,
    ↪jdfrac)
    write(*,*) jd0, jdfrac ! print 22450083.0 0.1267361110076308

    call calceph_close(peph)
endif
```

4.3.9 calceph_time_cal_utc_to_jd_tdb

```
function calceph_time_cal_utc_to_jd_tdb(eph, yy, month, day, hh, min, sec, jd0_tdb,
                                         jdfrac_tdb) BIND(C)
```

Parameters

- **eph** [TYPE(C_PTR), VALUE, intent(in)] :: ephemeris descriptor.
- **yy** [INTEGER(C_INT), VALUE, intent(in)] :: Year
- **month** [INTEGER(C_INT), VALUE, intent(in)] :: Month
- **day** [INTEGER(C_INT), VALUE, intent(in)] :: Day
- **hh** [INTEGER(C_INT), VALUE, intent(in)] :: Hour
- **min** [INTEGER(C_INT), VALUE, intent(in)] :: Minute
- **sec** [REAL(C_DOUBLE), VALUE, intent(in)] :: Second
- **jd0_tdb** [REAL(C_DOUBLE), intent(out)] :: Integer part of the resulting TDB Julian Date
- **jdfrac_tdb** [REAL(C_DOUBLE), intent(out)] :: Fractional part of the resulting TDB Julian Date

Return

calceph_time_cal_utc_to_jd_tdb [INTEGER(C_INT)] :: 0 if an error occurs, otherwise non-zero value.

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 `calceph_time_jd_tt_to_jd_tdb()`.

This function requires that the ephemeris file associated to *eph* contains both leap second constants (for UTC → TT) and the necessary data for the TT → TDB transformation that can be initialized with *calceph_time_set_relationship_tt_tdb()*.

The following example converts a UTC calendar date to TDB:

```
integer res
real(8) jd0, jdfrac
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    res = calceph_time_set_relationship_tt_tdb(peph, 1)
    ! convert 2025-01-03T22:59:50.300 UTC to the julian day TDB
    res = calceph_time_cal_utc_to_jd_tdb(peph, 2025, 1, 3, 22, 59, 50.3, jd0, jdfrac)
    write(*,*) jd0, jdfrac ! print 2460679.0 0.45902180578559637

    call calceph_close(peph)
endif
```

4.3.10 calceph_time_jd_tdb_to_cal_utc

```
function calceph_time_jd_tdb_to_cal_utc(eph, jd0_tdb, jdfrac_tdb, yy, month, day, hh, min,
                                         sec) BIND(C)
```

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **jd0_tdb** [*REAL(C_DOUBLE), intent(in)*] :: Integer part of the TDB Julian Date
- **jdfrac_tdb** [*REAL(C_DOUBLE), intent(in)*] :: Fractional part of the TDB Julian Date
- **yy** [*INTEGER(C_INT), intent(out)*] :: Year
- **month** [*INTEGER(C_INT), intent(out)*] :: Month
- **day** [*INTEGER(C_INT), intent(out)*] :: Day
- **hh** [*INTEGER(C_INT), intent(out)*] :: Hour
- **min** [*INTEGER(C_INT), intent(out)*] :: Minute
- **sec** [*REAL(C_DOUBLE), intent(out)*] :: Second

Return

calceph_time_jd_tdb_to_cal_utc [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

This is the inverse operation of *calceph_time_cal_utc_to_jd_tdb()*.

The transformation chain performed is:

1. Converts TDB to TT using *calceph_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 associated to *eph* contains the necessary time constants.

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

```
integer res, yy, mo, dd, hh,mi
real(8) ss
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    res = calceph_time_set_relationship_tt_tdb(peph, 1)
    ! convert 2025-01-03T22:59:50.300 UTC to the julian day TDB
    res = calceph_time_jd_tdb_to_cal_utc(peph, 2460679.0, 0.45902180578559637, yy, mo,
    →dd, hh,mi, ss)
    write(*,*) yy, mo, dd, hh,mi, ss ! print 2025 1 3 22 59 50.300000662574196

    call calceph_close(peph)
endif
```

4.3.11 calceph_time_str_utc_to_jd_tdb

function `calceph_time_str_utc_to_jd_tdb`(*eph*, *const char *str*, *jd0_tdb*, *jdfrac_tdb*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor.
- **str** [*CHARACTER(len=*)*, *intent(in)*] :: UTC time string to parse
- **jd0_tdb** [*REAL(C_DOUBLE)*, *intent(out)*] :: Integer part of the resulting TDB Julian Date
- **jdfrac_tdb** [*REAL(C_DOUBLE)*, *intent(out)*] :: Fractional part of the resulting TDB Julian Date

Return

calceph_time_str_utc_to_jd_tdb [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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:

```
integer res
real(8) jd0, jdfrac
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then
```

(continues on next page)

(continued from previous page)

```

    res = calceph_time_set_relationship_tt_tdb(peph, 1)
    ! convert the calendar date UTC to the julian day TDB
    res = calceph_time_str_utc_to_jd_tdb(peph, "2025-01-03T22:59:50.300"//C_NULL_CHAR,
    ↪jd0, jdfraction)
    write(*,*) jd0, jdfraction ! print 2460679.0 0.45902180578559637

    call calceph_close(peph)
endif

```

4.3.12 calceph_time_str_any_to_jd_tdb

function calceph_time_str_any_to_jd_tdb(*eph*, *const char *str*, *jd0_tdb*, *jdfraction_tdb*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor.
- **str** [*CHARACTER(len=*)*, *intent(in)*] :: Time string to parse
- **jd0_tdb** [*REAL(C_DOUBLE)*, *intent(out)*] :: Integer part of the resulting TDB Julian Date
- **jdfraction_tdb** [*REAL(C_DOUBLE)*, *intent(out)*] :: Fractional part of the resulting TDB Julian Date

Return

calceph_time_str_any_to_jd_tdb [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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:

- **CALCEPH_TDB**: No conversion needed.
- **CALCEPH_TT**: Performs the transformation TT → TDB.
- **CALCEPH_TAI**: Performs the transformation chain TAI → TT → TDB.
- **CALCEPH_UTC**: Performs the transformation chain UTC → TAI → TT → TDB.

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

This function requires that the ephemeris file associated to *eph* contains the necessary data for the requested conversions (leap seconds for UTC/TAI, and relationship model for TT/TDB that can be initialized with **calceph_time_set_relationship_tt_tdb()**).

The following example converts a TAI string to TDB:

```

integer res
real(8) jd0, jdfraction
TYPE(C_PTR) :: peph

peph = calceph_open("example_lsk.tls"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

```

(continues on next page)

(continued from previous page)

```

res = calceph_time_set_relationship_tt_tdb(peph, 1)
! convert the calendar date UTC to the julian day TDB
res = calceph_time_str_any_to_jd_tdb(peph, "2025-01-03T22:59:50.300 UTC"//C_NULL_
↪CHAR, jd0, jdfrac)
write(*,*) jd0, jdfrac ! print 2460679.0 0.45902180578559637

call calceph_close(peph)
endif

```

4.3.13 calceph_time_str_spacecraft_clock_to_jd_tdb

```

function calceph_time_str_spacecraft_clock_to_jd_tdb(eph, itarget, str, jd0_tdb,
jdfrac_tdb) BIND(C)

```

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **itarget** [*INTEGER(C_INT), VALUE, intent(in)*] :: NAIF ID of the spacecraft
- **str** [*CHARACTER(len=*, intent(in))*] :: Spacecraft clock string (e.g., "1/1234:56")
- **jd0_tdb** [*REAL(C_DOUBLE), intent(out)*] :: Integer part of the resulting TDB Julian Date
- **jdfrac_tdb** [*REAL(C_DOUBLE), intent(out)*] :: Fractional part of the resulting TDB Julian Date

Return

calceph_time_str_spacecraft_clock_to_jd_tdb [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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 *calceph_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:

```

integer res
real(8) jd0, jdfrac
TYPE(C_PTR) :: peph
character(len=256), dimension (2) :: filear
filear(1) = "example_lsk.tls"//C_NULL_CHAR
filear(2) = "example_sclk.tsc"//C_NULL_CHAR
peph = calceph_open_array(2, filear, 256)

```

(continues on next page)

(continued from previous page)

```

if (C_ASSOCIATED(peph)) then
    res = calceph_time_set_relationship_tt_tdb(peph, 1)
    res = calceph_time_str_spacecraft_clock_to_jd_tdb(peph, 28, "1/0707109102:11026"/C_
↪NULL_CHAR, jd0, jdfrac)
    write(*,*) jd0, jdfrac
    call calceph_close(peph)
endif

```

4.3.14 calceph_time_jd_tdb_to_str_spacecraft_clock

```

function calceph_time_jd_tdb_to_str_spacecraft_clock(eph, itarget, jd0_tdb, jdfrac_tdb,
                                                    str) BIND(C)

```

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor.
- **itarget** [*INTEGER(C_INT), VALUE, intent(in)*] :: NAIF ID of the spacecraft
- **jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: Integer part of Julian Date (TDB)
- **jdfrac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: Fractional part of Julian Date (TDB)
- **str** [*CHARACTER(len=CALCEPH_MAX_CONSTANTVALUE), intent(out)*] :: Output spacecraft clock time string

Return

calceph_time_jd_tdb_to_str_spacecraft_clock [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

This is the inverse operation of [calceph_time_str_spacecraft_clock_to_jd_tdb\(\)](#). The input Julian Date is provided as two double-precision numbers (*jd0_tdb* and *jdfrac_tdb*).

The resulting spacecraft clock string is stored in the character array *str*. Trailing blanks are added to the character array *str*.

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 [calceph_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:

```

integer res
TYPE(C_PTR) :: peph

```

(continues on next page)

(continued from previous page)

```
character(len=CALCEPH_MAX_CONSTANTVALUE) strdate
character(len=256), dimension (2) :: filear

filear(1) = "example_lsk.tls"//C_NULL_CHAR
filear(2) = "example_sclk.tsc"//C_NULL_CHAR
peph = calceph_open_array(2, filear, 256)
if (C_ASSOCIATED(peph)) then

    res = calceph_time_set_relationship_tt_tdb(peph, 1)
    res = calceph_time_jd_tdb_to_str_spacecraft_clock(peph, 28, 2459728D0, 0.
↪ 63395731542095745681D0, strdate)
    write(*,*) strdate

    call calceph_close(peph)
endif
```


INSTRUMENT FUNCTIONS

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

5.1 calceph_getfov

function `calceph_getfov`(*eph, instrumentid, shape, frame, vector, arraybounds, nbounds*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **instrumentid** [*INTEGER(C_INT), VALUE, intent(in)*] :: Instrument identifier
- **shape** [*INTEGER(C_INT), intent(out)*] :: FOV shape
- **frame** [*CHARACTER(len=1,kind=C_CHAR), dimension(CALCEPH_MAX_CONSTANTVALUE), intent(out)*] :: Reference frame name
- **vector** [*REAL(C_DOUBLE), dimension(1:3), intent(out)*] :: Boresight vector
- **arraybounds** [*REAL(C_DOUBLE), dimension(1:3*nbounds), intent(out)*] ::
Array of boundary vectors
The length of the array has to be greater than 3*nbounds. (because each vector has 3 components)
can be NULL to query nbounds necessary
- **nbounds** [*INTEGER(C_INT), VALUE, intent(in)*] :: Maximum number of boundary vectors

Return

calceph_getfov [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise the number of boundary vectors

This function retrieves the instrument Field Of View (FOV) description from an instrument kernel (IK) of the ephemeris file associated to *eph*, 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 .

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

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

If the argument **nbounds** is **0**, the function does not write any boundary vector. Instead, it returns the number of boundary vectors required for the given instrument and FOV definition.

This allows the caller to query the required size first, allocate the appropriate array, and then call the function again to retrieve the actual boundary vectors.

Trailing blanks are added the name of the frame.

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

```

integer res, instrumentid
real(8), dimension(1:3) :: boresight_vector
real(8), dimension(1:3) :: boundary_vector
character(len=CALCEPH_MAX_CONSTANTVALUE, kind=C_CHAR) frame
TYPE(C_PTR) :: peph

peph = calceph_open("example_ik.ti"//C_NULL_CHAR)
if (C_ASSOCIATED(peph)) then

    instrumentid = -42550
    res = calceph_getfov(peph, instrumentid, shape, frame, boresight_vector, boundary_
↪vector, 3)

    write(*,*) "Instrument ", instrumentid
    write(*,*) "Shape", shape
    write(*,*) "Frame", frame
    write(*,*) "Boresight vector", boresight_vector
    write(*,*) "Boundary vector", boundary_vector

    call calceph_close(peph)
endif

```


ERROR FUNCTIONS

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

6.1 Usage

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

The example in Fortran 2003 language is `f2003error.f`.

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

```

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

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

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

stop
end

```

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

```

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

subroutine myhandler(msg, msglen) BIND(C)
  USE, INTRINSIC :: ISO_C_BINDING
  implicit none
  character(kind=C_CHAR), dimension(msglen), intent(in) :: msg
  integer(C_INT), VALUE, intent(in) :: msglen
  write (*,*) "The calceph calls the function myhandler"

```

(continues on next page)

(continued from previous page)

```

write (*,*) "The message contains ",msglen," characters"
write(*,*) "The error message is :"
write(*,*) "-----"
write(*,*) msg
write(*,*) "-----"
write(*,*) "The error handler returns"
end

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

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

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

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

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

  ! ...

  stop
end

```

6.2 calceph_seterrorhandler

subroutine calceph_seterrorhandler(*typehandler*, *userfunc*) *BIND(C)*

Parameters

- **typehandler** [*TYPE(C_INT)*, *VALUE*, *intent(in)*] :: type of handler
- **userfunc** [*TYPE(C_FUNPTR)*, *VALUE*, *intent(in)*] :: user function

This function defines the behavior of the library when an error occurs during the execution of the library's functions.

This function should be (not mandatory) called before any other functions of the library. The behavior depends on the value of *typehandler*.

The possible values for *typehandler* are :

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

If the function is called with 1 or 2 for *typehandler*, the parameter *userfunc* must be set to *C_NULL_FUNPTR*.

The function *userfunc* must be defined as

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

This function must have an explicit interface.

MISCELLANEOUS FUNCTIONS

7.1 calceph_getmaxsupportedorder

function `calceph_getmaxsupportedorder`(*segid*) *BIND(C)*

Parameters

segid [*INTEGER, intent(in)*] :: type of the segment.

Return

calceph_getmaxsupportedorder [*INTEGER(C_INT)*] :: maximal order of the computable derivatives for this type of segment.

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

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

```
integer maxorder

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

7.2 calceph_getversion_str

subroutine `calceph_getversion_str`(*version*) *BIND(C)*

Parameters

version [*CHARACTER(len=1,kind=C_CHAR), dimension(CALCEPH_MAX_CONSTANTNAME), intent(out)*] :: version of the library

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

Trailing blanks are added to the name version.

```
character(len=CALCEPH_MAX_CONSTANTNAME) version

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


WRITING FUNCTIONS

These functions are used to create and write data to binary SPK and PCK ephemerids files.

CALCEPH Library is able to write the following segment type into the SPK files.

Data representation	Time spacing	Times	Segment type	Sequential functions	Parallel functions
Chebyshev polynomials of the positions	uniform block	TDB	2	writeph_spk2_seq_wr	writeph_spk2_par_reserve and writeph_spk2_par_write
Chebyshev polynomials of the positions	uniform block	TCB	102	writeph_spk102_seq_w	writeph_spk102_par_reserve and writeph_spk102_par_write
Chebyshev polynomials of the positions and velocities	uniform block	TDB	3	writeph_spk3_seq_wr	writeph_spk3_par_reserve and writeph_spk3_par_write
Chebyshev polynomials of the positions and velocities	uniform block	TCB	103	writeph_spk103_seq_w	writeph_spk103_par_reserve and writeph_spk103_par_write
Chebyshev polynomials of the positions and velocities	non-uniform block	TDB	14	writeph_spk14_seq_block, ..._add, ..._end	
discrete states of the positions and velocities (Lagrange interpolation)	equal timesteps	TDB	8	writeph_spk8_seq_wr	writeph_spk8_par_reserve and writeph_spk8_par_write
discrete states of the positions and velocities (Lagrange interpolation)	unequal timesteps	TDB	9	writeph_spk9_seq_wr	writeph_spk9_par_reserve and writeph_spk9_par_write
discrete states of the positions and velocities (Hermite interpolation)	equal timesteps	TDB	12	writeph_spk12_seq_w	writeph_spk12_par_reserve and writeph_spk12_par_write
discrete states of the positions and velocities (Hermite interpolation)	unequal timesteps	TDB	13	writeph_spk13_seq_w	writeph_spk13_par_reserve and writeph_spk13_par_write

CALCEPH Library is able to write the following segment type into the PCK files.

Data representation	Time spacing	Timesc	Segment type	Sequential functions	Parallel functions
Chebyshev polynomials of the orientations	uniform block	TDB	2	writeph_pck2_se	writeph_pck2_par_reserve and writeph_pck2_par_write
Chebyshev polynomials of the orientations	uniform block	TCB	102	writeph_pck102_se	writeph_pck102_par_reserve and writeph_pck102_par_write
Chebyshev polynomials of the orientations and their derivatives	uniform block	TDB	3	writeph_pck3_se	writeph_pck3_par_reserve and writeph_pck3_par_write
Chebyshev polynomials of the orientations and their derivatives	uniform block	TCB	103	writeph_pck103_se	writeph_pck103_par_reserve and writeph_pck103_par_write

8.1 Thread notes

There are two modes to write SPK files : sequential and parallel.

In sequential mode, segments are written one after the other using either the function *writeph_spkN_seq_write*, or the functions *writeph_spkN_begin*, *writeph_spkN_add* and *writeph_spkN_end*, with N the type of segment.

Once *writeph_spkN_begin* is called, no other segment can be written to the file it has been called with, until *writeph_spkN_end* is called. (Even using *writeph_spkN_seq_write*.) Hence *writeph_spkN_add* is the only writing function that can be used between *writeph_spkN_begin* and *writeph_spkN_end* for a same file.

In parallel mode, a segment can be written by several threads simultaneously using the functions *writeph_spkN_par_reserve* and *writeph_spkN_par_write*, with N the type of segment. First, the function *writeph_spkN_par_reserve* is called to reserve space in the file for several segments. Then, multiple threads can call simultaneously the function *writeph_spkN_par_write* to write records in the reserved segments, **it is the responsibility of the user to ensure that no two threads write the same record of a segment**.

NB: The parallel mode is only efficient if the underlying file system supports concurrent write access to the same file from multiple threads and multithreading libraries (e.g. pthreads). Otherwise the performance may be worse than the sequential mode.

8.2 Usage

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

The example in Fortran 2003 language is `f2003writeph.f`.

8.3 SPK Functions

8.3.1 writeph_spk_create

function `writeph_spk_create`(*filename*, *ifname*, *flags*) *BIND(C)*

Parameters

- **filename** [*CHARACTER(len=1,kind=C_CHAR, intent(in))*] :: pathname of the file
- **ifname** [*CHARACTER(len=1,kind=C_CHAR, intent(in))*] :: internal filename (max 60 characters)

- **flags** [*INTEGER(C_INT), VALUE, intent(in)*] :: reserved for future extensions (must be zero)

This function creates for writing a new SPK file whose pathname is the string pointed to by *filename*, and returns an ephemeris descriptor associated to it. If a file with the same name already exists, it is overwritten.

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

The following example creates a new ephemeris file `ephemeris.bsp`

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "asteroid_1"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    ! ... compute and write to weph ...

    call writeph_close(weph)
endif
```

8.3.2 writeph_spk_open

function `writeph_spk_open(filename) BIND(C)`

Parameters

filename [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: pathname of the file

This function opens an existing SPK ephemeris file in order to append ephemeris data to it, whose pathname is the string pointed to by *filename*, and returns an ephemeris descriptor associated to it. If the file doesn't exist, or if it isn't a SPK file, the function fails and returns NULL. This file must be compliant to the format specified by the 'SPICE' SPK ephemeris file.

The function `writeph_close()` must be called to flush data to the disk and to free memory allocated by this function.

The following example appends data the ephemeris file `ephemeris.bsp`

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph

weph = writeph_spk_open("ephemeris.bsp"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    ! ... computation ...

    call writeph_close(weph)
endif
```

8.3.3 writeph_close

function `writeph_close(eph) BIND(C)`

Parameters

eph [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor

Return

writeph_close [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function closes the file associated to the ephemeris descriptor *eph*, and frees all memory allocated by the functions open/create functions. After this call, the ephemeris descriptor *eph* is no longer valid.

The following example creates a new ephemeris file *ephemeris.bsp*

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "asteroid_1"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    ! ... compute and write to weph ...

    call writeph_close(weph)
endif
```

8.3.4 writeph_dump

function writeph_dump(*eph, filename*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **filename** [*CHARACTER(len=1, kind=C_CHAR), intent(in)*] :: pathname of the file

Return

writeph_dump [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function dumps the content of the ephemeris descriptor *eph* to a binary dump file whose pathname is the string pointed to by *filename*. The state of the ephemeris descriptor *eph* can be restored from this dump file using the function [writeph_restore\(\)](#).

This function can be called to create a checkpoint restart if the creation of the file is very long.

The following example writes a first part to the ephemeris file *ephemeris.bsp* and continue from a restart checkpoint.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER(C_INT) :: ret

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "asteroid_1"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    ! ... compute and write to weph ...

    ret = writeph_dump(weph, "checkpoint_ephemeris_bsp.dump")

    ! ... compute and write to weph ...
    ! ... may be an interruption here ....

    call writeph_close(weph)
```

(continues on next page)

(continued from previous page)

```

endif

weph = writeph_spk_open("ephemeris.bsp"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    ! restart from the checkpoint
    ret = writeph_restore(weph, "checkpoint_ephemeris_bsp.dump")

    ! ... compute and write to weph ...

    call writeph_close(weph)
endif

```

8.3.5 writeph_restore

function `writeph_restore`(*eph*, *filename*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **filename** [*CHARACTER(len=1,kind=C_CHAR)*, *intent(in)*] :: pathname of the file

Return

writeph_restore [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function restores the content of the ephemeris descriptor *eph* from a binary dump file whose pathname is the string pointed to by *filename*. The dump file must have been created by the function `writeph_dump()`.

This function can be called to restart from a checkpoint restart.

The following example writes a first part to the ephemeris file `ephemeris.bsp` and continue from a restart checkpoint.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER(C_INT) :: ret

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "asteroid_1"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    ! ... compute and write to weph ...

    ret = writeph_dump(weph, "checkpoint_ephemeris_bsp.dump")

    ! ... compute and write to weph ...
    ! ... may be an interruption here ....

    call writeph_close(weph)
endif

weph = writeph_spk_open("ephemeris.bsp"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    ! restart from the checkpoint

```

(continues on next page)

(continued from previous page)

```

    ret = writeph_restore(weph, "checkpoint_ephemeris_bsp.dump")

    ! ... compute and write to weph ...

    call writeph_close(weph)
endif

```

8.3.6 writeph_comment

function `writeph_comment`(*eph*, *comment*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **comment** [*CHARACTER(len=1,kind=C_CHAR)*, *intent(in)*] :: comment to be written.

Return

writeph_comment [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes a comment to the SPK file associated to the ephemeris descriptor *eph*. The comment is a null-terminated string of characters pointed to by *comment*. This function cannot be called after writing a segment to the SPK file (i.e. after calling any of the *writeph_spkN_seq_write* or *writeph_spkN_par_reserve* functions), otherwise it will fail.

The following example adds a comment to the new ephemeris file *ephemeris.bsp*

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "asteroid_1"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    ret = writeph_comment(weph, "Created by ..."//C_NULL_CHAR)
    ret = writeph_comment(weph, "Date ..."//C_NULL_CHAR)

    ! ... compute and write to weph ...

    call writeph_close(weph)
endif

```

8.3.7 writeph_spk2_seq_write

function `writeph_spk2_seq_write`(*eph*, *target*, *center*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*, *end_frac_tdb*, *intlen_jd_tdb*, *polynomials*, *record_count*, *deg*, *segid*)
BIND(C)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The body or reference point whose coordinates will be written (see *NAIF identification numbers*).
- **center** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The origin of the coordinate system (see *NAIF identification numbers*).

- **frame** [*INTEGER(C_INT), VALUE, intent(in)*] :: reference frame (see *Frame numbers*).
- **start_jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date fraction part (TDB).
- **intlen_jd_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: length of the interpolation interval (in days, TDB).
- **polynomials** [*REAL(C_DOUBLE), dimension(*), intent(in)*] :: array of Chebyshev polynomial coefficients.
- **record_count** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of records to be written.
- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of the Chebyshev polynomials.
- **segid** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_spk2_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes, in sequential mode, a type 2 segment (Chebyshev polynomials of Chebyshev polynomials of the positions only) to the SPK file associated to the ephemeris descriptor *eph*.

The *polynomials* array must be of size $\text{record_count} * (\text{deg} + 1) * 3$ and have the following structure :

X's 1st coef	...	X's (deg + 1)th	Y's 1st coef	...	Y's (deg + 1)th	Z's 1st coef	...	Z's (deg + 1)th
1st record		coef 1st record	1st record		coef 1st record	1st record		coef 1st record
...	...	...	...	...	...	...	...	...
X's 1st coef	...	X's (deg + 1)th	Y's 1st coef	...	Y's (deg + 1)th	Z's 1st coef	...	Z's (deg + 1)th
(record_count		1)th coef	(record_count		1)th coef	(record_count		1)th coef
record		(record_count)th	record		(record_count)th	record		(record_count)th
		record			record			record

The following example creates a new ephemeris file *ephemeris.bsp* with segment of type 2 for the heliocentric coordinates of Venus.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, record_count, degree, k, ret
REAL(8), dimension(390) :: coefs ! size = 10*13*3 = record_count * (degree+1) * 3_
↪ components

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 10

```

(continues on next page)

(continued from previous page)

```

degree = 12
jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then
    do k=0, record_count-1
        ! ... fill the array coefs with the coefficients of the Chebychev polynomials .
        ↪...
        ! coefs(...) =...
    enddo
    ret = writeph_spk2_seq_write(weph, NAIFID_VENUS, NAIFID_SUN, frame, jd_start, 0.0, ↪
    ↪jd_end, 0.0,
        len_timespan, coefs, record_count, degree+1, "seg_planet"//C_NULL_
    ↪CHAR)
    call writeph_close(weph)
endif

```

8.3.8 writeph_spk2_par_reserve

function `writeph_spk2_par_reserve`(*eph*, *target_count*, *targets*, *center*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*, *end_frac_tdb*, *intlens_jd_tdb*, *record_counts*, *deg*, *segids*)

Parameters

- **eph** [*TYPE*(*C_PTR*), *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER*(*C_INT*), *dimension*(*), *intent(in)*] :: array of bodies or reference points whose coordinates will be written (see [NAIF identification numbers](#)).
- **center** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: The origin of the coordinate system (see [NAIF identification numbers](#)).
- **frame** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **intlens_jd_tdb** [*REAL*(*C_DOUBLE*), *dimension*(*), *intent(in)*] :: array containing the interpolation interval length for each target (in days, TDB).
- **record_counts** [*INTEGER*(*C_INT*), *dimension*(*), *intent(in)*] :: array of number of records for each target.

- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of the Chebyshev polynomials.
- **segids** [*CHARACTER(len=1,kind=C_CHAR), dimension(*), intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_spk2_par_reserve [*INTEGER(C_INT)*] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 2 segments (Chebyshev polynomials of the position only) in the time scale TDB to the SPK file associated to the ephemeris descriptor *eph*.

It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own interpolation interval length specified in the *intlens_jd_tdb* array, its own number of records specified in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *center*, *frame*, *start_..._tdb*, *end_..._tdb* and *deg* parameters.

The array *targets*, *intlens_jd_tdb*, *record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function `writeph_spk2_par_write()` to write in the corresponding reserved space. In addition, the function `writeph_spk2_par_write()` will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to `writeph_spk2_par_reserve` is always performed by a single thread and is followed by several calls to `writeph_spk2_par_write()` by one or several threads.

Warning

The data covers the same timespan from the date *start_jd_tdb*+*start_frac_tdb* to *end_jd_tdb*+*end_frac_tdb* for all targets, but each target may have a different number of polynomials. The time span *record_counts***intlens_jd_tdb* must be equal to the timespan defined from the date *start_jd_tdb*+*start_frac_tdb* to *end_jd_tdb*+*end_frac_tdb*. If the targets doesnot have the timespan, multiple reservations must be performed before the call to writing `writeph_spk2_par_write()`.

The following example creates a new ephemeris file ephemeris.bsp with segment of type 2 for the heliocentric coordinates of Mercury and Venus using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, degree, body, k, ret
REAL(8), dimension(39) :: coefs ! size = 13*3 = (degree+1) * 3 components
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension (2) :: segids

frame = 1 ! ICRF
target_count = 2
targets(1) = NAIFID_MERCURY
targets(2) = NAIFID_VENUS
len_timespan(1) = 32
len_timespan(2) = 16
record_counts(1) = 10
```

(continues on next page)

(continued from previous page)

```

record_counts(2) = 20
segids(1) = "seg_mercury"//C_NULL_CHAR
segids(2) = "seg_venus"//C_NULL_CHAR
degree = 12
jd_start = 2460000
jd_end = jd_start+32*10

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    reservation = writeph_spk2_seq_reserve(weph, target_count, targets, NAIFID_SUN,
                                           frame, jd_start, 0.0, jd_end, 0.0,
                                           len_timespan, record_counts, degree+1, segids)

    !$omp parallel for private(body)
    do body=0, target_count
        !$omp parallel for private(k, coefs)
        do k=0, record_counts(body)-1
            ! ... fill the array coefs with the coefficients of the Chebychev
            ↪polynomials ...
            ! coefs(...) =...
            ret = writeph_spk2_par_write(weph, reservation, body, k, 1, coefs)
        enddo
    enddo
    call writeph_close(weph)
endif

```

8.3.9 writeph_spk2_par_write

function `writeph_spk2_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*,
polynomials)

Parameters

- **eph** [*TYPE*(C_PTR), *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: number of records to be written.
- **polynomials** [*REAL*(C_DOUBLE), *dimension*(*), *intent(in)*] :: array of Chebyshev polynomial coefficients.

Return

`writeph_spk2_par_write` [*INTEGER*(C_INT)] :: 0 if an error occurs, otherwise non-zero value.

This function writes, in parallel mode, the records of a type 2 segment (Chebyshev polynomials of the position only) in the time scale TDB to the SPK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_spk2_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function *writeph_spk2_par_reserve()*, starting from 0.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment, starting from 0.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *polynomials* array must have the following structure :

X's 1st coef 1st record	...	X's (deg + 1)th coef 1st record	Y's 1st coef 1st record	...	Y's (deg + 1)th coef 1st record	Z's 1st coef 1st record	...	Z's (deg + 1)th coef 1st record
...	...	...	...	...	...	...	...	...
X's 1st coef (record_count record	...	X's (deg + 1)th coef (record_count)th record	Y's 1st coef (record_count record	...	Y's (deg + 1)th coef (record_count)th record	Z's 1st coef (record_count record	...	Z's (deg + 1)th coef (record_count)th record

With the 1st record being the record at index *record_begin_index*, and the (record_count)th record being the record at index *record_begin_index* + *record_count* - 1.

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.3.10 writeph_spk3_seq_write

function *writeph_spk3_seq_write*(*eph*, *target*, *center*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*,
end_frac_tdb, *intlen_jd_tdb*, *polynomials*, *record_count*, *deg*, *segid*)
BIND(C)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The body or reference point whose coordinates will be written (see *NAIF identification numbers*).
- **center** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The origin of the coordinate system (see *NAIF identification numbers*).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see *Frame numbers*).
- **start_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **intlen_jd_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: length of the interpolation interval (in days, TDB).
- **polynomials** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of Chebyshev polynomial coefficients.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.

- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of the Chebyshev polynomials.
- **segid** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_spk3_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes, in sequential, mode a type 3 segment (Chebyshev polynomials of the position and velocity) in the time scale TDB to the SPK file associated to the ephemeris descriptor *eph*.

The *polynomials* array must be of size `record_count*(deg+1)*6` and have the following structure :

X's	...	X's	Y's	...	Y's	Z's	...	Z's	dX's	...	dX's	dY's	...	dY's	dZ's	...	dZ's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		1st			1st			1st			1st			1st			1st
		record			record			record			record			record			record
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
X's	...	X's	Y's	...	Y's	Z's	...	Z's	dX's	...	dX's	dY's	...	dY's	dZ's	...	dZ's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		(reco			(reco			(reco			(reco			(reco			(record_count)th
		record			record			record			record			record			record

The following example creates a new ephemeris file *ephemeris.bsp* with segment of type 3 for the heliocentric coordinates of Venus.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, record_count, degree, k, ret
REAL(8), dimension(780) :: coefs ! size = 780 = record_count * (degree+1) * 6 components

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 10
degree = 12
jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    do k=0, record_count-1
        ! ... fill the array coefs with the coefficients of the Chebychev polynomials .
        ! coefs(...) =...
    end do

```

(continues on next page)

(continued from previous page)

```

    enddo
    ret = writeph_spk3_seq_write(weph, NAIFID_VENUS, NAIFID_SUN, frame, jd_start, 0.0,
    ↪jd_end, 0.0,
        len_timespan, coefs, record_count, degree+1, "seg_planet"/C_NULL_
    ↪CHAR)

    call writeph_close(weph)
endif

```

8.3.11 writeph_spk3_par_reserve

function `writeph_spk3_par_reserve`(*eph*, *target_count*, *targets*, *center*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*, *end_frac_tdb*, *intlens_jd_tdb*, *record_counts*, *deg*, *segids*)

Parameters

- **eph** [*TYPE*(C_PTR), *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER*(C_INT), *dimension*(*), *intent(in)*] :: array of bodies or reference points whose coordinates will be written (see [NAIF identification numbers](#)).
- **center** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: The origin of the coordinate system (see [NAIF identification numbers](#)).
- **frame** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL*(C_DOUBLE), *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL*(C_DOUBLE), *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL*(C_DOUBLE), *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL*(C_DOUBLE), *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **intlens_jd_tdb** :: array containing the interpolation interval length for each target (in days, TDB).
- **record_counts** [*INTEGER*(C_INT), *dimension*(*), *intent(in)*] :: array of number of records for each target.
- **deg** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: degree of the Chebyshev polynomials.
- **segids** [*CHARACTER*(len=1,kind=C_CHAR), *dimension*(*), *intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_spk3_par_reserve [*INTEGER*(C_INT)] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 3 segments (Chebyshev polynomials of the position and velocity) in the time scale TDB to the SPK file associated to the ephemeris descriptor *eph*.

It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own interpolation interval length specified in the *intlens_jd_tdb* array, its own number of records specified in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *center*, *frame*, *start_..._tdb*, *end_..._tdb* and *deg* parameters.

The array *targets*, *intlens_jd_tdb*, *record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function `writeph_spk3_par_write()` to write in the corresponding reserved space. In addition, the function `writeph_spk3_par_write()` will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to `writeph_spk3_par_reserve` is always performed by a single thread and is followed by several calls to `writeph_spk3_par_write()` by one or several threads.

Warning

The data covers the same timespan from the date *start_jd_tdb*+*start_frac_tdb* to *end_jd_tdb*+*end_frac_tdb* for all targets, but each target may have a different number of polynomials. The time span *record_counts***intlens_jd_tdb* must be equal to the timespan defined from the date *start_jd_tdb*+*start_frac_tdb* to *end_jd_tdb*+*end_frac_tdb*. If the targets doesnot have the timespan, multiple reservations must be performed before the call to writing `writeph_spk3_par_write()`.

The following example creates a new ephemeris file `ephemeris.bsp` with segment of type 3 for the heliocentric coordinates of Mercury and Venus using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, degree, body, k, ret
REAL(8), dimension(78) :: coefs ! size = 13*6 = (degree+1) * 6 components
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension (2) :: segids

frame = 1 ! ICRF
target_count = 2
targets(1) = NAIFID_MERCURY
targets(2) = NAIFID_VENUS
len_timespan(1) = 32
len_timespan(2) = 16
record_counts(1) = 10
record_counts(2) = 20
segids(1) = "seg_mercury"//C_NULL_CHAR
segids(2) = "seg_venus"//C_NULL_CHAR
degree = 12
jd_start = 2460000
jd_end = jd_start+32*10

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then
```

(continues on next page)

(continued from previous page)

```

reservation = writeph_spk3_seq_reserve(weph, target_count, targets, NAIFID_SUN,
                                     frame, jd_start, 0.0, jd_end, 0.0,
                                     len_timespan, record_counts, degree+1, segids)

!$omp parallel for private(body)
do body=0, target_count
    !$omp parallel for private(k, coefs)
    do k=0, record_counts(body)-1
        ! ... fill the array coefs with the coefficients of the Chebychev_
↪polynomials ...
        ! coefs(...) =...
        ret = writeph_spk3_par_write(weph, reservation, body, k, 1, coefs)
    enddo
enddo
call writeph_close(weph)
endif

```

8.3.12 writeph_spk3_par_write

function `writeph_spk3_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*,
polynomials) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **polynomials** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of Chebyshev polynomial coefficients.

Return

writeph_spk3_par_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in parallel mode records of a type 3 segment (Chebyshev polynomials of the position and velocity) in the time scale TDB to the SPK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_spk3_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function `writeph_spk3_par_reserve()`, starting from 0.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment, starting from 0.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *polynomials* array must be of size `record_count*(deg+1)*6` and have the following structure :

X's	...	X's	Y's	...	Y's	Z's	...	Z's	dX's	...	dX's	dY's	...	dY's	dZ's	...	dZ's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		1st			1st			1st			1st			1st			1st
		record			record			record			record			record			record
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
X's	...	X's	Y's	...	Y's	Z's	...	Z's	dX's	...	dX's	dY's	...	dY's	dZ's	...	dZ's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		(reco			(reco			(reco			(reco			(reco			(reco
		record			record			record			record			record			record
																	(record_count)th
																	record

With the 1st record being the record at index *record_begin_index*, and the (record_count)th record being the record at index *record_begin_index* + *record_count* - 1.

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.3.13 writeph_spk8_seq_write

function writeph_spk8_seq_write(*eph*, *target*, *center*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*, *end_frac_tdb*, *intlen_jd_tdb*, *polynomials*, *record_count*, *deg*, *segid*)
BIND(C)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The body or reference point whose coordinates will be written (see *NAIF identification numbers*).
- **center** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The origin of the coordinate system (see *NAIF identification numbers*).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see *Frame numbers*).
- **start_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **intlen_jd_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: length of the interpolation interval (in days, TDB).
- **states** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of state vectors.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.

- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of interpolation.
- **segid** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_spk8_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in sequential mode a type 8 segment (discrete states of the positions and velocities, Lagrange interpolation with equal timesteps) in the time scale TDB to the SPK file associated to the ephemeris descriptor *eph*.

The *states* array must be of size *record_count**6 and have the following structure :

X coord 1st record	Y coord 1st record	Z coord 1st record	dX 1st record	dY 1st record	dZ 1st record
...	...	...	...	...	...
X coord	Y coord	Z coord	dX	dY	dZ
(record_count)th record	(record_count)th record	(record_count)th record	(record_count)th record	(record_count)th record	(record_count)th record

The following example creates a new ephemeris file *ephemeris.bsp* with segment of type 8 for the heliocentric coordinates of Venus.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end, jd
INTEGER frame, record_count, interpolation_degree, k, ret
REAL(8), dimension(600) :: pos_vel ! size = 600 = record_count * 6 components

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 100
interpolation_degree = 7
jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    do k=0, record_count-1
        jd = jd_start+k*len_timespan
        ! ... fill the array pos_vel with the positions and velocities at the date jd
        ↪...
        ! pos_vel(...) =...
    enddo
    ret = writeph_spk8_seq_write(weph, NAIFID_VENUS, NAIFID_SUN, frame, jd_start, 0.0, ↪
    ↪jd_end, 0.0,
                                len_timespan, pos_vel, record_count, interpolation_degree, "seg_
    ↪planet"//C_NULL_CHAR)

    call writeph_close(weph)
endif

```

8.3.14 writeph_spk8_par_reserve

function writeph_spk8_par_reserve(*eph*, *target_count*, *targets*, *center*, *frame*, *start_jd0_tdb*, *start_frac_tdb*,
end_jd0_tdb, *end_frac_tdb*, *intlens_jd_tdb*, *record_counts*, *deg*, *segids*)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER(C_INT)*, *dimension(*)*, *intent(in)*] :: array of bodies or reference points whose coordinates will be written (see [NAIF identification numbers](#)).
- **center** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The origin of the coordinate system (see [NAIF identification numbers](#)).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **intlens_jd_tdb** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array containing the interpolation interval length for each target (in days, TDB).
- **record_counts** [*INTEGER(C_INT)*, *dimension(*)*, *intent(in)*] :: array of number of records for each target.
- **deg** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: degree of interpolation.
- **segids** [*CHARACTER(len=1,kind=C_CHAR)*, *dimension(*)*, *intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_spk8_par_reserve [*INTEGER(C_INT)*] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 8 segments (discrete states of the positions and velocities, Lagrange interpolation with equal timesteps) in the time scale TDB to the SPK file associated to the ephemeris descriptor *eph*.

It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own interpolation interval length specified in the *intlens_jd_tdb* array, its own number of records specified in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *center*, *frame*, *start_..._tdb*, *end_..._tdb* and *deg* parameters.

The array *targets*, *intlens_jd_tdb*, *record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function [writeph_spk8_par_write\(\)](#) to write in the corresponding reserved space. In addition, the function [writeph_spk8_par_write\(\)](#) will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to [writeph_spk8_par_reserve](#) is always performed by a single thread and is followed by several calls to [writeph_spk8_par_write\(\)](#) by one or several threads.

Warning

The data covers the same timespan from the date `start_jd_tdb+start_frac_tdb` to `end_jd_tdb+end_frac_tdb` for all targets, but each target may have a different number of polynomials. The time span `record_counts*intlen_jd_tdb` must be equal to the timespan defined from the date `start_jd_tdb+start_frac_tdb` to `end_jd_tdb+end_frac_tdb`. If the targets doesnot have the timespan, multiple reservations must be performed before the call to writing `writeph_spk8_par_write()`.

The following example creates a new ephemeris file `ephemeris.bsp` with segment of type 8 for the heliocentric coordinates of Mercury and Venus using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end, jd
INTEGER frame, body, k, ret
REAL(8), dimension(6) :: pos_vel ! size = 6 components
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension (2) :: segids
INTEGER interpolation_degree

frame = 1 ! ICRF
target_count = 2
targets(1) = NAIFID_MERCURY
targets(2) = NAIFID_VENUS
len_timespan(1) = 32
len_timespan(2) = 16
record_counts(1) = 10
record_counts(2) = 20
segids(1) = "seg_mercury"//C_NULL_CHAR
segids(2) = "seg_venus"//C_NULL_CHAR
interpolation_degree = 12
jd_start = 2460000
jd_end = jd_start+32*10

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    reservation = writeph_spk8_seq_reserve(weph, target_count, targets, NAIFID_SUN,
        frame, jd_start, 0.0, jd_end, 0.0,
        len_timespan, record_counts, interpolation_degree, segids)

    !$omp parallel for private(body)
    do body=0, target_count
        !$omp parallel for private(k, pos_vel, jd, ret)
        do k=0, record_counts(body)-1
            jd = jd_start+k*len_timespan(body)
            ! ... fill the array pos_vel with the positions and velocities at the date_
        ↪jd ...
```

(continues on next page)

(continued from previous page)

```

! pos_vel(...) =...
ret = writeph_spk8_par_write(weph, reservation, body, k, 1, pos_vel)
enddo
enddo
call writeph_close(weph)
endif

```

8.3.15 writeph_spk8_par_write

function `writeph_spk8_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*, *states*)
BIND(C)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **states** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of state vectors.

Return

writeph_spk8_par_write [*INTEGER(C_INT)*] :: return a reservation identifier on success, 0 on failure.

This function writes in parallel mode records of a type 8 segment (discrete states of the positions and velocities, Lagrange interpolation with equal timesteps) to the SPK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_spk8_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function `writeph_spk8_par_reserve()`, starting from 0.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment, starting from 0.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *states* array must have the following structure :

X coord 1st record	Y coord 1st record	Z coord 1st record	dX 1st record	dY 1st record	dZ 1st record
...	...	...	...	...	...
X coord	Y coord	Z coord	dX	dY	dZ
(record_count)th	(record_count)th	(record_count)th	(record_count)th	(record_count)th	(record_count)th
record	record	record	record	record	record

With the 1st record being the record at index *record_begin_index*, and the (record_count)th record being the record at index *record_begin_index* + *record_count* - 1.

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.3.16 writeph_spk9_seq_write

function writeph_spk9_seq_write(*eph*, *target*, *center*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*, *end_frac_tdb*, *states*, *epochs*, *record_count*, *deg*, *segid*) BIND(C)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The body or reference point whose coordinates will be written (see [NAIF identification numbers](#)).
- **center** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The origin of the coordinate system (see [NAIF identification numbers](#)).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **states** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of state vectors.
- **epochs** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of epochs (in Julian date TCB) one per state vector.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **deg** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: degree of interpolation.
- **segid** [*CHARACTER(len=1,kind=C_CHAR)*, *intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_spk8_par_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in sequential mode a type 9 segment (discrete states of the positions and velocities, Lagrange interpolation with unequal timesteps) to the SPK file associated to the ephemeris descriptor *eph*.

The *states* array must be of size *record_count**6 and have the following structure :

X coord 1st record	Y coord 1st record	Z coord 1st record	dX 1st record	dY 1st record	dZ 1st record
...	...	...	...	...	...
X coord (record_count)th record	Y coord (record_count)th record	Z coord (record_count)th record	dX (record_count)th record	dY (record_count)th record	dZ (record_count)th record

The *epochs* array must be of size *record_count* and contain the epochs (in Julian date TDB) corresponding to each state vector, it has the following structure :

Epoch of the 1st record
...
Epoch of the (record_count)th record

The following example creates a new ephemeris file `ephemeris.bsp` with segment of type 9 for the heliocentric coordinates of Venus.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end, jd
INTEGER frame, record_count, interpolation_degree, k, ret
REAL(8), dimension(600) :: pos_vel ! size = 600 = record_count * 6 components
REAL(8), dimension(100) :: epochs ! size = 100 = record_count states

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 100
interpolation_degree = 7
jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR,0)
if (C_ASSOCIATED(weph)) then

    do k=0, record_count-1
        ! ... fill the array pos_vel with the positions and velocities at the date
        ↪ epochs[k] ...
        ! epochs(...) = ....
        ! pos_vel(...) = ...
    enddo
    ret = writeph_spk9_seq_write(weph, NAIFID_VENUS, NAIFID_SUN, frame, jd_start, 0.0, ↪
    ↪ jd_end, 0.0,
                                len_timespan, pos_vel, epochs, record_count, interpolation_degree,
    ↪ "seg_planet"//C_NULL_CHAR)

    call writeph_close(weph)
endif

```

8.3.17 writeph_spk9_par_reserve

function `writeph_spk9_par_reserve`(*eph*, *target_count*, *targets*, *center*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*, *end_frac_tdb*, *record_counts*, *deg*, *segids*)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER(C_INT)*, *dimension(*)*, *intent(in)*] :: array of bodies or reference points whose coordinates will be written (see *NAIF identification numbers*).
- **center** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The origin of the coordinate system (see *NAIF identification numbers*).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see *Frame numbers*).

- **start_jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date fraction part (TDB).
- **record_counts** [*INTEGER(C_INT), dimension(*), intent(in)*] :: array of number of records for each target.
- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of interpolation.
- **segids** [*CHARACTER(len=1,kind=C_CHAR), dimension(*), intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_spk9_par_reserve [*INTEGER(C_INT)*] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 9 segments (discrete states of the positions and velocities, Lagrange interpolation with unequal timesteps) to the SPK file associated to the ephemeris descriptor *eph*.

It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own number of records specified in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *center*, *frame*, *start_..._tdb*, *end_..._tdb* and *deg* parameters.

The array *targets*, *record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function [writeph_spk9_par_write\(\)](#) to write in the corresponding reserved space. In addition, the function [writeph_spk9_par_write\(\)](#) will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to [writeph_spk9_par_reserve](#) is always performed by a single thread and is followed by several calls to [writeph_spk9_par_write\(\)](#) by one or several threads.

The following example creates a new ephemeris file ephemeris.bsp with segment of type 9 for the heliocentric coordinates of Mercury and Venus using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
REAL(8) :: jd_start, jd_end
INTEGER frame, body, k, ret
REAL(8), dimension(6) :: pos_vel ! size = 6 components
REAL(8), dimension(1) :: epochs ! size = 1 date
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timestep
CHARACTER(len=40), dimension(2) :: segids
INTEGER interpolation_degree

frame = 1 ! ICRF
target_count = 2
targets(1) = NAIFID_MERCURY
```

(continues on next page)

(continued from previous page)

```

targets(2) = NAIFID_VENUS
record_counts(1) = 10
record_counts(2) = 20
segids(1) = "seg_mercury"//C_NULL_CHAR
segids(2) = "seg_venus"//C_NULL_CHAR
interpolation_degree = 7
jd_start = 2460000
jd_end = jd_start+10

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR,0)
if (C_ASSOCIATED(weph)) then

    reservation = writeph_spk9_seq_reserve(weph, target_count, targets, NAIFID_SUN,
                                           frame, jd_start, 0.0, jd_end, 0.0,
                                           len_timespan, record_counts, interpolation_degree, segids)

    !$omp parallel for private(body)
    do body=0, target_count
        !$omp parallel for private(k, pos_vel)
        do k=0, record_counts(body)-1
            ! ... fill the array epochs and pos_vel with the positions and velocities
            ↪ ...
            ! epochs (0) = ...
            ! pos_vel(...) = ...
            ret = writeph_spk9_par_write(weph, reservation, body, k, 1, pos_vel,
            ↪ epochs)
        enddo
    enddo
    call writeph_close(weph)
endif

```

8.3.18 writeph_spk9_par_write

function `writeph_spk9_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*, *states*, *epochs*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **states** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of state vectors.
- **epochs** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of epochs (in Julian date TCB) one per state vector.

Return

writeph_spk9_par_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in parallel mode records of a type 9 segment (discrete states of the positions and velocities, Lagrange interpolation with unequal timesteps) to the SPK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function [writeph_spk9_par_reserve\(\)](#).

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function [writeph_spk9_par_reserve\(\)](#), starting from 0.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment, starting from 0.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *states* array must be of size *record_count**6 and have the following structure :

X coord 1st record	Y coord 1st record	Z coord 1st record	dX 1st record	dY 1st record	dZ 1st record
...	...	...	...	...	...
X coord (record_count)th record	Y coord (record_count)th record	Z coord (record_count)th record	dX (record_count)th record	dY (record_count)th record	dZ (record_count)th record

The *epochs* array must be of size *record_count* and contain the epochs (in Julian date TDB) corresponding to each state vector, it has the following structure :

Epoch of the 1st record
...
Epoch of the (record_count)th record

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.3.19 writeph_spk12_seq_write

function writeph_spk12_seq_write(*eph, target, center, frame, start_jd0_tdb, start_frac_tdb, end_jd0_tdb, end_frac_tdb, intlen_jd_tdb, states, record_count, deg, segid*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT), VALUE, intent(in)*] :: The body or reference point whose coordinates will be written (see [NAIF identification numbers](#)).
- **center** [*INTEGER(C_INT), VALUE, intent(in)*] :: The origin of the coordinate system (see [NAIF identification numbers](#)).
- **frame** [*INTEGER(C_INT), VALUE, intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date integer part (TDB).

- **end_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date fraction part (TDB).
- **intlen_jd_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: length of the interpolation interval (in days, TDB).
- **states** [*REAL(C_DOUBLE), dimension(*), intent(in)*] :: array of state vectors.
- **record_count** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of records to be written.
- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of interpolation.
- **segid** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_spk12_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in sequential mode a type 12 segment (discrete states of the positions and velocities, Hermite interpolation with equal timesteps) to the SPK file associated to the ephemeris descriptor *eph*.

The *states* array must be of size `record_count*6` and have the following structure :

X coord 1st record	Y coord 1st record	Z coord 1st record	dX 1st record	dY 1st record	dZ 1st record
...	...	...	...	...	...
X coord (record_count)th record	Y coord (record_count)th record	Z coord (record_count)th record	dX (record_count)th record	dY (record_count)th record	dZ (record_count)th record

The following example creates a new ephemeris file `ephemeris.bsp` with segment of type 12 for the heliocentric coordinates of Venus.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end, jd
INTEGER frame, record_count, interpolation_degree, k, ret
REAL(8), dimension(600) :: pos_vel ! size = 600 = record_count * 6 components

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 100
interpolation_degree = 7
jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR,0)
if (C_ASSOCIATED(weph)) then

    do k=0, record_count-1
        jd = jd_start+k*len_timespan;
        ! ... fill the array pos_vel with the positions and velocities at the date jd
        ↪ ...
        ! pos_vel(...) =...

```

(continues on next page)

(continued from previous page)

```

    enddo
    ret = writeph_spk12_seq_write(weph, NAIFID_VENUS, NAIFID_SUN, frame, jd_start, 0.0,
    ↪jd_end, 0.0,
        len_timespan, pos_vel, record_count, interpolation_degree, "seg_
    ↪planet"//C_NULL_CHAR)

    call writeph_close(weph)
endif

```

8.3.20 writeph_spk12_par_reserve

function `writeph_spk12_par_reserve`(*eph*, *target_count*, *targets*, *center*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*, *end_frac_tdb*, *intlens_jd_tdb*, *record_counts*, *deg*, *segids*)

Parameters

- **eph** [*TYPE*(*C_PTR*), *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER*(*C_INT*), *dimension*(*), *intent(in)*] :: array of bodies or reference points whose coordinates will be written (see [NAIF identification numbers](#)).
- **center** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: The origin of the coordinate system (see [NAIF identification numbers](#)).
- **frame** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **intlens_jd_tdb** [*REAL*(*C_DOUBLE*), *dimension*(*), *intent(in)*] :: array containing the interpolation interval length for each target (in days, TDB).
- **record_counts** [*INTEGER*(*C_INT*), *dimension*(*), *intent(in)*] :: array of number of records for each target.
- **deg** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: degree of interpolation.
- **segids** [*CHARACTER*(*len=1*,*kind=C_CHAR*), *dimension*(*), *intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_spk12_par_reserve [*INTEGER*(*C_INT*)] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 12 segments (discrete states of the positions and velocities, Hermite interpolation with equal timesteps) to the SPK file associated to the ephemeris descriptor *eph*. This function is similar to [writeph_spk2_par_reserve\(\)](#), except that it reserves enough space for

type 12 segments (state vectors with derivatives). In the same way `writeph_spk2_par_reserve()` is used with `writeph_spk2_par_write()`, this function must be used with `writeph_spk12_par_write()`.

It reserves space in the file for `target_count` segments, each segment associated to a target in the `targets` array. Each target has its own interpolation interval length specified in the `intlens_jd_tdb` array, its own number of records specified in the `record_counts` array, and its own segment identifier specified in the `segids` array. All the targets share the same `center`, `frame`, `start_..._tdb`, `end_..._tdb` and `deg` parameters.

The array `targets`, `intlens_jd_tdb`, `record_counts` and `segids` must be of size `target_count`.

The reservation `id` returned by this function must be used in the function `writeph_spk12_par_write()` to write in the corresponding reserved space. In addition, the function `writeph_spk12_par_write()` will require a `target_index`, which is the index of the target in the `targets` array used in this function.

A call to `writeph_spk12_par_reserve` is always performed by a single thread and is followed by several calls to `writeph_spk12_par_write()` by one or several threads.

Warning

The data covers the same timespan from the date `start_jd_tdb+start_frac_tdb` to `end_jd_tdb+end_frac_tdb` for all targets, but each target may have a different number of polynomials. The time span `record_counts*intlens_jd_tdb` must be equal to the timespan defined from the date `start_jd_tdb+start_frac_tdb` to `end_jd_tdb+end_frac_tdb`. If the targets doesnot have the timespan, multiple reservations must be performed before the call to writing `writeph_spk12_par_write()`.

The following example creates a new ephemeris file `ephemeris.bsp` with segment of type 12 for the heliocentric coordinates of Mercury and Venus using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end, jd
INTEGER frame, body, k, ret
REAL(8), dimension(6) :: pos_vel ! size = 6 components
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension(2) :: segids
INTEGER interpolation_degree

frame = 1 ! ICRF
target_count = 2
targets(1) = NAIFID_MERCURY
targets(2) = NAIFID_VENUS
len_timespan(1) = 32
len_timespan(2) = 16
record_counts(1) = 10
record_counts(2) = 20
segids(1) = "seg_mercury"//C_NULL_CHAR
segids(2) = "seg_venus"//C_NULL_CHAR
interpolation_degree = 12
jd_start = 2460000
jd_end = jd_start+32*10
```

(continues on next page)

(continued from previous page)

```

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR,0)
if (C_ASSOCIATED(weph)) then

    reservation = writeph_spk12_seq_reserve(weph, target_count, targets, NAIFID_SUN,
        frame, jd_start, 0.0, jd_end, 0.0,
        len_timespan, record_counts, interpolation_degree, segids)

    !$omp parallel for private(body)
    do body=0, target_count
        !$omp parallel for private(k, pos_vel, jd, ret)
        do k=0, record_counts(body)-1
            jd = jd_start+k*len_timespan(body)
            ! ... fill the array pos_vel with the positions and velocities at the date_
            ↪jd ...
            ! pos_vel(...) =...
            ret = writeph_spk12_par_write(weph, reservation, body, k, 1, pos_vel)
        enddo
    enddo
    call writeph_close(weph)
endif

```

8.3.21 writeph_spk12_par_write

function `writeph_spk12_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*, *states*)
BIND(C)

Parameters

- **eph** [*TYPE*(*C_PTR*), *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: number of records to be written.
- **states** [*REAL*(*C_DOUBLE*), *dimension*(*), *intent(in)*] :: array of state vectors.

Return

writeph_spk12_par_write [*INTEGER*(*C_INT*)] :: 0 if an error occurs, otherwise non-zero value.

This function writes in parallel mode records of a type 12 segment (discrete states of the positions and velocities, Hermite interpolation with equal timesteps) to the SPK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_spk8_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function `writeph_spk8_par_reserve()`, starting from 0.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment, starting from 0.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *states* array must have the following structure :

X coord 1st record	Y coord 1st record	Z coord 1st record	dX 1st record	dY 1st record	dZ 1st record
...	...	...	...	...	...
X coord (record_count)th record	Y coord (record_count)th record	Z coord (record_count)th record	dX (record_count)th record	dY (record_count)th record	dZ (record_count)th record

With the 1st record being the record at index *record_begin_index*, and the (record_count)th record being the record at index *record_begin_index* + *record_count* - 1.

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.3.22 writeph_spk13_seq_write

function writeph_spk13_seq_write(*eph, target, center, frame, start_jd0_tdb, start_frac_tdb, end_jd0_tdb, end_frac_tdb, intlen_jd_tdb, states, record_count, deg, segid*) BIND(C)

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT), VALUE, intent(in)*] :: The body or reference point whose coordinates will be written (see [NAIF identification numbers](#)).
- **center** [*INTEGER(C_INT), VALUE, intent(in)*] :: The origin of the coordinate system (see [NAIF identification numbers](#)).
- **frame** [*INTEGER(C_INT), VALUE, intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date fraction part (TDB).
- **intlen_jd_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: length of the interpolation interval (in days, TDB).
- **states** [*REAL(C_DOUBLE), dimension(*), intent(in)*] :: array of state vectors.
- **record_count** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of records to be written.
- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of interpolation.
- **segid** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_spk13_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in sequential mode a type 13 segment (discrete states of the positions and velocities, Hermite interpolation with unequal timesteps) to the SPK file associated to the ephemeris descriptor *eph*.

The *states* array must be of size `record_count*6` and have the following structure :

X coord 1st record	Y coord 1st record	Z coord 1st record	dX 1st record	dY 1st record	dZ 1st record
...	...	...	...	...	...
X coord (record_count)th record	Y coord (record_count)th record	Z coord (record_count)th record	dX (record_count)th record	dY (record_count)th record	dZ (record_count)th record

The *epochs* array must be of size `record_count` and contain the epochs (in Julian date TDB) corresponding to each state vector, it has the following structure :

Epoch of the 1st record
...
Epoch of the (record_count)th record

The following example creates a new ephemeris file *ephemeris.bsp* with segment of type 9 for the heliocentric coordinates of Venus.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end, jd
INTEGER frame, record_count, interpolation_degree, k, ret
REAL(8), dimension(600) :: pos_vel ! size = 600 = record_count * 6 components
REAL(8), dimension(100) :: epochs ! size = 100 = record_count states

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 100
interpolation_degree = 7
jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR,0)
if (C_ASSOCIATED(weph)) then
    do k=0, record_count-1
        ! ... fill the array pos_vel with the positions and velocities at the date_
        epochs[k] ...
        ! epochs (k) = ...
        ! pos_vel(...) =...
    enddo
    ret = writeph_spk13_seq_write(weph, NAIFID_VENUS, NAIFID_SUN, frame, jd_start, 0.0,
    jd_end, 0.0,
        pos_vel, epochs, record_count, interpolation_degree, "seg_planet"//
    C_NULL_CHAR)

```

(continues on next page)

(continued from previous page)

```

    call writeph_close(weph)
endif

```

8.3.23 writeph_spk13_par_reserve

```

function writeph_spk13_par_reserve(eph, target_count, targets, center, frame, start_jd0_tdb, start_frac_tdb,  

                                   end_jd0_tdb, end_frac_tdb, intlens_jd_tdb, record_counts, deg,  

                                   segids)

```

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER(C_INT), dimension(*), intent(in)*] :: array of bodies or reference points whose coordinates will be written (see [NAIF identification numbers](#)).
- **center** [*INTEGER(C_INT), VALUE, intent(in)*] :: The origin of the coordinate system (see [NAIF identification numbers](#)).
- **frame** [*INTEGER(C_INT), VALUE, intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date fraction part (TDB).
- **intlens_jd_tdb** [*REAL(C_DOUBLE), dimension(*), intent(in)*] :: array containing the interpolation interval length for each target (in days, TDB).
- **record_counts** [*INTEGER(C_INT), dimension(*), intent(in)*] :: array of number of records for each target.
- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of interpolation.
- **segids** [*CHARACTER(len=1,kind=C_CHAR), dimension(*), intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_spk13_par_reserve [*INTEGER(C_INT)*] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 13 segments (discrete states of the positions and velocities, Hermite interpolation with variable timesteps) to the SPK file associated to the ephemeris descriptor *eph*.

It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own number of records specified in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *center, frame, start_..._tdb, end_..._tdb* and *deg* parameters.

The array *targets, record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function `writeph_spk13_par_write()` to write in the corresponding reserved space. In addition, the function `writeph_spk13_par_write()` will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to `writeph_spk13_par_reserve` is always performed by a single thread and is followed by several calls to `writeph_spk13_par_write()` by one or several threads.

The following example creates a new ephemeris file `ephemeris.bsp` with segment of type 13 for the heliocentric coordinates of Mercury and Venus using OpenMP.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
REAL(8) :: jd_start, jd_end
INTEGER frame, body, k, ret
REAL(8), dimension(6) :: pos_vel ! size = 6 components
REAL(8), dimension(1) :: epochs ! size = 1 date
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension(2) :: segids
INTEGER interpolation_degree

frame = 1 ! ICRF
target_count = 2
targets(1) = NAIFID_MERCURY
targets(2) = NAIFID_VENUS
record_counts(1) = 10
record_counts(2) = 20
segids(1) = "seg_mercury"//C_NULL_CHAR
segids(2) = "seg_venus"//C_NULL_CHAR
interpolation_degree = 7
jd_start = 2460000
jd_end = jd_start+10

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR,0)
if (C_ASSOCIATED(weph)) then

    reservation = writeph_spk13_seq_reserve(weph, target_count, targets, NAIFID_SUN,
                                           frame, jd_start, 0.0, jd_end, 0.0,
                                           len_timespan, record_counts, interpolation_degree, segids)

    !$omp parallel for private(body)
    do body=0, target_count
        !$omp parallel for private(k, pos_vel)
        do k=0, record_counts(body)-1
            ! ... fill the array epochs and pos_vel with the positions and velocities
            ↪ ...
            ! epochs (0) = ...
            ! pos_vel(...) = ...
            ret = writeph_spk13_par_write(weph, reservation, body, k, 1, pos_vel,
            ↪ epochs)
        enddo
    enddo
enddo

```

(continues on next page)

(continued from previous page)

```

    call writeph_close(weph)
endif

```

8.3.24 writeph_spk13_par_write

function `writeph_spk13_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*, *states*)
BIND(C)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **states** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of state vectors.

Return

writeph_spk13_par_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in parallel mode records of a type 13 segment (discrete states of the positions and velocities, Hermite interpolation with variable timesteps) to the SPK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_spk13_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function `writeph_spk13_par_reserve()`, starting from 0.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment, starting from 0.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *states* array must be of size *record_count**6 and have the following structure :

X coord 1st record	Y coord 1st record	Z coord 1st record	dX 1st record	dY 1st record	dZ 1st record
...	...	...	...	...	...
X coord (record_count)th record	Y coord (record_count)th record	Z coord (record_count)th record	dX (record_count)th record	dY (record_count)th record	dZ (record_count)th record

The *epochs* array must be of size *record_count* and contain the epochs (in Julian date TDB) corresponding to each state vector, it has the following structure :

Epoch of the 1st record
...
Epoch of the (record_count)th record

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.3.25 writeph_spk14_begin

function writeph_spk14_begin(*eph*, *target*, *center*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*,
end_frac_tdb, *deg*, *segid*) BIND(C)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The body or reference point whose coordinates will be written (see *NAIF identification numbers*).
- **center** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The origin of the coordinate system (see *NAIF identification numbers*).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see *Frame numbers*).
- **start_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **deg** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: degree of the Chebyshev polynomials.
- **segid** [*CHARACTER(len=1,kind=C_CHAR)*, *intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_spk14_begin [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function begins the sequential writing of a type 14 segment (Chebyshev polynomials of the positions and velocities, with variable timesteps) to the SPK file associated to the ephemeris descriptor *eph*. Once this function is called, no other segment can be written to the same file, and only *writeph_spk14_add()* can be used with *eph*, until *writeph_spk14_end()* is called.

The following example creates a new ephemeris file ephemeris.bsp with segment of type 3 for the heliocentric coordinates of Venus.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, degree, k, ret
REAL(8), dimension(78) :: coefs ! size = 78 = (degree+1) * 6 components
REAL(8), dimension(1) :: epochs ! size = 1 = one date only

len_timespan = 32 ! days
frame = 1 ! ICRF
```

(continues on next page)

(continued from previous page)

```

degree = 12
jd_start = 2460000
jd_end = 2460300

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(weph)) then

    ret = writeph_spk14_seq_write(weph, NAIFID_VENUS, NAIFID_SUN, frame, jd_start, 0.0,
    ↪ jd_end, 0.0,
        degree+1, "seg_planet"//C_NULL_CHAR)

    do k=0, 100
        ! ... fill the array coefs with the coefficients of the Chebychev polynomials .
    ↪ ...
        ! coefs(...) = ...
        ! epochs(...) = ...
        ret = writeph_spk14_add(weph, 1, coefs, epochs)
    enddo
    ret = writeph_spk14_end(weph)

    call writeph_close(weph)
endif

```

8.3.26 writeph_spk14_add

function writeph_spk14_add(*eph*, *record_count*, *data*, *epochs*) BIND(C)

Parameters

- **eph** [TYPE(C_PTR), VALUE, intent(in)] :: ephemeris descriptor
- **record_count** [INTEGER(C_INT), VALUE, intent(in)] :: number of records to be written.
- **data** [REAL(C_DOUBLE), dimension(*), intent(in)] :: array data to be written.
- **epochs** [REAL(C_DOUBLE), dimension(*), intent(in)] :: array of epochs (in Julian date TCB) one per state vector.

Return

writeph_spk14_add [INTEGER(C_INT)] :: 0 if an error occurs, otherwise non-zero value.

This function adds data to the segment that has been begun with writeph_spk14_begin.

The data array must be of size $\text{record_count} \times ((\text{deg}+1) \times 6 + 2)$ and have the following structure :

- **start_jd0_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date integer part (TCB).
- **start_frac_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date fraction part (TCB).
- **end_jd0_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date integer part (TCB).
- **end_frac_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date fraction part (TCB).
- **intlen_jd_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: length of the interpolation interval (in days, TCB).
- **polynomials** [*REAL(C_DOUBLE), dimension(*), intent(in)*] :: array of Chebyshev polynomial coefficients.
- **record_count** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of records to be written.
- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of the Chebyshev polynomials.
- **segid** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_spk102_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in sequential mode a type 102 segment (Chebyshev polynomials of the position only) in the time scale TCB to the SPK file associated to the ephemeris descriptor *eph*.

The *polynomials* array must be of size $\text{record_count} * (\text{deg} + 1) * 3$ and have the following structure :

X's 1st coef 1st record	...	X's (deg + 1)th coef 1st record	Y's 1st coef 1st record	...	Y's (deg + 1)th coef 1st record	Z's 1st coef 1st record	...	Z's (deg + 1)th coef 1st record
...	...	...	...	...	...	...	...	...
X's 1st coef (record_count record	...	X's (deg + 1)th coef (record_count)th record	Y's 1st coef (record_count record	...	Y's (deg + 1)th coef (record_count)th record	Z's 1st coef (record_count record	...	Z's (deg + 1)th coef (record_count)th record

The following example creates a new ephemeris file ephemeris.bsp with segment of type 2 for the heliocentric coordinates of Venus, in the timescale TCB.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, record_count, degree, k, ret
REAL(8), dimension(390) :: coefs ! size = 10*13*3 = record_count * (degree+1) * 3_
↪ components

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 10
degree = 12

```

(continues on next page)

(continued from previous page)

```

jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(peph)) then

    do k=0, record_count-1
        ! ... fill the array coefs with the coefficients of the Chebychev polynomials .
        ↪ ...
        ! coefs(...) =...
    enddo
    ret = writeph_spk102_seq_write(weph, NAIFID_VENUS, NAIFID_SUN, frame, jd_start, 0.0,
    ↪ jd_end, 0.0,
        len_timespan, coefs, record_count, degree+1, "seg_planet"//C_NULL_
    ↪ CHAR)

    call writeph_close(peph)
endif

```

8.3.29 writeph_spk102_par_reserve

```

function writeph_spk102_par_reserve(eph, target_count, targets, center, frame, start_jd0_tcb,
                                   start_frac_tcb, end_jd0_tcb, end_frac_tcb, intlens_jd_tcb,
                                   record_counts, deg, segids)

```

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER(C_INT), dimension(*), intent(in)*] :: array of bodies or reference points whose coordinates will be written (see *NAIF identification numbers*).
- **center** [*INTEGER(C_INT), VALUE, intent(in)*] :: The origin of the coordinate system (see *NAIF identification numbers*).
- **frame** [*INTEGER(C_INT), VALUE, intent(in)*] :: reference frame (see *Frame numbers*).
- **start_jd0_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date integer part (TCB).
- **start_frac_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date fraction part (TCB).
- **end_jd0_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date integer part (TCB).
- **end_frac_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date fraction part (TCB).
- **intlens_jd_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: array containing the interpolation interval length for each target (in days, TCB).
- **record_counts** [*INTEGER(C_INT), dimension(*), intent(in)*] :: array of number of records for each target.

- **deg** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: degree of the Chebyshev polynomials.
- **segids** [*CHARACTER*(*len=1,kind=C_CHAR*), *dimension*(*), *intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_spk102_par_reserve [*INTEGER*(*C_INT*)] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 102 segments (Chebyshev polynomials of the position only) in the time scale TCB to the SPK file associated to the ephemeris descriptor *eph*. It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own interpolation interval length specified in the *intlens_jd_tcb* array, its own number of records specified in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *center*, *frame*, *start_..._tcb*, *end_..._tcb* and *deg* parameters.

The array *targets*, *intlens_jd_tcb*, *record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function `writeph_spk102_par_write()` to write in the corresponding reserved space. In addition, the function `writeph_spk102_par_write()` will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to `writeph_spk102_par_reserve` is always performed by a single thread and is followed by several calls to `writeph_spk102_par_write()` by one or several threads.

Warning

The data covers the same timespan from the date *start_jd_tcb+start_frac_tcb* to *end_jd_tcb+end_frac_tcb* for all targets, but each target may have a different number of polynomials. The time span *record_counts*intlens_jd_tcb* must be equal to the timespan defined from the date *start_jd_tcb+start_frac_tcb* to *end_jd_tcb+end_frac_tcb*. If the targets do not have the timespan, multiple reservations must be performed before the call to writing `writeph_spk3_par_write()`.

The following example creates a new ephemeris file `ephemeris.bsp` with segment of type 102 for the heliocentric coordinates of Mercury and Venus, in the timescale TCB, using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, degree, body, k, ret
REAL(8), dimension(39) :: coefs ! size = 13*3 = (degree+1) * 3 components
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension(2) :: segids

frame = 1 ! ICRF
target_count = 2
targets(1) = NAIFID_MERCURY
targets(2) = NAIFID_VENUS
len_timespan(1) = 32
len_timespan(2) = 16
record_counts(1) = 10
```

(continues on next page)

(continued from previous page)

```

record_counts(2) = 20
segids(1) = "seg_mercury"//C_NULL_CHAR
segids(2) = "seg_venus"//C_NULL_CHAR
degree = 12
jd_start = 2460000
jd_end = jd_start+32*10

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(weph)) then

    reservation = writeph_spk102_seq_reserve(weph, target_count, targets, NAIFID_SUN,
        frame, jd_start, 0.0, jd_end, 0.0,
        len_timespan, record_counts, degree+1, segids)

    !$omp parallel for private(body)
    do body=0, target_count
        !$omp parallel for private(k, coefs)
        do k=0, record_counts(body)-1
            ! ... fill the array coefs with the coefficients of the Chebychev_
            ↪polynomials ...
            ! coefs(...) =...
            writeph_spk102_par_write(weph, reservation, body, k, 1, coefs)
        enddo
    enddo
    call writeph_close(weph)
endif

```

8.3.30 writeph_spk102_par_write

function `writeph_spk102_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*,
polynomials) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **polynomials** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of Chebyshev polynomial coefficients.

Return

`writeph_spk102_par_write` [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in parallel mode records of a type 102 segment (Chebyshev polynomials of the position only) in the time scale TCB to the SPK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_spk2_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function *writeph_spk2_par_reserve()*, starting from 0.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment, starting from 0.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *polynomials* array must have the following structure :

X's 1st coef 1st record	...	X's (deg + 1)th coef 1st record	Y's 1st coef 1st record	...	Y's (deg + 1)th coef 1st record	Z's 1st coef 1st record	...	Z's (deg + 1)th coef 1st record
...	...	...	...	...	...	...	...	...
X's 1st coef (record_count record	...	X's (deg + 1)th coef (record_count)th record	Y's 1st coef (record_count record	...	Y's (deg + 1)th coef (record_count)th record	Z's 1st coef (record_count record	...	Z's (deg + 1)th coef (record_count)th record

With the 1st record being the record at index *record_begin_index*, and the (record_count)th record being the record at index *record_begin_index* + *record_count* - 1.

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.3.31 writeph_spk103_seq_write

function *writeph_spk103_seq_write*(*eph*, *target*, *center*, *frame*, *start_jd0_tcb*, *start_frac_tcb*, *end_jd0_tcb*,
end_frac_tcb, *intlen_jd_tcb*, *polynomials*, *record_count*, *deg*, *segid*)
BIND(C)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The body or reference point whose coordinates will be written (see *NAIF identification numbers*).
- **center** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The origin of the coordinate system (see *NAIF identification numbers*).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see *Frame numbers*).
- **start_jd0_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TCB).
- **start_frac_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TCB).
- **end_jd0_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TCB).
- **end_frac_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TCB).
- **intlen_jd_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: length of the interpolation interval (in days, TCB).
- **polynomials** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of Chebyshev polynomial coefficients.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.

- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of the Chebyshev polynomials.
- **segid** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_spk103_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in sequential mode a type 103 segment (Chebyshev polynomials of the position and velocity) in the time scale TCB to the SPK file associated to the ephemeris descriptor *eph*.

The *polynomials* array must be of size `record_count*(deg+1)*6` and have the following structure :

X's	...	X's	Y's	...	Y's	Z's	...	Z's	dX's	...	dX's	dY's	...	dY's	dZ's	...	dZ's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		1st			1st			1st			1st			1st			1st
		record			record			record			record			record			record
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
X's	...	X's	Y's	...	Y's	Z's	...	Z's	dX's	...	dX's	dY's	...	dY's	dZ's	...	dZ's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		(reco			(reco			(reco			(reco			(reco			(reco
		record			record			record			record			record			record
																	(record_count)th

The following example creates a new ephemeris file *ephemeris.bsp* with segment of type 3 for the heliocentric coordinates of Venus, in the timescale TCB.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, record_count, degree, k, ret
REAL(8), dimension(780) :: coefs ! size = 780 = record_count * (degree+1) * 6 components

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 10
degree = 12
jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR,0)
if (C_ASSOCIATED(peph)) then

    do k=0, record_count-1
        ! ... fill the array coefs with the coefficients of the Chebychev polynomials .
    end do
end if

```

(continues on next page)

(continued from previous page)

```

        ! coefs(...) =...
    enddo
    ret = writeph_spk103_seq_write(weph, NAIFID_VENUS, NAIFID_SUN, frame, jd_start, 0.0,
    ↪ jd_end, 0.0,
        len_timespan, coefs, record_count, degree+1, "seg_planet"//C_NULL_
    ↪ CHAR)
    call writeph_close(peph)
endif

```

8.3.32 writeph_spk103_par_reserve

function writeph_spk103_par_reserve(*eph*, *target_count*, *targets*, *center*, *frame*, *start_jd0_tcb*,
start_frac_tcb, *end_jd0_tcb*, *end_frac_tcb*, *intlens_jd_tcb*,
record_counts, *deg*, *segids*)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER(C_INT)*, *dimension(*)*, *intent(in)*] :: array of bodies or reference points whose coordinates will be written (see [NAIF identification numbers](#)).
- **center** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The origin of the coordinate system (see [NAIF identification numbers](#)).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TCB).
- **start_frac_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TCB).
- **end_jd0_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TCB).
- **end_frac_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TCB).
- **intlens_jd_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: array containing the interpolation interval length for each target (in days, TCB).
- **record_counts** [*INTEGER(C_INT)*, *dimension(*)*, *intent(in)*] :: array of number of records for each target.
- **deg** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: degree of the Chebyshev polynomials.
- **segids** [*CHARACTER(len=1,kind=C_CHAR)*, *dimension(*)*, *intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_spk103_par_reserve [*INTEGER(C_INT)*] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 103 segments (Chebyshev polynomials of the position and velocity) in the time scale TCB to the SPK file associated to the ephemeris descriptor

eph. It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own interpolation interval length specified in the *intlens_jd_tcb* array, its own number of records specified in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *center*, *frame*, *start_..._tcb*, *end_..._tcb* and *deg* parameters.

The array *targets*, *intlens_jd_tcb*, *record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function `writeph_spk103_par_write()` to write in the corresponding reserved space. In addition, the function `writeph_spk103_par_write()` will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to `writeph_spk103_par_reserve` is always performed by a single thread and is followed by several calls to `writeph_spk103_par_write()` by one or several threads.

Warning

The data covers the same timespan from the date *start_jd_tcb*+*start_frac_tcb* to *end_jd_tcb*+*end_frac_tcb* for all targets, but each target may have a different number of polynomials. The time span *record_counts***intlens_jd_tcb* must be equal to the timespan defined from the date *start_jd_tcb*+*start_frac_tcb* to *end_jd_tcb*+*end_frac_tcb*. If the targets doesnot have the timespan, multiple reservations must be performed before the call to writing `writeph_spk103_par_write()`.

The following example creates a new ephemeris file `ephemeris.bsp` with segment of type 103 for the heliocentric coordinates of Mercury and Venus, in the timescale TCB, using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, degree, body, k, ret
REAL(8), dimension(78) :: coefs ! size = 13*6 = (degree+1) * 6 components
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension (2) :: segids

frame = 1 ! ICRF
target_count = 2
targets(1) = NAIFID_MERCURY
targets(2) = NAIFID_VENUS
len_timespan(1) = 32
len_timespan(2) = 16
record_counts(1) = 10
record_counts(2) = 20
segids(1) = "seg_mercury"//C_NULL_CHAR
segids(2) = "seg_venus"//C_NULL_CHAR
degree = 12
jd_start = 2460000
jd_end = jd_start+32*10

weph = writeph_spk_create("ephemeris.bsp"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(weph)) then
```

(continues on next page)

(continued from previous page)

```

reservation = writeph_spk103_seq_reserve(weph, target_count, targets, NAIFID_SUN,
                                         frame, jd_start, 0.0, jd_end, 0.0,
                                         len_timespan, record_counts, degree+1, segids)

!$omp parallel for private(body)
do body=0, target_count
    !$omp parallel for private(k, coefs)
    do k=0, record_counts(body)-1
        ! ... fill the array coefs with the coefficients of the Chebychev_
        ↪polynomials ...
        ! coefs(...) =...
        writeph_spk103_par_write(weph, reservation, body, k, 1, coefs)
    enddo
enddo
call writeph_close(weph)
endif

```

8.3.33 writeph_spk103_par_write

function `writeph_spk103_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*,
polynomials) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **polynomials** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of Chebyshev polynomial coefficients.

Return

writeph_spk103_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in parallel mode records of a type 103 segment (Chebyshev polynomials of the position and velocity) in the time scale TCB to the SPK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_spk3_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function `writeph_spk3_par_reserve()`, starting from 0.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment, starting from 0.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *polynomials* array must have the following structure :

X's	...	X's	Y's	...	Y's	Z's	...	Z's	dX's	...	dX's	dY's	...	dY's	dZ's	...	dZ's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th
reco _r		coef	reco _r		coef	reco _r		coef	reco _r		coef	reco _r		coef	reco _r		coef
		1st			1st			1st			1st			1st			1st
		record			record			record			record			record			record
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
X's	...	X's	Y's	...	Y's	Z's	...	Z's	dX's	...	dX's	dY's	...	dY's	dZ's	...	dZ's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
(reco _r		1)th	(reco _r		1)th	(reco _r		1)th	(reco _r		1)th	(reco _r		1)th	(reco _r		1)th
reco _r		coef	reco _r		coef	reco _r		coef	reco _r		coef	reco _r		coef	reco _r		coef
		(reco _r			(reco _r			(reco _r			(reco _r			(reco _r			(reco _r
		record			record			record			record			record			record
																	(record_count)th

With the 1st record being the record at index *record_begin_index*, and the (record_count)th record being the record at index *record_begin_index* + *record_count* - 1.

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.4 PCK Functions

8.4.1 writeph_pck_create

function **writeph_pck_create**(*filename*, *ifname*, *flags*) *BIND(C)*

Parameters

- **filename** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: pathname of the file
- **ifname** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: internal filename (max 60 characters)
- **flags** [*INTEGER(C_INT), VALUE, intent(in)*] :: reserved for future extensions (must be zero)

This function creates for writing a new PCK file whose pathname is the string pointed to by *filename*, and returns an ephemeris descriptor associated to it. If a file with the same name already exists, it is overwritten.

The function *writeph_close()* must be called to free memory allocated by this function.

The function *writeph_dump()* and *writeph_restore()* can be called to create a checkpoint restart if the creation of the file is very long.

The following example creates and writes the ephemeris file *ephemeris.bpc*

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph

weph = writeph_pck_create("ephemeris.bpc"//C_NULL_CHAR, "asteroid_1"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(weph)) then
```

(continues on next page)

(continued from previous page)

```

! ... computation ...

call writeph_close(weph)
endif

```

8.4.2 writeph_pck_open

function `writeph_pck_open`(*filename*) *BIND*(*C*)

Parameters

filename [*CHARACTER*(*len=1,kind=C_CHAR*), *intent(in)*] :: pathname of the file

This function opens an existing PCK ephemeris file in order to append ephemeris data to it, whose pathname is the string pointed to by *filename*, and returns an ephemeris descriptor associated to it. If the file doesn't exist, or if it isn't a PCK file, the function fails and returns NULL. This file must be compliant to the format specified by the 'SPICE' PCK ephemeris file.

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

The following example appends data the ephemeris file `ephemeris.bpc`

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph

weph = writeph_pck_open("ephemeris.bpc"//C_NULL_CHAR)
if (C_ASSOCIATED(weph)) then

    ! ... computation ...

    call writeph_close(weph)
endif

```

The following PCK functions are to be used to write binary PCK files:

8.4.3 writeph_pck2_seq_write

function `writeph_pck2_seq_write`(*eph*, *target*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*,
end_frac_tdb, *intlen_jd_tdb*, *polynomials*, *record_count*, *deg*, *segid*)
BIND(*C*)

Parameters

- **eph** [*TYPE*(*C_PTR*), *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: The body or reference point whose coordinates will be written (see *NAIF identification numbers*).
- **frame** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: reference frame (see *Frame numbers*).
- **start_jd0_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).

- **end_frac_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date fraction part (TDB).
- **intlen_jd_tdb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: length of the interpolation interval (in days, TDB).
- **polynomials** [*REAL(C_DOUBLE), dimension(*), intent(in)*] :: array of Chebyshev polynomial coefficients.
- **record_count** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of records to be written.
- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of the Chebyshev polynomials.
- **segid** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_pck2_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in sequential mode a type 2 segment (Chebyshev polynomials of Chebyshev polynomials of the angles only) in the time scale TDB to the PCK file associated to the ephemeris descriptor *eph*.

The *polynomials* array must be of size $\text{record_count} * (\text{deg} + 1) * 3$ and have the following structure :

E1's 1st coef 1st record.	...	E1's (deg + 1)th coef 1st record.	E2's 1st coef 1st record.	...	E2's (deg + 1)th coef 1st record.	E3's 1st coef 1st record.	...	E3's (deg + 1)th coef 1st record.
...	...	...	...	...	...	...	...	...
E1's 1st coef (record_count record	...	E1's (deg + 1)th coef (record_count)th record	E2's 1st coef (record_count record	...	E2's (deg + 1)th coef (record_count)th record	E3's 1st coef (record_count record	...	E3's (deg + 1)th coef (record_count)th record

The following example creates a new ephemeris file ephemeris.bsp with segment of type 2 for the orientation of one body.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, record_count, degree, k, ret
REAL(8), dimension(390) :: coefs ! size = 10*13*3 = record_count * (degree+1) * 3_
↳ components

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 10
degree = 12
jd_start = 2460000
jd_end = jd_start + record_count * len_timespan

weph = writeph_pck_create("ephemeris.bpc"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(weph)) then

    do k=0, record_count-1
        ! ... fill the array coefs with the coefficients of the Chebychev polynomials .

```

(continues on next page)

(continued from previous page)

```

→ ...
      ! coefs(...) = ...
    enddo
    ret = writeph_pck2_seq_write(weph, 4000099, frame, jd_start, 0.0, jd_end, 0.0,
                               len_timespan, coefs, record_count, degree+1, "seg_planet"//C_NULL_
→CHAR)
    call writeph_close(weph)
endif

```

8.4.4 writeph_pck2_par_reserve

function writeph_pck2_par_reserve(*eph*, *target_count*, *targets*, *frame*, *start_jd0_tdb*, *start_frac_tdb*,
end_jd0_tdb, *end_frac_tdb*, *intlens_jd_tdb*, *record_counts*, *deg*, *segids*)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER(C_INT)*, *dimension(*)*, *intent(in)*] :: array of bodies or reference points whose coordinates will be written (see [NAIF identification numbers](#)).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **intlens_jd_tdb** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array containing the interpolation interval length for each target (in days, TDB).
- **record_counts** [*INTEGER(C_INT)*, *dimension(*)*, *intent(in)*] :: array of number of records for each target.
- **deg** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: degree of the Chebyshev polynomials.
- **segids** [*CHARACTER(len=1,kind=C_CHAR)*, *dimension(*)*, *intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_pck2_par_reserve [*INTEGER(C_INT)*] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 2 segments (Chebyshev polynomials of the angle only) in the time scale TDB to the PCK file associated to the ephemeris descriptor *eph*.

It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own interpolation interval length specified in the *intlens_jd_tdb* array, its own number of records specified

in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *frame*, *start_..._tdb*, *end_..._tdb* and *deg* parameters.

The arrays *targets*, *intlens_jd_tdb*, *record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function *writeph_pck2_par_write()* to write in the corresponding reserved space. In addition, the function *writeph_pck2_par_write()* will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to *writeph_pck2_par_reserve()* is always performed by a single thread and is followed by several calls to *writeph_pck2_par_write()* by one or several threads.

Warning

The data covers the same timespan from the date *start_jd_tdb*+*start_frac_tdb* to *end_jd_tdb*+*end_frac_tdb* for all targets, but each target may have a different number of polynomials. The time span *record_counts***intlens_jd_tdb* must be equal to the timespan defined from the date *start_jd_tdb*+*start_frac_tdb* to *end_jd_tdb*+*end_frac_tdb*. If the targets doesnot have the timespan, multiple reservations must be performed before the call to writing *writeph_pck2_par_write()*.

The following example creates a new ephemeris file *ephemeris.bsp* with segment of type 2 for the orientation of two bodies using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, degree, body, k, ret
REAL(8), dimension(39) :: coefs ! size = 13*3 = (degree+1) * 3 components
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension (2) :: segids

frame = 1 ! ICRF
target_count = 2
targets(1) = 3000099
targets(2) = 4000099
len_timespan(1) = 32
len_timespan(2) = 16
record_counts(1) = 10
record_counts(2) = 20
segids(1) = "seg_body1"//C_NULL_CHAR
segids(2) = "seg_body2"//C_NULL_CHAR
degree = 12
jd_start = 2460000
jd_end = jd_start+32*10

weph = writeph_pck_create("ephemeris.bpc"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(weph)) then

    reservation = writeph_pck2_seq_reserve(weph, target_count, targets,
                                           frame, jd_start, 0.0, jd_end, 0.0,
```

(continues on next page)

(continued from previous page)

```

len_timespan, record_counts, degree+1, segids)

!$omp parallel for private(body)
do body=0, target_count
  !$omp parallel for private(k, coefs)
  do k=0, record_counts(body)-1
    ! ... fill the array coefs with the coefficients of the Chebychev_
    ↪polynomials ...
    ! coefs(...) =...
    ret = writeph_pck2_par_write(weph, reservation, body, k, 1, coefs)
  enddo
enddo
call writeph_close(weph)
endif

```

8.4.5 writeph_pck2_par_write

function `writeph_pck2_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*,
polynomials) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **polynomials** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of Chebyshev polynomial coefficients.

Return

`writeph_pck2_par_write` [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in parallel mode records of a type 2 segment (Chebyshev polynomials of the angle only) in the time scale TDB to the PCK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_pck2_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function `writeph_pck2_par_reserve()`.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *polynomials* array must be of size `record_count*(deg+1)*3` and have the following structure :

E1's 1st coef 1st record.	...	E1's (deg + 1)th coef 1st record.	E2's 1st coef 1st record.	...	E2's (deg + 1)th coef 1st record.	E3's 1st coef 1st record.	...	E3's (deg + 1)th coef 1st record.
...	...	...	...	...	...	...	...	...
E1's 1st coef (record_count record	...	E1's (deg + 1)th coef (record_count)th record	E2's 1st coef (record_count record	...	E2's (deg + 1)th coef (record_count)th record	E3's 1st coef (record_count record	...	E3's (deg + 1)th coef (record_count)th record

With the 1st record being the record at index *record_begin_index*, and the (record_count)th record being the record at index *record_begin_index* + *record_count* - 1.

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.4.6 writeph_pck3_seq_write

function writeph_pck3_seq_write(*eph*, *target*, *frame*, *start_jd0_tdb*, *start_frac_tdb*, *end_jd0_tdb*,
end_frac_tdb, *intlen_jd_tdb*, *polynomials*, *record_count*, *deg*, *segid*)
BIND(C)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The body or reference point whose coordinates will be written (see [NAIF identification numbers](#)).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **intlen_jd_tdb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: length of the interpolation interval (in days, TDB).
- **polynomials** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of Chebyshev polynomial coefficients.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **deg** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: degree of the Chebyshev polynomials.
- **segid** [*CHARACTER(len=1,kind=C_CHAR)*, *intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_pck3_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in sequential mode a type 3 segment (Chebyshev polynomials of the angles and their derivatives) in the time scale TDB to the PCK file associated to the ephemeris descriptor *eph*.

The *polynomials* array must be of size `record_count*(deg+1)*6` and have the following structure :

E1's	...	E1's	E2's	...	E2's	E3's	...	E3's	dE1's	...	dE1's	dE2's	...	dE2's	dE3's	...	dE3's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		1st			1st			1st			1st			1st			1st
		record			record			record			record			record			record
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
E1's	...	E1's	E2's	...	E2's	E3's	...	E3's	dE1's	...	dE1's	dE2's	...	dE2's	dE3's	...	dE3's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		(reco			(reco			(reco			(reco			(reco			(reco
		record			record			record			record			record			record
																	(record_count)th
																	record

The following example creates a new ephemeris file `ephemeris.bsp` with segment of type 3 for the orientation of one body.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, record_count, degree, k, ret
REAL(8), dimension(780) :: coefs ! size = 780 = record_count * (degree+1) * 6 components

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 10
degree = 12
jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_pck_create("ephemeris.bpc"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(weph)) then

    do k=0, record_count-1
        ! ... fill the array coefs with the coefficients of the Chebychev polynomials .
        ↪ ...
        ! coefs(...) = ...
    enddo
    ret = writeph_pck3_seq_write(weph, 4000099, frame, jd_start, 0.0, jd_end, 0.0,
        len_timespan, coefs, record_count, degree+1, "seg_planet"//C_NULL_
        ↪CHAR)

    call writeph_close(weph)
endif

```

8.4.7 writeph_pck3_par_reserve

function writeph_pck3_par_reserve(*eph*, *target_count*, *targets*, *frame*, *start_jd0_tdb*, *start_frac_tdb*,
end_jd0_tdb, *end_frac_tdb*, *intlens_jd_tdb*, *record_counts*, *deg*, *segids*)

Parameters

- **eph** [*TYPE*(*C_PTR*), *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER*(*C_INT*), *dimension*(*), *intent(in)*] :: array of bodies or reference points whose coordinates will be written (see [NAIF identification numbers](#)).
- **frame** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: starting Julian date integer part (TDB).
- **start_frac_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: starting Julian date fraction part (TDB).
- **end_jd0_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: stop Julian date integer part (TDB).
- **end_frac_tdb** [*REAL*(*C_DOUBLE*), *VALUE*, *intent(in)*] :: stop Julian date fraction part (TDB).
- **intlens_jd_tdb** :: array containing the interpolation interval length for each target (in days, TDB).
- **record_counts** [*INTEGER*(*C_INT*), *dimension*(*), *intent(in)*] :: array of number of records for each target.
- **deg** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: degree of the Chebyshev polynomials.
- **segids** [*CHARACTER*(*len=1*, *kind=C_CHAR*), *dimension*(*), *intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_pck3_par_reserve [*INTEGER*(*C_INT*)] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 3 segments (Chebyshev polynomials of the angles and their derivatives) to the PCK file associated to the ephemeris descriptor *eph*.

It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own interpolation interval length specified in the *intlens_jd_tdb* array, its own number of records specified in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *frame*, *start_..._tdb*, *end_..._tdb* and *deg* parameters.

The array *targets*, *intlens_jd_tdb*, *record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function [writeph_pck3_par_write\(\)](#) to write in the corresponding reserved space. In addition, the function [writeph_pck3_par_write\(\)](#) will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to [writeph_pck3_par_reserve](#) is always performed by a single thread and is followed by several calls to [writeph_pck3_par_write\(\)](#) by one or several threads.

Warning

The data covers the same timespan from the date `start_jd_tdb+start_frac_tdb` to `end_jd_tdb+end_frac_tdb` for all targets, but each target may have a different number of polynomials. The time span `record_counts*intlens_jd_tdb` must be equal to the timespan defined from the date `start_jd_tdb+start_frac_tdb` to `end_jd_tdb+end_frac_tdb`. If the targets doesnot have the timespan, multiple reservations must be performed before the call to writing `writeph_pck3_par_write()`.

The following example creates a new ephemeris file `ephemeris.bsp` with segment of type 3 for the orientation of two bodies using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, degree, body, k, ret
REAL(8), dimension(78) :: coefs ! size = 13*6 = (degree+1) * 6 components
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension (2) :: segids

frame = 1 ! ICRF
target_count = 2
targets(1) = 3000099
targets(2) = 4000099
len_timespan(1) = 32
len_timespan(2) = 16
record_counts(1) = 10
record_counts(2) = 20
segids(1) = "seg_body1"//C_NULL_CHAR
segids(2) = "seg_body2"//C_NULL_CHAR
degree = 12
jd_start = 2460000
jd_end = jd_start+32*10

weph = writeph_pck_create("ephemeris.bpc"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(weph)) then

    reservation = writeph_pck3_seq_reserve(weph, target_count, targets,
                                           frame, jd_start, 0.0, jd_end, 0.0,
                                           len_timespan, record_counts, degree+1, segids)

    !$omp parallel for private(body)
    do body=0, target_count-1
        !$omp parallel for private(k, coefs)
        do k=0, record_counts(body)-1
            ! ... fill the array coefs with the coefficients of the Chebychev
            ↪ polynomials ...
            ! coefs(...) = ...
            ret = writeph_pck3_par_write(weph, reservation, body, k, 1, coefs)
        end do
    end do
end if
```

(continues on next page)

(continued from previous page)

```

        enddo
    enddo
    call writeph_close(weph)
endif

```

8.4.8 writeph_pck3_par_write

function `writeph_pck3_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*,
polynomials) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **polynomials** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of Chebyshev polynomial coefficients.

Return

`writeph_pck3_par_write` [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in parallel mode records of a type 3 segment (Chebyshev polynomials of the angles and their derivatives) in the time scale TDB to the PCK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_pck3_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function `writeph_pck3_par_reserve()`, starting from 0.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment, starting from 0.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *polynomials* array must be of size `record_count*(deg+1)*6` and have the following structure :

E1's	...	E1's	E2's	...	E2's	E3's	...	E3's	dE1's	...	dE1's	dE2's	...	dE2's	dE3's	...	dE3's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		1st			1st			1st			1st			1st			1st
		record			record			record			record			record			record
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
E1's	...	E1's	E2's	...	E2's	E3's	...	E3's	dE1's	...	dE1's	dE2's	...	dE2's	dE3's	...	dE3's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		(reco			(reco			(reco			(reco			(reco			(record_count)th
		record			record			record			record			record			record

With the 1st record being the record at index *record_begin_index*, and the (record_count)th record being the record at index *record_begin_index* + *record_count* - 1.

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.4.9 writeph_pck102_seq_write

function **writeph_pck102_seq_write**(*eph, target, frame, start_jd0_tcb, start_frac_tcb, end_jd0_tcb, end_frac_tcb, intlen_jd_tcb, polynomials, record_count, deg, segid*)
BIND(C)

Parameters

- **eph** [*TYPE(C_PTR), VALUE, intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT), VALUE, intent(in)*] :: The body or reference point whose coordinates will be written (see *NAIF identification numbers*).
- **frame** [*INTEGER(C_INT), VALUE, intent(in)*] :: reference frame (see *Frame numbers*).
- **start_jd0_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date integer part (TCB).
- **start_frac_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: starting Julian date fraction part (TCB).
- **end_jd0_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date integer part (TCB).
- **end_frac_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: stop Julian date fraction part (TCB).
- **intlen_jd_tcb** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: length of the interpolation interval (in days, TCB).
- **polynomials** [*REAL(C_DOUBLE), dimension(*), intent(in)*] :: array of Chebyshev polynomial coefficients.
- **record_count** [*INTEGER(C_INT), VALUE, intent(in)*] :: number of records to be written.
- **deg** [*INTEGER(C_INT), VALUE, intent(in)*] :: degree of the Chebyshev polynomials.

- **segid** [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_pck102_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in sequential mode a type 102 segment (Chebyshev polynomials of the angle only) in the time scale TCB to the PCK file associated to the ephemeris descriptor *eph*.

The *polynomials* array must be of size `record_count*(deg+1)*3` and have the following structure :

E1's 1st coef 1st record.	...	E1's (deg + 1)th coef 1st record.	E2's 1st coef 1st record.	...	E2's (deg + 1)th coef 1st record.	E3's 1st coef 1st record.	...	E3's (deg + 1)th coef 1st record.
...	...	...	...	...	...	...	...	...
E1's 1st coef (record_count record	...	E1's (deg + 1)th coef (record_count)th record	E2's 1st coef (record_count record	...	E2's (deg + 1)th coef (record_count)th record	E3's 1st coef (record_count record	...	E3's (deg + 1)th coef (record_count)th record

The following example creates a new ephemeris file *ephemeris.bsp* with segment of type 2 for the orientation of one body, in the timescale TCB.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, record_count, degree, k, ret
REAL(8), dimension(390) :: coefs ! size = 10*13*3 = record_count * (degree+1) * 3_
↳ components

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 10
degree = 12
jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_pck_create("ephemeris.bpc"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(peph)) then

    do k=0, record_count-1
        ! ... fill the array coefs with the coefficients of the Chebychev polynomials .
↳ ...
        ! coefs(...) = ...
    enddo
    ret = writeph_pck102_seq_write(weph, 4000099, frame, jd_start, 0.0, jd_end, 0.0,
        len_timespan, coefs, record_count, degree+1, "seg_planet"//C_NULL_
↳ CHAR)

    call writeph_close(peph)
endif

```

8.4.10 writeph_pck102_par_reserve

```
function writeph_pck102_par_reserve(eph, target_count, targets, frame, start_jd0_tcb, start_frac_tcb,  
                                   end_jd0_tcb, end_frac_tcb, intlens_jd_tcb, record_counts, deg,  
                                   segids)
```

Parameters

- **eph** [*TYPE*(C_PTR), *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER*(C_INT), *dimension*(*), *intent(in)*] :: array of bodies or reference points whose coordinates will be written (see [NAIF identification numbers](#)).
- **frame** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tcb** [*REAL*(C_DOUBLE), *VALUE*, *intent(in)*] :: starting Julian date integer part (TCB).
- **start_frac_tcb** [*REAL*(C_DOUBLE), *VALUE*, *intent(in)*] :: starting Julian date fraction part (TCB).
- **end_jd0_tcb** [*REAL*(C_DOUBLE), *VALUE*, *intent(in)*] :: stop Julian date integer part (TCB).
- **end_frac_tcb** [*REAL*(C_DOUBLE), *VALUE*, *intent(in)*] :: stop Julian date fraction part (TCB).
- **intlens_jd_tcb** [*REAL*(C_DOUBLE), *VALUE*, *intent(in)*] :: array containing the interpolation interval length for each target (in days, TCB).
- **record_counts** [*INTEGER*(C_INT), *dimension*(*), *intent(in)*] :: array of number of records for each target.
- **deg** [*INTEGER*(C_INT), *VALUE*, *intent(in)*] :: degree of the Chebyshev polynomials.
- **segids** [*CHARACTER*(len=1,kind=C_CHAR), *dimension*(*), *intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_pck102_par_reserve [*INTEGER*(C_INT)] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 102 segments (Chebyshev polynomials of the angle only) in the time scale TCB to the PCK file associated to the ephemeris descriptor *eph*.

It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own interpolation interval length specified in the *intlens_jd_tcb* array, its own number of records specified in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *frame*, *start_..._tcb*, *end_..._tcb* and *deg* parameters.

The arrays *targets*, *intlens_jd_tcb*, *record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function [writeph_pck102_par_write\(\)](#) to write in the corresponding reserved space. In addition, the function [writeph_pck102_par_write\(\)](#) will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to [writeph_pck102_par_reserve\(\)](#) is always performed by a single thread and is followed by several calls to [writeph_pck102_par_write\(\)](#) by one or several threads.

Warning

The data covers the same timespan from the date *start_jd_tcb+start_frac_tcb* to *end_jd_tcb+end_frac_tcb* for all targets, but each target may have a different number of polynomials. The time span *record_counts*intlens_jd_tcb* must be equal to the timespan defined from the date *start_jd_tcb+start_frac_tcb* to *end_jd_tcb+end_frac_tcb*. If the targets doesnot have the timespan, multiple reservations must be performed before the call to writing *writeph_pck102_par_write()*.

The following example creates a new ephemeris file ephemeris.bsp with segment of type 102 for the heliocentric coordinates of Mercury and Venus, in the timescale TCB, using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, degree, body, k, ret
REAL(8), dimension(39) :: coefs ! size = 13*3 = (degree+1) * 3 components
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension (2) :: segids

frame = 1 ! ICRF
target_count = 2
targets(1) = 3000099
targets(2) = 4000099
len_timespan(1) = 32
len_timespan(2) = 16
record_counts(1) = 10
record_counts(2) = 20
segids(1) = "seg_body1"//C_NULL_CHAR
segids(2) = "seg_body2"//C_NULL_CHAR
degree = 12
jd_start = 2460000
jd_end = jd_start+32*10

weph = writeph_pck_create("ephemeris.bpc"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(weph)) then

    reservation = writeph_pck102_seq_reserve(weph, target_count, targets,
                                             frame, jd_start, 0.0, jd_end, 0.0,
                                             len_timespan, record_counts, degree+1, segids)

    !$omp parallel for private(body)
    do body=0, target_count
        !$omp parallel for private(k, coefs)
        do k=0, record_counts(body)-1
            ! ... fill the array coefs with the coefficients of the Chebychev_
            ↪polynomials ...
            ! coefs(...) =...
            writeph_pck102_par_write(weph, reservation, body, k, 1, coefs)
        end do
    end do
end if
```

(continues on next page)

(continued from previous page)

```

        enddo
    enddo
    call writeph_close(weph)
endif

```

8.4.11 writeph_pck102_par_write

function `writeph_pck102_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*, *polynomials*) *BIND(C)*

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **polynomials** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of Chebyshev polynomial coefficients.

Return

`writeph_pck102_par_write` [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in parallel mode records of a type 102 segment (Chebyshev polynomials of the angle only) in the time scale TCB to the PCK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_pck102_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function `writeph_pck102_par_reserve()`.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *polynomials* array must have the following structure :

E1's 1st coef 1st record.	...	E1's (deg + 1)th coef 1st record.	E2's 1st coef 1st record.	...	E2's (deg + 1)th coef 1st record.	E3's 1st coef 1st record.	...	E3's (deg + 1)th coef 1st record.
...	...	...	...	...	...	...	...	...
E1's 1st coef (record_count record	...	E1's (deg + 1)th coef (record_count)th record	E2's 1st coef (record_count record	...	E2's (deg + 1)th coef (record_count)th record	E3's 1st coef (record_count record	...	E3's (deg + 1)th coef (record_count)th record

With the 1st record being the record at index *record_begin_index*, and the (record_count)th record being the record at index *record_begin_index* + *record_count* - 1.

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

8.4.12 writeph_pck103_seq_write

function writeph_pck103_seq_write(*eph*, *target*, *frame*, *start_jd0_tcb*, *start_frac_tcb*, *end_jd0_tcb*,
end_frac_tcb, *intlen_jd_tcb*, *polynomials*, *record_count*, *deg*, *segid*)
 BIND(C)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: The body or reference point whose coordinates will be written (see [NAIF identification numbers](#)).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see [Frame numbers](#)).
- **start_jd0_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TCB).
- **start_frac_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TCB).
- **end_jd0_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TCB).
- **end_frac_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TCB).
- **intlen_jd_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: length of the interpolation interval (in days, TCB).
- **polynomials** [*REAL(C_DOUBLE)*, *dimension(*)*, *intent(in)*] :: array of Chebyshev polynomial coefficients.
- **record_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of records to be written.
- **deg** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: degree of the Chebyshev polynomials.
- **segid** [*CHARACTER(len=1,kind=C_CHAR)*, *intent(in)*] :: segment identifier (must be unique within the file, max 40 characters).

Return

writeph_pck103_seq_write [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function writes in sequential mode a type 103 segment (Chebyshev polynomials of the angles and their derivatives) in the time scale TCB to the PCK file associated to the ephemeris descriptor *eph*.

The *polynomials* array must be of size *record_count*(deg+1)*6* and have the following structure :

E1's	...	E1's	E2's	...	E2's	E3's	...	E3's	dE1's	...	dE1's	dE2's	...	dE2's	dE3's	...	dE3's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		1st			1st			1st			1st			1st			1st
		record			record			record			record			record			record
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
E1's	...	E1's	E2's	...	E2's	E3's	...	E3's	dE1's	...	dE1's	dE2's	...	dE2's	dE3's	...	dE3's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		(reco			(reco			(reco			(reco			(reco			(record_count)th
		record			record			record			record			record			record

The following example creates a new ephemeris file ephemeris.bsp with segment of type 3 for the orientation of one body, in the timescale TCB.

```

USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, record_count, degree, k, ret
REAL(8), dimension(780) :: coefs ! size = 780 = record_count * (degree+1) * 6 components

len_timespan = 32 ! days
frame = 1 ! ICRF
record_count = 10
degree = 12
jd_start = 2460000
jd_end = jd_start+record_count*len_timespan

weph = writeph_pck_create("ephemeris.bpc"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(peph)) then

    do k=0, record_count-1
        ! ... fill the array coefs with the coefficients of the Chebychev polynomials .
        ↪ ...
        ! coefs(...) = ...
    enddo
    ret = writeph_pck103_seq_write(weph, 4000099, frame, jd_start, 0.0, jd_end, 0.0,
        ↪ len_timespan, coefs, record_count, degree+1, "seg_planet"//C_NULL_
        ↪ CHAR)

    call writeph_close(peph)
endif

```

8.4.13 writeph_pck103_par_reserve

function writeph_pck103_par_reserve(*eph*, *target_count*, *targets*, *frame*, *start_jd0_tcb*, *start_frac_tcb*, *end_jd0_tcb*, *end_frac_tcb*, *intlens_jd_tcb*, *record_counts*, *deg*, *segids*)

Parameters

- **eph** [*TYPE(C_PTR)*, *VALUE*, *intent(in)*] :: ephemeris descriptor
- **target_count** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: number of targets the reservation is done for.
- **targets** [*INTEGER(C_INT)*, *dimension(*)*, *intent(in)*] :: array of bodies or reference points whose coordinates will be written (see *NAIF identification numbers*).
- **frame** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: reference frame (see *Frame numbers*).
- **start_jd0_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date integer part (TCB).
- **start_frac_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: starting Julian date fraction part (TCB).
- **end_jd0_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date integer part (TCB).
- **end_frac_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: stop Julian date fraction part (TCB).
- **intlens_jd_tcb** [*REAL(C_DOUBLE)*, *VALUE*, *intent(in)*] :: array containing the interpolation interval length for each target (in days, TCB).
- **record_counts** [*INTEGER(C_INT)*, *dimension(*)*, *intent(in)*] :: array of number of records for each target.
- **deg** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: degree of the Chebyshev polynomials.
- **segids** [*CHARACTER(len=1,kind=C_CHAR)*, *dimension(*)*, *intent(in)*] :: array of segment identifiers (one per target, each must be unique within the file, max 40 characters).

Return

writeph_pck103_par_reserve [*INTEGER(C_INT)*] :: return a reservation identifier on success, 0 on failure.

This function is meant to enable the parallel writing (with multiple threads) of several type 103 segments (Chebyshev polynomials of the angles and their derivatives) in the time scale TCB to the PCK file associated to the ephemeris descriptor *eph*.

It reserves space in the file for *target_count* segments, each segment associated to a target in the *targets* array. Each target has its own interpolation interval length specified in the *intlens_jd_tcb* array, its own number of records specified in the *record_counts* array, and its own segment identifier specified in the *segids* array. All the targets share the same *frame*, *start_..._tcb*, *end_..._tcb* and *deg* parameters.

The array *targets*, *intlens_jd_tcb*, *record_counts* and *segids* must be of size *target_count*.

The reservation *id* returned by this function must be used in the function *writeph_pck103_par_write()* to write in the corresponding reserved space. In addition, the function *writeph_pck103_par_write()* will require a *target_index*, which is the index of the target in the *targets* array used in this function.

A call to *writeph_pck103_par_reserve* is always performed by a single thread and is followed by several calls to *writeph_pck103_par_write()* by one or several threads.

Warning

The data covers the same timespan from the date *start_jd_tcb+start_frac_tcb* to *end_jd_tcb+end_frac_tcb* for all targets, but each target may have a different number of polynomials. The time span *record_counts*intlens_jd_tcb* must be equal to the timespan defined from the date *start_jd_tcb+start_frac_tcb* to *end_jd_tcb+end_frac_tcb*. If the targets doesnot have the timespan, multiple reservations must be performed before the call to writing *writeph_pck103_par_write()*.

The following example creates a new ephemeris file ephemeris.bsp with segment of type 103 for the heliocentric coordinates of Mercury and Venus, in the timescale TCB, using OpenMP.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
TYPE(C_PTR) :: weph
INTEGER len_timespan
REAL(8) :: jd_start, jd_end
INTEGER frame, degree, body, k, ret
REAL(8), dimension(78) :: coefs ! size = 13*6 = (degree+1) * 6 components
INTEGER, dimension(2) :: record_counts, targets
INTEGER target_count, reservation
REAL(8), dimension(2) :: len_timespan
CHARACTER(len=40), dimension (2) :: segids

frame = 1 ! ICRF
target_count = 2
targets(1) = 3000099
targets(2) = 4000099
len_timespan(1) = 32
len_timespan(2) = 16
record_counts(1) = 10
record_counts(2) = 20
segids(1) = "seg_body1"//C_NULL_CHAR
segids(2) = "seg_body2"//C_NULL_CHAR
degree = 12
jd_start = 2460000
jd_end = jd_start+32*10

weph = writeph_pck_create("ephemeris.bpc"//C_NULL_CHAR, "planet_2"//C_NULL_CHAR, 0)
if (C_ASSOCIATED(weph)) then

    reservation = writeph_pck103_seq_reserve(weph, target_count, targets,
                                             frame, jd_start, 0.0, jd_end, 0.0,
                                             len_timespan, record_counts, degree+1, segids)

    !$omp parallel for private(body)
    do body=0, target_count
        !$omp parallel for private(k, coefs)
        do k=0, record_counts(body)-1
            ! ... fill the array coefs with the coefficients of the Chebychev_
            ↪polynomials ...
            ! coefs(...) =...
            writeph_pck103_par_write(weph, reservation, body, k, 1, coefs)
        end do
    end do
end if
```

(continues on next page)

(continued from previous page)

```

        enddo
    enddo
    call writeph_close(weph)
endif

```

8.4.14 writeph_pck103_par_write

function `writeph_pck103_par_write`(*eph*, *reservation*, *target_index*, *record_begin_index*, *record_count*,
polynomials)

Parameters

- **eph** [*TYPE*(*C_PTR*), *VALUE*, *intent(in)*] :: ephemeris descriptor
- **reservation** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: reservation identifier.
- **target_index** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: index of the target body in the reservation.
- **record_begin_index** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: index of the first record to be written in the target's segment.
- **record_count** [*INTEGER*(*C_INT*), *VALUE*, *intent(in)*] :: number of records to be written.
- **polynomials** [*REAL*(*C_DOUBLE*), *dimension*(*), *intent(in)*] :: array of Chebyshev polynomial coefficients.

Return

writeph_pck103_seq_write [*INTEGER*(*C_INT*)] :: 0 if an error occurs, otherwise non-zero value.

This function writes in parallel mode records of a type 103 segment (Chebyshev polynomials of the angles and their derivatives) in the time scale TCB to the PCK file associated to the ephemeris descriptor *eph*.

The *reservation* parameter must be the reservation id returned by the function `writeph_pck3_par_reserve()`.

The *target_index* parameter is the index of the target in the *targets* array that was passed to the function `writeph_pck3_par_reserve()`, starting from 0.

The *record_begin_index* parameter is the index of the first record to be written in the target's segment, starting from 0.

The *record_count* parameter is the number of records to be written starting from the *record_begin_index*.

The *polynomials* array must be of size `record_count*(deg+1)*6` and have the following structure :

E1's	...	E1's	E2's	...	E2's	E3's	...	E3's	dE1's	...	dE1's	dE2's	...	dE2's	dE3's	...	dE3's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th	1st		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		1st			1st			1st			1st			1st			1st
		record			record			record			record			record			record
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
E1's	...	E1's	E2's	...	E2's	E3's	...	E3's	dE1's	...	dE1's	dE2's	...	dE2's	dE3's	...	dE3's
1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg	1st		(deg
coef		+	coef		+	coef		+	coef		+	coef		+	coef		+
(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th	(reco		1)th
reco		coef	reco		coef	reco		coef	reco		coef	reco		coef	reco		coef
		(reco			(reco			(reco			(reco			(reco			(record_count)th
		record			record			record			record			record			record

With the 1st record being the record at index *record_begin_index*, and the (record_count)th record being the record at index *record_begin_index* + *record_count* - 1.

Multiple threads can call this function at the same time, but with different values of *target_index* and/or *record_begin_index*. The behavior is undefined if multiple threads overlap the written data : e.g., two threads write to the same target *target_index* and a common record number.

SINGLE FILE ACCESS FUNCTIONS

Warning

Deprecated since version 4.0: Use *Reading/Evaluation functions* instead.

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

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

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

could be replaced by

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

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

When an error occurs, these functions execute error handlers according to the behavior defined by the function *calceph_seterrorhandler()*.

The python interface does not provide these functions, the listed in the section *Reading/Evaluation functions* must be used.

9.1 Time notes

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

9.2 Thread notes

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

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

If the library was configured with the default option *-DENABLE_THREAD=OFF*, these functions use an internal global variable per process and are not thread-safe. If multiple threads are used in the process and call the function *calceph_scompute()* at the same time, the caller thread must surround the call to this function with locking primitives, such as *pthread_lock/pthread_unlock* if POSIX Pthreads are used.

9.3 Usage

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

The example in Fortran 2003 language is *f2003single.f*.

```
USE, INTRINSIC :: ISO_C_BINDING
use calceph
implicit none
integer res
real(8) AU, EMRAT, GM_Mer
real(8) jd0
real(8) dt
real(8) PV(6)
integer j
real(8) valueconstant
character(len=CALCEPH_MAX_CONSTANTNAME) nameconstant

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

  ! compute and print the coordinates
  ! the geocentric moon coordinates
  res = calceph_scompute(jd0, dt, 10, 3, PV)
  call printcoord(PV, "geocentric coordinates of the Moon")

```

(continues on next page)

(continued from previous page)

```

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

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

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

```

9.4 Functions

9.4.1 calceph_sopen

function `calceph_sopen(filename) BIND(C)`

Parameters

filename [*CHARACTER(len=1,kind=C_CHAR), intent(in)*] :: pathname of the file.

Return

calceph_sopen [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

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

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

The following example opens the ephemeris file example1.dat

```

integer res
res = calceph_sopen("example1.dat"//C_NULL_CHAR)
if (res.eq.1) then
    ! ... computation ...
endif
call calceph_sclose

```

9.4.2 calceph_scompute

function `calceph_scompute`(*JD0, time, target, center, PV*) *BIND(C)*

Parameters

- **JD0** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: Integer part of the Julian date (TDB or TCB)
- **time** [*REAL(C_DOUBLE), VALUE, intent(in)*] :: Fraction part of the Julian date (TDB or TCB)
- **target** [*INTEGER(C_INT), VALUE, intent(in)*] :: The body or reference point whose coordinates are required (see the list, below).
- **center** [*INTEGER(C_INT), VALUE, intent(in)*] :: The origin of the coordinate system (see the list, below). If *target* is 14, 15, 16 or 17 (nutration, libration, TT-TDB or TCG-TCB), *center* must be 0.
- **PV** [*REAL(C_DOUBLE), dimension(1:6), intent(out)*] :: Depending on the target value, an array to receive the cartesian position (x,y,z) and the velocity (xdot, ydot, zdot), or a time scale transformation value, or the angles of the librations of the Moon and their derivatives, or the nutation angles and their derivatives.

Return

`calceph_scompute` [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

Warning

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

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

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

value	meaning
1	Mercury Barycenter
2	Venus Barycenter
3	Earth
4	Mars Barycenter
5	Jupiter Barycenter
6	Saturn Barycenter
7	Uranus Barycenter
8	Neptune Barycenter
9	Pluto Barycenter
10	Moon
11	Sun
12	Solar Sytem barycenter
13	Earth-moon barycenter
14	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=2451624.5 and at 2451624.9

```

integer res
real(8) jd0
real(8) dt1, dt2
real(8) PV(6)

jd0 = 2442457
dt1 = 0.5D0
dt2 = 0.9D0

res = calceph_sopen("example1.dat"//C_NULL_CHAR)
if (res.eq.1) then
    ! the heliocentric coordinates of Mars
    res = calceph_scompute(jd0, dt1, 4, 11, PV)
    write (*,*) PV

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

    call calceph_sclose
endif

```

9.4.3 calceph_sgetconstant

function calceph_sgetconstant(*name*, *value*) *BIND(C)*

Parameters

- **name** [*CHARACTER(len=1,kind=C_CHAR, intent(in))*] :: name of the constant.
- **value** [*REAL(C_DOUBLE), intent(out)*] :: first value of the constant.

Return

calceph_sgetconstant [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

The function [calceph_sopen\(\)](#) must be previously called before.

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

```
integer res
real(8) AU
! open the ephemeris file
res = calceph_sopen("example1.dat"//C_NULL_CHAR)
if (res.eq.1) then
  if (calceph_sgetconstant("AU"//C_NULL_CHAR, AU).eq.1) then
    write (*,*) "AU=", AU
  endif
endif
endif
```

9.4.4 calceph_sgetconstantcount

function calceph_sgetconstantcount() *BIND(C)*

Return

calceph_sgetconstantcount [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

The function [calceph_sopen\(\)](#) must be previously called before.

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

```
integer res
integer n
res = calceph_sopen("example1.dat"//C_NULL_CHAR)
if (res.eq.1) then

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

  call calceph_sclose
endif
```

9.4.5 calceph_sgetconstantindex

function `calceph_sgetconstantindex`(*index*, *name*, *value*) *BIND(C)*

Parameters

- **index** [*INTEGER(C_INT)*, *VALUE*, *intent(in)*] :: index of the constant, between 1 and `calceph_getconstantcount()`
- **name** [*CHARACTER(len=1,kind=C_CHAR)*, *dimension(CALCEPH_MAX_CONSTANTNAME)*, *intent(out)*] :: name of the constant.
- **value** [*REAL(C_DOUBLE)*, *intent(out)*] :: first value of the constant

Return

calceph_sgetconstantindex [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

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

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

res = calceph_sopen("example1.dat"//C_NULL_CHAR)
if (res.eq.1) then

  do j=1, calceph_sgetconstantcount()
    res = calceph_sgetconstantindex(j,nameconstant,valueconstant)
    write (*,*) nameconstant,"=",valueconstant
  enddo

call calceph_sclose
```

9.4.6 calceph_sgetfileversion

function `calceph_sgetfileversion`(*version*) *BIND(C)*

Parameters

version [*CHARACTER(len=1,kind=C_CHAR)*, *dimension(CALCEPH_MAX_CONSTANTVALUE)*, *intent(out)*] :: version of the file

Return

calceph_sgetfileversion [*INTEGER(C_INT)*] :: returns 0 if the file version was not found, otherwise non-zero value.

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

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

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

The following example prints the version of the ephemeris file.

```
integer res
character(len=CALCEPH_MAX_CONSTANTVALUE) version
res = calceph_sopen("example1.dat"//C_NULL_CHAR)
if (res.eq.1) then

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

    call calceph_sclose
endif
```

9.4.7 calceph_sgettimescale

function calceph_sgettimescale(*BIND(C)*)

Return

calceph_sgettimescale [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

This function returns the timescale of the ephemeris file :

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

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

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

```
integer res
integer t
res = calceph_sopen("example1.dat"//C_NULL_CHAR)
if (res.eq.1) then

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

    call calceph_sclose
endif
```

9.4.8 calceph_sgettimespan

function calceph_sgettimespan(*firsttime, lasttime, continuous BIND(C)*)

Parameters

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

Return

calceph_sgettimespan [*INTEGER(C_INT)*] :: 0 if an error occurs, otherwise non-zero value.

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

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

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

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

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

```
integer res
integer cont
real(8) jdfirst, jdlast
res = calceph_sopen("example1.dat"//C_NULL_CHAR)
if (res.eq.1) then

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

    call calceph_sclose
endif
```

9.4.9 calceph_sclose

subroutine `calceph_sclose()`

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

NAIF IDENTIFICATION NUMBERS

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

This list is already predefined in the module file `calceph.mod`.

10.1 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

10.2 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

10.3 Planet centers and satellites

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

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

Predefined Macros	NAIF ID	Name
NAIFID_MERCURY	199	Mercury
NAIFID_VENUS	299	Venus
NAIFID_EARTH	399	Earth
NAIFID_MOON	301	Moon
NAIFID_MARS	499	Mars
NAIFID_PHOBOS	401	Phobos
NAIFID_DEIMOS	402	Deimos
NAIFID_JUPITER	599	Jupiter
NAIFID_IO	501	Io
NAIFID_EUROPA	502	Europa
NAIFID_GANYMEDE	503	Ganymede
NAIFID_CALLISTO	504	Callisto
NAIFID_AMALTHEA	505	Amalthea
NAIFID_HIMALIA	506	Himalia
NAIFID_ELARA	507	Elara
NAIFID_PASIPHAЕ	508	Pasiphae
NAIFID_SINOPE	509	Sinope
NAIFID_LYSITHEA	510	Lysithea
NAIFID_CARME	511	Carme
NAIFID_ANANKE	512	Ananke
NAIFID_LEDA	513	Leda
NAIFID_THEBE	514	Thebe
NAIFID_ADRASTEА	515	Adrastea
NAIFID_METIS	516	Metis
NAIFID_CALLIRRHOE	517	Callirrhoe
NAIFID_THEMISTO	518	Themisto
NAIFID_MEGACLITE	519	Megaclite
NAIFID_TAYGETE	520	Taygete
NAIFID_CHALDENE	521	Chaldene
NAIFID_HARPALYKE	522	Harpalyke
NAIFID_KALYKE	523	Kalyke
NAIFID_IOCASTE	524	Iocaste
NAIFID_ERINOME	525	Erinome
NAIFID_ISONOE	526	Isonoe
NAIFID_PRAXIDIKE	527	Praxidike
NAIFID_AUTONOE	528	Autonoe
NAIFID_THYONE	529	Thyone
NAIFID_HERMIPPE	530	Hermippe
NAIFID_AITNE	531	Aitne
NAIFID_EURYDOME	532	Eurydome
NAIFID_EUANTHE	533	Euanthe
NAIFID_EUPORIE	534	Euporie
NAIFID_ORTHOSIE	535	Orthosie
NAIFID_SPONDE	536	Sponde
NAIFID_KALE	537	Kale
NAIFID_PASITHEE	538	Pasithee
NAIFID_HEGEMONE	539	Hegemone

continues on next page

Table 1 – continued from previous page

Predefined Macros	NAIF ID	Name
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
NAIFID_POLYDEUCES	634	Polydeuces
NAIFID_DAPHNIS	635	Daphnis
NAIFID_AEGIR	636	Aegir

continues on next page

Table 1 – continued from previous page

Predefined Macros	NAIF ID	Name
NAIFID_BEBHIONN	637	Bebhionn
NAIFID_BERGELMIR	638	Bergelmir
NAIFID_BESTLA	639	Bestla
NAIFID_FARBAUTI	640	Farbauti
NAIFID_FENRIR	641	Fenrir
NAIFID_FORNJOT	642	Fornjot
NAIFID_HATI	643	Hati
NAIFID_HYROKKIN	644	Hyrokkin
NAIFID_KARI	645	Kari
NAIFID_LOGE	646	Loge
NAIFID_SKOLL	647	Skoll
NAIFID_SURTUR	648	Surtur
NAIFID_ANTHE	649	Anthe
NAIFID_JARNSAXA	650	Jarnsaxa
NAIFID_GREIP	651	Greip
NAIFID_TARQEQ	652	Tarqeq
NAIFID_AEGAEON	653	Aegaeon
NAIFID_URANUS	799	Uranus
NAIFID_ARIEL	701	Ariel
NAIFID_UMBRIEL	702	Umbriel
NAIFID_TITANIA	703	Titania
NAIFID_OBERON	704	Oberon
NAIFID_MIRANDA	705	Miranda
NAIFID_CORDELIA	706	Cordelia
NAIFID_OPHELIA	707	Ophelia
NAIFID_BIANCA	708	Bianca
NAIFID_CRESSIDA	709	Cressida
NAIFID_DESDEMONA	710	Desdemona
NAIFID_JULIET	711	Juliet
NAIFID_PORTIA	712	Portia
NAIFID_ROSALIND	713	Rosalind
NAIFID_BELINDA	714	Belinda
NAIFID_PUCK	715	Puck
NAIFID_CALIBAN	716	Caliban
NAIFID_SYCORAX	717	Sycorax
NAIFID_PROSPERO	718	Prospero
NAIFID_SETEBOS	719	Setebos
NAIFID_STEPHANO	720	Stephano
NAIFID_TRINCULO	721	Trinculo
NAIFID_FRANCISCO	722	Francisco
NAIFID_MARGARET	723	Margaret
NAIFID_FERDINAND	724	Ferdinand
NAIFID_PERDITA	725	Perdita
NAIFID_MAB	726	Mab
NAIFID_CUPID	727	Cupid
NAIFID_NEPTUNE	899	Neptune
NAIFID_TRITON	801	Triton
NAIFID_NEREID	802	Nereid

continues on next page

Table 1 – continued from previous page

Predefined Macros	NAIF ID	Name
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

10.4 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
NAIFID_DUBIAGO	1000024	Dubiago
NAIFID_ENCKE	1000025	Encke
NAIFID_FAYE	1000026	Faye

continues on next page

Table 2 – continued from previous page

Predefined Macros	NAIF ID	Name
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
NAIFID_RUSSELL_3	1000074	Russell 3
NAIFID_RUSSELL_4	1000075	Russell 4
NAIFID_SANGUIN	1000076	Sanguin

continues on next page

Table 2 – continued from previous page

Predefined Macros	NAIF ID	Name
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_NIJIMA	1000113	Urata-Nijima
NAIFID_WISEMAN_SKIFF	1000114	Wiseman-Skiff
NAIFID_HELIN	1000115	Helin
NAIFID_MUELLER	1000116	Mueller
NAIFID_SHOEMAKER_HOLT_1	1000117	Shoemaker-Holt 1
NAIFID_HELIN_ROMAN_CROCKETT	1000118	Helin-Roman-Crockett
NAIFID_HARTLEY_3	1000119	Hartley 3
NAIFID_PARKER_HARTLEY	1000120	Parker-Hartley
NAIFID_HELIN_ROMAN_ALU_1	1000121	Helin-Roman-Alu 1
NAIFID_WILD_4	1000122	Wild 4
NAIFID_MUELLER_2	1000123	Mueller 2
NAIFID_MUELLER_3	1000124	Mueller 3
NAIFID_SHOEMAKER_LEVY_1	1000125	Shoemaker-Levy 1
NAIFID_SHOEMAKER_LEVY_2	1000126	Shoemaker-Levy 2

continues on next page

Table 2 – continued from previous page

Predefined Macros	NAIF ID	Name
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

FRAME NUMBERS

value	Name
1	ICRF

RELEASE NOTES

- **Version 5.0.1**

Support the computation of the acceleration and jerk for the segment 1 and 21 in the SPICE kernel file.
Check the Fortran/C Compiler compatibility and emit a cmake warning during the compilation.
Fix the compilation if the Fortran compiler does not add a trailing underscore.
Fix the compilation if the C compiler supports the ISO C11 multi-threading standard on windows.
Fix the compilation with cmake and Visual Studio generators.
Fix the compilation if `ENABLE_INSTALL=OFF` or `ENABLE_THREAD=OFF`.
Remove compilation warnings about missing prototypes, duplicated declarations and implicit conversions.
Fix the reduction of the stack size of the arrays for the evaluation with C99 compilers.

- **Version 5.0.0**

Support the Leapseconds Clock Kernel files (spice files .tls).
Support the Sspacecraft 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` $\geq 3.1.0$.

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, parenthesis, 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 environment MinGW.

- **Version 3.5.1**

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

- **Version 3.5.0**

Add the function calceph_getmaxsupportedorder, calceph_getpositionrecordindex2 and calceph_getorientrecordindex2.
 Update config.sub and config.guess to support new processors.
 Fix an incorrect result of calceph_(s)getconstant and calceph_getconstantvd with SPICE text kernel if the current locale has a decimal separator different than the decimal point (e.g. comma).
 Fix compilation warnings with gcc 11 about pointers and arrays.
 Fix the installation with python 3.10 or later (PEP 632).
 Fix compilation errors with the android SDK API level < 24.
 Reduce the stack size of the arrays for the evaluation with C99 compilers.
 Change the array size of the function calceph_getfileversion from the value CALCEPH_MAX_CONSTANTNAME to CALCEPH_MAX_CONSTANTVALUE, to conform to the documentation and examples.
 Only C and Fortran 2003 are affected by this change : previous header file contains an invalid declaration of the function.
 Few programs may be affected by this change, as the required array has a smaller size : No buffer overflow may appear.

- **Version 3.4.7**

Fix a decode error of the little-endian SPICE kernel files on the big-endian architectures (e.g. processor s390x).
 Fix the transmission of the flags FCFLAGS to the fortran compilers.

- **Version 3.4.6**

Fix a wrong error message about unsupported order for the segment 21.
 Fix incorrect results for SPICE kernel files containing segments of type 21 with many records (>=100) and improved the accuracy if segments of type 21 contain few records (<100).

- **Version 3.4.5**

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

- **Version 3.4.4**

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

- **Version 3.4.3**

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

Fix the installation of python package under Anaconda.

- **Version 3.4.2**

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

Support SPICE kernels larger than 4GBytes.

- **Version 3.4.1**

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

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

- **Version 3.4.0**

Add the function `calceph_isthreadsafe`.

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

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

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

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

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

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

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

Add the utilities `calceph_queryposition` and `calceph_queryorientation`.

- **Version 3.3.1**

Fix the installation with python 3.7.0 or later.

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

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

- **Version 3.3.0**

Add the functions `calceph_getfileversion`.

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

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

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

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

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

- **Version 3.2.0**

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

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

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

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

Support the IAU 1980 Nutation Angles of the JPL files.

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

Add the option `installnodoc` to the `make` command.

- **Version 3.1.0**

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

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

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

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

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

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

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

- **Version 3.0.0**

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

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

Add the function `calceph_gettimescale` and `calceph_gettimespan`.

Add the function `calceph_getpositionrecordcount` and `calceph_getpositionrecordindex`.

Add the function `calceph_getorientrecordcount` and `calceph_getorientrecordindex`.

Add the function `calceph_sgettimescale` and `calceph_sgettimespan`.

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

Use `sphinx-doc` to produce the documentation.

- **Version 2.3.2**

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

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

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

- **Version 2.3.1**

Fix the compilation warnings with the Pelles compiler.

Fix the compilation warnings with the C89 standard.

Fix the compilation warnings with the GNU C compilers.

Fix the documentation for the constant `CALCEPH_VERSION_STRING`.

- **Version 2.3.0**

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

Add the preprocessor macro `CALCEPH_VERSION_STRING`.

Add the function `calceph_getversion_str`.

Add the function `calceph_compute_order` and `calceph_orient_order`.

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

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

- **Version 2.2.5**

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

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

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

- **Version 2.2.4**

Update the version number of the dynamic library.

- **Version 2.2.3**

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

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

- **Version 2.2.2**

Support the compilation in the standard C89.

- **Version 2.2.1**

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

Support the Portland compilers.

Fix the info documentation.

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

- **Version 2.2.0**

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

Support the NAIF identification numbers.

Add the functions calceph_orient_unit and calceph_prefetch.

- **Version 2.1.0**

Fix a bug in calceph_getconstant and calceph_sgetconstant with an invalid name

Remove the null character in the name of the constant returned by the function (f90)calceph_(s)getconstantindex when the Fortran interface is used.

- **Version 2.0.0**

Fix memory leaks in calceph_open when errors occur.

Support INPOP file format 2.0 (supports TCB ephemeris file and add asteroids in the binary file).

Add the function calceph_open_array and calceph_compute_unit.

Add the tools calceph_inspector to show details about ephemeris file.

Support SPICE kernel file (SPK with segment 2 or 3, text and binary PCK, meta kernel, basic frame kernel).

Improve the performances.

Correct the Fortran 2003 interface for calceph_sgetconstantindex.

Add the constant 17 to get TCG-TCB from TCB ephemeris file.

- **Version 1.2.0**

Change the licensing : triple licenses to support integration in BSD software.

Remove explicit dependencies on the record size for DExxx.

- **Version 1.1.2**

Fix a compilation warning with oracle studio compiler 12.

Fix a bug with gcc on solaris in 64 bit mode.

Fix the copyright statements.

- **Version 1.1.1**

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

- **Version 1.1.0**

Add the function calceph_seterrorhandler for the custom error handlers.

- **Version 1.0.3**

Support the JPL ephemeris file DE423.

- **Version 1.0.2**

Fix memory leaks in the fortran-90 interface.

- **Version 1.0.1**

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

Fix the documentation of the function f90calceph_sopen.

Fix an invalid open mode on Windows operating systems.

Report accurately the I/O errors.

- **Version 1.0.0**

Initial release.

REPORTING BUGS

If you think you have found a bug in the CALCEPH Library, first have a look on the CALCEPH Library web page <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.

CALCEPH LIBRARY COPYING CONDITIONS

Copyright 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026

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

Contributed by

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

Fienga A. , Observatoire de la Côte d'Azur

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

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

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

The fact that you are presently reading this means that you have had knowledge of the CeCILL-C, CeCILL-B or CeCILL version 2.1 license and that you accept its terms.