
SpiceyPy Documentation

Release 3.0.0

Andrew Annex

Jan 10, 2020

CONTENTS

1	Installation	3
1.1	A simple example program	4
1.2	SpiceyPy Documentation	4
2	Common Issues	7
2.1	SSL Alert Handshake Issue	7
3	How to install from source (for bleeding edge updates)	9
4	Cassini Position Example	11
5	Cells Explained	15
6	Lessons	17
6.1	Basics of SpiceyPy	17
6.2	Remote Sensing Hands-On Lesson, using CASSINI (Python)	19
6.3	Geometric Event Finding Hands-On Lesson, using MEX (Python)	59
6.4	In-situ Sensing Hands-On Lesson, using CASSINI (Python)	80
6.5	Binary PCK Hands-On Lesson (Python)	102
6.6	Other Stuff (Python)	117
7	SpiceyPy package	153
7.1	spiceypy module	153
7.2	spiceypy.utils.support_types module	301
7.3	spiceypy.utils.libspice module	307
8	Indices and tables	309
	Python Module Index	311
	Index	313

This is the documentation for SpiceyPy. The documentation for each function in the wrapper is in large part copied from the “Abstract” and “Brief_I/O” sections of the corresponding CSPICE function documentation. Each wrapper function has a link back to the corresponding original CSPICE function documentation hosted at the NAIF website. For more in-depth information about SPICE, please visit the NAIF website or [click here](#) to view the entire CSPICE documentation.

The intent of the function doc-strings is to serve only as a quick reference to what the parameter’s expected types are for the purpose of getting started with the wrapper. As each function has a link to the CSPICE documentation for that function, more detailed explanations are deferred to the NAIF via those links.

Contents:

INSTALLATION

SpiceyPy is currently supported on Mac, Linux, FreeBSD, and Windows systems.

If you are new to python, it is a good idea to read a bit about it first docs.python-guide.org. For new installations of python, it is encouraged to install and or update: pip, setuptools, wheel, numpy, six, and certifi first before installing SpiceyPy

```
pip install -U pip setuptools wheel
pip install -U numpy six certifi
```

Then to install SpiceyPy, simply run:

```
pip install spiceypy
```

If you use anaconda/miniconda/conda run:

```
conda config --add channels conda-forge
conda install spiceypy
```

If no error was returned you have successfully installed SpiceyPy. To verify this you can list the installed packages via this pip command:

```
pip list
```

You should see spiceypy in the output of this command. Or you can start a python interpreter and try importing SpiceyPy like so:

```
import spiceypy

# print out the toolkit version installed
print(spiceypy.tkvrns('TOOLKIT'))
```

This should print out the toolkit version without any errors. You have now verified that SpiceyPy is installed.

1.1 A simple example program

This script calls the spiceypy function ‘tkvrsn’ and outputs the return value.

```
File tkvrsn.py

from __future__ import print_function
import spiceypy

def print_ver():
    """Prints the TOOLKIT version
    """
    print(spiceypy.tkvrsn('TOOLKIT'))

if __name__ == '__main__':
    print_ver()
```

From the command line, execute the function:

```
$ python tkvrsn.py
CSPICE_N0066
```

From Python, execute the function:

```
$ python
>>> import tkvrsn
>>> tkvrsn.print_ver()
CSPICE_N0066
```

1.2 SpiceyPy Documentation

The current version of SpiceyPy does not provide extensive documentation, but there are several ways to navigate your way through the Python version of the toolkit. One simple way is to use the standard Python mechanisms. All interfaces implemented in SpiceyPy can be listed using the standard built-in function `dir()`, which returns an alphabetized list of names comprising (among) other things, the API names. If you need to get additional information about an API parameters, the standard built-in function `help()` could be used:

```
>>> import spiceypy
>>> help(spiceypy.tkvrsn)
```

which produces

```
Help on function tkvrsn in module spiceypy.spiceypy:

tkvrsn(item)
    Given an item such as the Toolkit or an entry point name, return
    the latest version string.

    http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/tkvrsn_c.
    html

    :param item: Item for which a version string is desired.
    :type item: str
    :return: the latest version string.
    :rtype: str
```

As indicated in the help on the function, the complete documentation is available on the CSPICE toolkit version. Therefore it is recommended to have the CSPICE toolkit version installed locally in order to access its documentation offline.

COMMON ISSUES

2.1 SSL Alert Handshake Issue

Attention: As of 2020, users are not likely to experience this issue with python version 3.7 and above, and for newer 3.6.X releases. Users running older operating systems are encouraged to update to newer versions of python if they are attempting to install version 3.0.0 or above. See other sections of this document for more information.

In early 2017, JPL updated to a TLS1.2 certificate and enforced https connections causing installation issues for users, in particular for macOS users, with OpenSSL versions older than 1.0.1g. This is because older versions of OpenSSL still distributed in some environments which are incompatible with TLS1.2. As of late 2017 SpiceyPy has been updated with a strategy that can mitigate this issue on some systems, but it may not be totally reliable due to known deficiencies in setup tools and pip.

Another solution is to configure a new python installation that is linked against a newer version of OpenSSL, the easiest way to do this is to install python using homebrew, once this is done spiceypy can be installed to this new installation of python (IMHO this is the best option).

If your python 3.6 distribution was installed from the packages available at python.org an included command “Install Certificates.command” should be run before attempting to install SpiceyPy again. That command installs the certifi package that can also be install using pip.

Alternatively, installing an anaconda or miniconda python distribution and installing SpiceyPy using the conda command above is another possible work around.

Users continuing to have issues should report an issue to the github repository.

Supporting links:

<https://bugs.python.org/issue29065> <https://github.com/requests/requests/issues/2022> <https://pyfound.blogspot.com/2017/01/time-to-upgrade-your-python-tls-v12.html> <https://www.python.org/dev/peps/pep-0518> <https://github.com/AndrewAnnex/SpiceyPy/pull/202>

HOW TO INSTALL FROM SOURCE (FOR BLEEDING EDGE UPDATES)

Attention: If you have used the pip or conda install commands above you do not need to do any of the following commands. Installing from source is intended for advanced users. Users on machines running Windows should take note that attempting to install from source will require software such as visual studio and additional environment configuration. Given the complexity of this Windows users are highly encouraged to stick with the releases made available through PyPi/Anaconda Cloud.

If you wish to install from source, first simply clone the repository by running the following in your favorite shell:

```
git clone git@github.com:AndrewAnnex/SpiceyPy.git
```

If you do not have git, you can also directly download the source code from the GitHub repo for SpiceyPy at <https://github.com/AndrewAnnex/SpiceyPy>

To install the library, simply change into the root directory of the project and then run:

```
python setup.py install
```

The installation script will download the appropriate version of the SPICE toolkit for your system, and will build a shared library from the included static library files. Then the installation script will install SpiceyPy along with the generated shared library into your site-packages directory.

CASSINI POSITION EXAMPLE

Below is an example that uses spiceypy to plot the position of the Cassini spacecraft relative to the barycenter of Saturn.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

First import spiceypy and test it out.

```
import spiceypy as spice
```

```
# Print out the toolkit version
spice.tkvrns("TOOLKIT")
```

```
'CSPICE_N0066'
```

We will need to load some kernels. You will need to download the following kernels from the NAIF servers via the links provided. After the kernels have been downloaded to a common directory write a metakernel containing the file names for each downloaded kernel (provided after the links). I named the metakernel 'cassMetaK.txt' for this example. For more on defining meta kernels in spice, please consult the [Kernel Required Reading](#).

- naif0009.tls
- cas00084.tsc
- cpck05Mar2004.tpc
- cas_v37.tf
- 04135_04171pc_psisv2.bc
- 030201AP_SK_SM546_T45.bsp
- cas_iss_v09.ti
- 020514_SE_SAT105.bsp
- 981005_PLTEPH-DE405S.bsp

```
# The meta kernel file contains entries pointing to the following SPICE kernels,
↳ which the user needs to download.
# https://naif.jpl.nasa.gov/pub/naif/generic_kernels/lsk/a_old_versions/naif0009.tls
# https://naif.jpl.nasa.gov/pub/naif/CASSINI/kernels/sclk/cas00084.tsc
# https://naif.jpl.nasa.gov/pub/naif/CASSINI/kernels/pck/cpck05Mar2004.tpc
# https://naif.jpl.nasa.gov/pub/naif/CASSINI/kernels/fk/release.11/cas_v37.tf
# https://naif.jpl.nasa.gov/pub/naif/CASSINI/kernels/ck/04135_04171pc_psisv2.bc
```

(continues on next page)

(continued from previous page)

```
# https://naif.jpl.nasa.gov/pub/naif/CASSINI/kernels/spk/030201AP_SK_SM546_T45.bsp
# https://naif.jpl.nasa.gov/pub/naif/CASSINI/kernels/ik/release.11/cas_iss_v09.ti
# https://naif.jpl.nasa.gov/pub/naif/CASSINI/kernels/spk/020514_SE_SAT105.bsp
# https://naif.jpl.nasa.gov/pub/naif/CASSINI/kernels/spk/981005_PLTEPH-DE405S.bsp
#
# The following is the contents of a metakernel that was saved with
# the name 'cassMetaK.txt'.
# \begindata
# KERNELS_TO_LOAD=(
#   'naif0009.tls',
#   'cas00084.tsc',
#   'cpck05Mar2004.tpc',
#   '020514_SE_SAT105.bsp',
#   '981005_PLTEPH-DE405S.bsp',
#   '030201AP_SK_SM546_T45.bsp',
#   '04135_04171pc_psiv2.bc',
#   'cas_v37.tf',
#   'cas_iss_v09.ti')
# \begintext
#
spice.furnsh("./cassMetaK.txt")
```

```
step = 4000
# we are going to get positions between these two dates
utc = ['Jun 20, 2004', 'Dec 1, 2005']

# get et values one and two, we could vectorize str2et
etOne = spice.str2et(utc[0])
etTwo = spice.str2et(utc[1])
print("ET One: {}, ET Two: {}".format(etOne, etTwo))
```

```
ET One: 140961664.18440723, ET Two: 186667264.18308285
```

```
# get times
times = [x*(etTwo-etOne)/step + etOne for x in range(step)]

# check first few times:
print(times[0:3])
```

```
[140961664.18440723, 140973090.5844069, 140984516.98440656]
```

```
# check the documentation on spkpos before continuing
help(spice.spkpos)
```

```
Help on function spkpos in module spiceypy.spiceypy:

spkpos(targ: str, et: Union[float, numpy.ndarray], ref: str, abcorr: str, obs: str) ->
↳ Union[Tuple[numpy.ndarray, float], Tuple[numpy.ndarray, numpy.ndarray]]
    Return the position of a target body relative to an observing
    body, optionally corrected for light time (planetary aberration)
    and stellar aberration.

    http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkpos_c.html
```

(continues on next page)

(continued from previous page)

```
:param targ: Target body name.
:param et: Observer epoch.
:param ref: Reference frame of output position vector.
:param abcorr: Aberration correction flag.
:param obs: Observing body name.
: return:
    Position of target,
    One way light time between observer and target.
```

```
#Run spkpos as a vectorized function
positions, lightTimes = spice.spkpos('Cassini', times, 'J2000', 'NONE', 'SATURN_
↳BARYCENTER')

# Positions is a 3xN vector of XYZ positions
print("Positions: ")
print(positions[0])

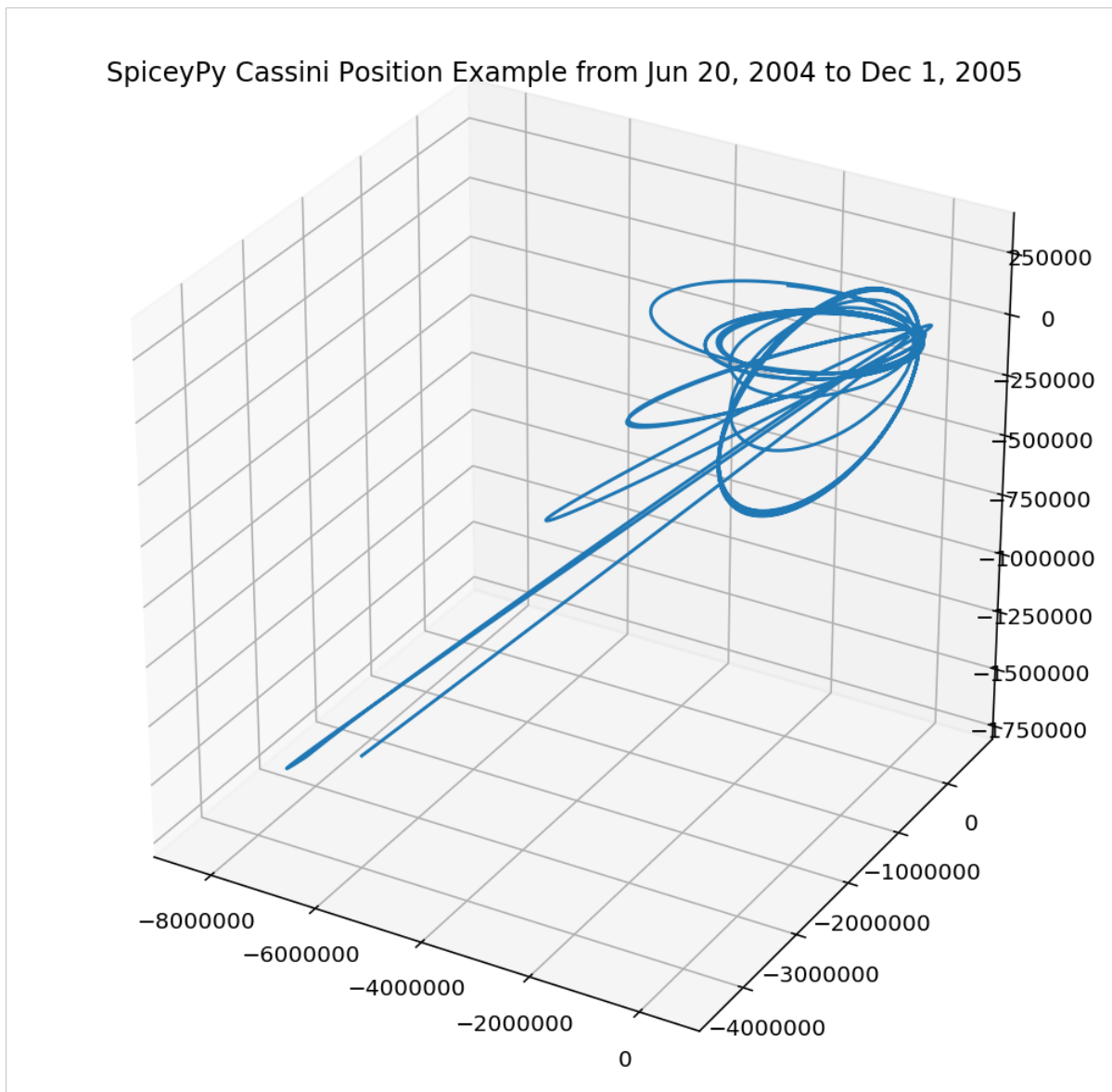
# Light times is a N vector of time
print("Light Times: ")
print(lightTimes[0])
```

```
Positions:
[-5461446.61080924 -4434793.40785864 -1200385.93315424]
Light Times:
23.8062238783
```

```
# Clean up the kernels
spice.kclear()
```

We will use matplotlib's 3D plotting to visualize Cassini's coordinates. We first convert the positions list to a 2D numpy array for easier indexing in the plot.

```
positions = positions.T # positions is shaped (4000, 3), let's transpose to (3, 4000)
↳for easier indexing
fig = plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111, projection='3d')
ax.plot(positions[0], positions[1], positions[2])
plt.title('SpiceyPy Cassini Position Example from Jun 20, 2004 to Dec 1, 2005')
plt.show()
```



CELLS EXPLAINED

Spice Cells are data structures included in SPICE and serve as the equivalents to lists and sets for CSPICE. For more primary documentation on cells, please see the [Cells required reading](#). For SpiceyPy, cells can be constructed in a variety of ways, shown below.

```
import spiceypy as spice

# create a spice bool cell using a function
bool_cell = spice.cell_bool(10)

# create a spice time cell using a function
time_cell = spice.cell_time(10)

# create a spice int cell using a function
int_cell = spice.cell_int(10)

# create a spice double cell using a function
double_cell = spice.cell_double(10)

# create a spice char cell using a function
char_cell = spice.cell_char(10, 10)
```

One can also use provided classes to provide easier type checking, in future versions SpiceyPy this may become default.

```
import spiceypy as spice

# create a spice bool cell using a function
bool_cell = spice.Cell_Bool(10)

# create a spice time cell using a function
time_cell = spice.Cell_Time(10)

# create a spice int cell using a function
int_cell = spice.Cell_Int(10)

# create a spice double cell using a function
double_cell = spice.Cell_Double(10)

# create a spice char cell using a function
char_cell = spice.Cell_Char(10, 10)
```


LESSONS

Here listed are the various SPICE lessons provided by the NAIF translated to use python code examples. Please refer to the NAIF lesson files for the kernel files needed to complete the exercises and to obtain the full content [naiflessons](#).

Contents:

6.1 Basics of SpiceyPy

6.1.1 Environment Set-up

Follow the installation instructions provided in installation.

6.1.2 Confirm SpiceyPy installation

There are multiple ways to verify your SpiceyPy installation. The first test is to simply run

```
pip list
```

You should see SpiceyPy in the list of your installed packages. If SpiceyPy is not present in the list then a configuration issue in your environment caused SpiceyPy to be installed in a non-standard way. Note this is an error prone to systems with multiple installed python versions.

If SpiceyPy is present in the pip list, then SpiceyPy is installed. Another verification step is within the python REPL run:

```
import spiceypy as spice
```

The version of the installed cspice toolkit (note: not SpiceyPy's version) should be printed out. Otherwise the Python interpreter should output an explanatory error message.

6.1.3 A simple example program

The following calls the SPICE function `spiceypy.spiceypy.tkvrns()` which outputs the version of cspice that SpiceyPy is wrapping.

```
import spiceypy as spice

spice.tkvrns('TOOLKIT')
```

This should output the following string:

```
'CSPICE_N0066'
```

6.1.4 Exceptions

SpiceyPy by default checks the spice error system for errors after all function calls and will raise `SpiceyErrors` (a python exception) when spice indicates an error. The exception message is a string that follows the format used elsewhere in spice and includes the toolkit version, the short description, explanation, long format description, and traceback (of spice calls). [Read the NAIF tutorial on exceptions here.](#)

Also, by default SpiceyPy captures the ‘found’ flags some functions return as it is not idiomatic to python and instead through a `SpiceyError` exception. This can be temporarily disabled using the `spiceypy.spiceypy.no_found_check()` context manager that allows the found flag to be returned to the user for action. Outside the context SpiceyPy functions will revert to default behavior.

```
import spiceypy as spice

spice.bodc2n(-9991) # will raise an exception

with spice.no_found_check():
    name, found = spice.bodc2n(-9991) # found is now available, no exception raised!
    assert not found # found is going to be False in this case.

spice.bodc2n(-9991) # will raise an exception again
```

There is also an accompanying context manager for enabling the default spiceypy behavior within a code block like so:

```
import spiceypy as spice

spice.bodc2n(-9991) # will raise an exception

with spice.found_check():
    name = spice.bodc2n(-9991) # will also raise an exception
```

In addition, for advanced users there are two function `spiceypy.spiceypy.found_check_off()` and `spiceypy.spiceypy.found_check_on()` which will disable and enable the behavior without use of the context manager. Additionally, a method `spiceypy.spiceypy.get_found_catch_state()` allows users to query the current state of found flag catching setting.

6.2 Remote Sensing Hands-On Lesson, using CASSINI (Python)

November 20, 2017

6.2.1 Overview

In this lesson you will develop a series of simple programs that demonstrate the usage of SpiceyPy to compute a variety of different geometric quantities applicable to experiments carried out by a remote sensing instrument flown on an interplanetary spacecraft. This particular lesson focuses on a framing camera flying on the Cassini spacecraft, but many of the concepts are easily extended and generalized to other scenarios.

6.2.2 References

This section lists SPICE documents referred to in this lesson.

In some cases the lesson explanations also refer to the information provided in the meta-data area of the kernels used in the lesson examples. It is especially true in case of the FK and IK files, which often contain comprehensive descriptions of the frames, instrument FOVs, etc. Since both the FK and IK are text kernels, the information provided in them can be viewed using any text editor, while the meta information provided in binary kernels—SPKs and CKs—can be viewed using “commnt” or “spacit” utility programs located in “cspice/exe” of Toolkit installation tree.

Tutorials

The following SPICE tutorials serve as references for the discussions in this lesson:

Name	Lesson steps/functions it describes
Time	Time Conversion
SCLK and LSK	Time Conversion
SPK	Obtaining Ephemeris Data
Frames	Reference Frames
Using Frames	Reference Frames
PCK	Planetary Constants Data
CK	Spacecraft Orientation Data
DSK	Detailed Target Shape (Topography) Data

These tutorials are available from the NAIF ftp server at JPL:

<http://naif.jpl.nasa.gov/naif/tutorials.html>

Required Readings

The Required Reading documents are provided with the Toolkit and are located under the “cspice/doc” directory in the CSPICE Toolkit installation tree.

Name	Lesson steps/functions that it describes
ck.req	Obtaining spacecraft orientation data
dsk.req	Obtaining detailed body shape data
frames.req	Using reference frames
naif_ids.req	Determining body ID codes
pck.req	Obtaining planetary constants data
sclk.req	SCLK time conversion
spk.req	Obtaining ephemeris Data
time.req	Time conversion

The Permuted Index

Another useful document distributed with the Toolkit is the permuted index. This is located under the “cspice/doc” directory in the C installation tree.

This text document provides a simple mechanism by which users can discover which SpiceyPy functions perform functions of interest, as well as the names of the source files that contain these functions.

SpiceyPy API Documentation

A SpiceyPy function’s parameters specification is available using the built-in Python help system.

For example, the Python help function

```
>>> import spiceypy
>>> help(spiceypy.str2et)
```

describes of the str2et function’s parameters, while the document

```
https://naif.jpl.nasa.gov/pub/naif/toolkit\_docs/C/cspice/str2et\_c.html
```

describes extensively the str2et functionality.

6.2.3 Kernels Used

The following kernels are used in examples provided in this lesson:

#	FILE NAME	TYPE	DESCRIPTION
1	naif0008.tls	LSK	Generic LSK
2	cas00084.tsc	SCLK	Cassini SCLK
3	981005_PLTEPH-DE405S.bsp	SPK	Solar System Ephemeris
4	020514_SE_SAT105.bsp	SPK	Saturnian Satellite Ephemeris
5	030201AP_SK_SM546_T45.bsp	SPK	Cassini Spacecraft SPK
6	cas_v37.tf	FK	Cassini FK
7	04135_04171pc_psiv2.bc	CK	Cassini Spacecraft CK
8	cpck05Mar2004.tpc	PCK	Cassini Project PCK
9	phoebe_64q.bds	DSK	Phoebe DSK
10	cas_iss_v09.ti	IK	ISS Instrument Kernel

These SPICE kernels are included in the lesson package available from the NAIF server at JPL:

```
ftp://naif.jpl.nasa.gov/pub/naif/toolkit\_docs/Lessons/
```

In addition to these kernels, the extra credit exercises require the following kernels:

#	FILE NAME	TYPE	DESCRIPTION
11	jup310_2004.bsp	SPK	Generic Jovian Satellite Ephemeris

These SPICE kernels are available from the NAIF server at JPL:

```
https://naif.jpl.nasa.gov/pub/naif/generic\_kernels/spk/satellites/
```

6.2.4 SpiceyPy Modules Used

This section provides a complete list of the functions and kernels that are suggested for usage in each of the exercises in this lesson. (You may wish to not look at this list unless/until you “get stuck” while working on your own.)

CHAPTER	EXERCISE	FUNCTIONS	NON-VOID	KERNELS
1	convtm	spiceypy.furnsh spiceypy.unload	spiceypy.str2et spiceypy.etcal spiceypy.timeout spiceypy.sce2s	1,2
	extra (*)		spiceypy.unitim spiceypy.sct2e spiceypy.et2utc spiceypy.scs2e	1,2
2	getsta	spiceypy.furnsh spiceypy.unload	spiceypy.str2et spiceypy.spkezr spiceypy.spkpos spiceypy.vnorm spiceypy.convrt	1,3-5
	extra (*)	spiceypy.kclear		1,3-5,11
3	xform	spiceypy.furnsh spiceypy.unload	spiceypy.str2et spiceypy.spkezr spiceypy.sxform spiceypy.mxvg spiceypy.spkpos spiceypy.pxform spiceypy.mxv spiceypy.convrt spiceypy.vsep	1-8
	extra (*)	spiceypy.kclear		1-8
4	subpts	spiceypy.furnsh spiceypy.unload	spiceypy.str2et spiceypy.subpnt spiceypy.vnorm spiceypy.subslr	1,3-5,8,9
	extra (*)	spiceypy.kclear	spiceypy.reclat spiceypy.dpr spiceypy.bodvrd spiceypy.recpgr	1,3-5,8
5	fovint	spiceypy.furnsh spiceypy.unload	spiceypy.str2et spiceypy.bodn2c spiceypy.getfov spiceypy.sincpt spiceypy.reclat spiceypy.dpr spiceypy.illumf spiceypy.et2lst	1-10
(*) Additional APIs and kernels used in Extra Credit tasks.				

Use the Python built-in help system on the various functions listed above for the API parameters' description, and refer to the headers of their corresponding CSPICE versions for detailed interface specifications.

6.2.5 Time Conversion (convtm)

6.2.6 Task Statement

Write a program that prompts the user for an input UTC time string, converts it to the following time systems and output formats:

1. Ephemeris Time (ET) **in** seconds past J2000
2. Calendar Ephemeris Time
3. Spacecraft Clock Time

and displays the results. Use the program to convert “2004 jun 11 19:32:00” UTC into these alternate systems.

6.2.7 Learning Goals

Familiarity with the various time conversion and parsing functions available in the Toolkit. Exposure to source code headers and their usage in learning to call functions.

6.2.8 Approach

The solution to the problem can be broken down into a series of simple steps:

```
--  Decide which SPICE kernels are necessary. Prepare a meta-kernel
--  listing the kernels and load it into the program.

--  Prompt the user for an input UTC time string.

--  Convert the input time string into ephemeris time expressed as
--  seconds past J2000 TDB. Display the result.

--  Convert ephemeris time into a calendar format. Display the
--  result.

--  Convert ephemeris time into a spacecraft clock string. Display
--  the result.
```

You may find it useful to consult the permuted index, the headers of various source modules, and the “Time Required Reading” (time.req) and” SCLK Required Reading” (sclk.req) documents.

When completing the “calendar format” step above, consider using one of two possible methods: `spiceypy.etcadl` or `spiceypy.timeout`.

6.2.9 Solution

Solution Meta-Kernel

The meta-kernel we created for the solution to this exercise is named 'convtm.tm'. Its contents follow:

```
KPL/MK

This is the meta-kernel used in the solution of the "Time
Conversion" task in the Remote Sensing Hands On Lesson.

The names and contents of the kernels referenced by this
meta-kernel are as follows:

File name                Contents
-----
naif0008.tls             Generic LSK
cas00084.tsc             Cassini SCLK

\begindata
KERNELS_TO_LOAD = ( 'kernels/lsk/naif0008.tls',
                    'kernels/sclk/cas00084.tsc' )
\beginext
```

Solution Source Code

A sample solution to the problem follows:

```
#
# Solution convtm
#
from __future__ import print_function
from builtins import input

import spiceypy

def convtm():
    #
    # Local Parameters
    #
    METAKR = 'convtm.tm'
    SCLKID = -82

    spiceypy.furnsh( METAKR )

    #
    # Prompt the user for the input time string.
    #
    utctim = input( 'Input UTC Time: ' )

    print( 'Converting UTC Time: {:s}'.format( utctim ) )

    #
    # Convert utctim to ET.
    #
    et = spiceypy.str2et( utctim )
```

(continues on next page)

(continued from previous page)

```

print( '    ET Seconds Past J2000: {:.16.3f}'.format( et ) )

#
# Now convert ET to a calendar time string.
# This can be accomplished in two ways.
#
calet = spiceypy.etcal( et )

print( '    Calendar ET (etcal):   {:s}'.format( calet ) )

#
# Or use timeout for finer control over the
# output format. The picture below was built
# by examining the header of timeout.
#
calet = spiceypy.timeout( et, 'YYYY-MON-DDTHR:MN:SC ::TDB' )

print( '    Calendar ET (timeout):  {:s}'.format( calet ) )

#
# Convert ET to spacecraft clock time.
#
sclkst = spiceypy.sce2s( SCLKID, et )

print( '    Spacecraft Clock Time:  {:s}'.format( sclkst ) )

spiceypy.unload( METAKR )

if __name__ == '__main__':
    convtm()

```

Solution Sample Output

Execute the program:

```

Input UTC Time: 2004 jun 11 19:32:00
Converting UTC Time: 2004 jun 11 19:32:00
    ET Seconds Past J2000:    140254384.185
    Calendar ET (etcal):    2004 JUN 11 19:33:04.184
    Calendar ET (timeout):  2004-JUN-11T19:33:04
    Spacecraft Clock Time:  1/1465674964.105

```

6.2.10 Extra Credit

In this “extra credit” section you will be presented with more complex tasks, aimed at improving your understanding of time conversions, the Toolkit routines that deal with them, and some common errors that may happen during the execution of these conversions.

These “extra credit” tasks are provided as task statements, and unlike the regular tasks, no approach or solution source code is provided. In the next section, you will find the numeric solutions (when applicable) and answers to the questions asked in these tasks.

Task statements and questions

1. Extend your program to convert the input UTC time string to TDB Julian Date. Convert "2004 jun 11 19:32:00" UTC.

(continues on next page)

(continued from previous page)

2. Remove the LSK from the original meta-kernel and run your program again, using the same inputs as before. Has anything changed? Why?
3. Remove the SCLK from the original meta-kernel and run your program again, using the same inputs as before. Has anything changed? Why?
4. Modify your program to perform conversion of UTC or ephemeris time, to a spacecraft clock string using the NAIF ID for the CASSINI ISS NAC camera. Convert "2004 jun 11 19:32:00" UTC.
5. Find the earliest UTC time that can be converted to CASSINI spacecraft clock.
6. Extend your program to convert the spacecraft clock time obtained in the regular task back to UTC Time and present it in ISO calendar date format, with a resolution of milliseconds.
7. Examine the contents of the generic LSK and the CASSINI SCLK kernels. Can you understand and explain what you see?

Solutions and answers

1. Two methods exist in order to convert ephemeris time to Julian Date: `spiceypy.unitim` and `spiceypy.timeout`. The difference between them is the type of output produced by each method. `spiceypy.unitim` returns the double precision value of an input epoch, while `spiceypy.timeout` returns the string representation of the ephemeris time in Julian Date format (when picture input is set to 'JULIAND.##### :TDB'). Refer to the function header for further details. The solution for the requested input UTC string is:

 Julian Date TDB: 2453168.3146318
2. When running the original program without the LSK kernel, an error is produced:

 Traceback (most recent call last):
 File "convtm.py", line 67, in <module>
 convtm()
 File "convtm.py", line 30, in convtm
 et = spiceypy.str2et(utctim)
 File "/home/bsemenov/local/lib/python3.5/site-packages/spiceypy/spiceypy.py", line 76, in with_errcheck
 check_for_spice_error(f)
 File "/home/bsemenov/local/lib/python3.5/site-packages/spiceypy/spiceypy.py", line 59, in check_for_spice_error
 raise stypes.SpiceyError(msg)
 spiceypy.utils.support_types.SpiceyError:
 =====
 =====

 Toolkit version: N0066

(continues on next page)

(continued from previous page)

```
SPICE(NOLEAPSECONDS) --
```

The variable that points to the leapseconds (DELTET/DELTA_AT) could not be located in the kernel pool. It is likely that the leapseconds kernel has not been loaded via the routine FURNISH.

```
str2et_c --> STR2ET --> TTRANS
```

```
=====
=====
```

This error is triggered by spiceypy.str2et because the variable that points to the leapseconds is not present in the kernel pool and therefore the program lacks data required to perform the requested UTC to ephemeris time conversion.

By default, SPICE will report, as a minimum, a short descriptive message and an expanded form of this short message where more details about the error are provided. If this error message is not sufficient for you to understand what has happened, you could go to the "Exceptions" section in the SPICELIB or CSPICE headers of the function that has triggered the error and find out more information about the possible causes.

3. When running the original program without the SCLK kernel, an error is produced:

Traceback (most recent call last):

```
File "convtm.py", line 67, in <module>
    convtm()
File "convtm.py", line 58, in convtm
    sclkst = spiceypy.sce2s( SCLKID, et )
File "/home/bsemenov/local/lib/python3.5/site-packages/spiceypy/spiceypy.py", line 76, in with_errcheck
    check_for_spice_error(f)
File "/home/bsemenov/local/lib/python3.5/site-packages/spiceypy/spiceypy.py", line 59, in check_for_spice_error
    raise stypes.SpiceyError(msg)
spiceypy.utils.support_types.SpiceyError:
=====
=====
```

Toolkit version: N0066

```
SPICE(KERNELVARNOTFOUND) --
```

The Variable Was not Found in the Kernel Pool.

SCLK_DATA_TYPE_82 not found. Did you load the SCLK kernel?

```
sce2s_c --> SCE2S --> SCE2T --> SCTYPE --> SCLI01
```

```
=====
=====
```

This error is triggered by spiceypy.sce2s. In this case the error message may not give you enough information to understand what has actually happened. Nevertheless, the expanded form of

(continues on next page)

(continued from previous page)

this short message clearly indicates that the SCLK kernel for the spacecraft ID -82 has not been loaded.

The UTC string to ephemeris time conversion and the conversion of ephemeris time into a calendar format worked normally as these conversions only require the LSK kernel to be loaded.

4. The first thing you need to do is to find out what the NAIF ID is for the CASSINI ISS NAC camera. In order to do so, examine the ISS instrument kernel listed above and look for the "NAIF ID Code to Name Mapping" and there, for the NAIF ID given to CASSINI_ISS_NAC (which is -82360). Then replace in your code the SCLK ID -82 with -82360. After executing the program using the original meta-kernel, you will be getting the same error as in the previous task. Despite the error being exactly the same, this case is different. Generally, spacecraft clocks are associated with the spacecraft ID and not with its payload, sensors or structures IDs. Therefore, in order to do conversions from/to spacecraft clock for payload, sensors or spacecraft structures, the spacecraft ID must be used.

Note that this does not need to be true for all missions or payloads, as SPICE does not restrict the SCLKs to spacecraft IDs only. Please refer to your mission's SCLK kernels for particulars.

5. Use `spiceypy.sct2e` with the encoding of the Cassini spacecraft clock time set to 0.0 ticks and convert the resulting ephemeris time to UTC using either `spiceypy.timeout` or `spiceypy.et2utc`. The solution for the requested SCLK string is:

```
Earliest UTC convertible to SCLK: 1980-01-01T00:00:00.000
```

6. Use `spiceypy.scs2e` with the SCLK string obtained in the computations performed in the regular tasks and convert the resulting ephemeris time to UTC using either `spiceypy.et2utc`, with 'ISOC' format and 3 digits precision, or using `spiceypy.timeout` using the time picture 'YYYY-MM-DDTHR:MN:SC.###::RND'. The solution of the requested conversion is:

```
Spacecraft Clock Time:      1/1465674964.105
UTC time from spacecraft clock: 2004-06-11T19:31:59.999
```

6.2.11 Obtaining Target States and Positions (getsta)

6.2.12 Task Statement

Write a program that prompts the user for an input UTC time string, computes the following quantities at that epoch:

1. The apparent state of Phoebe **as** seen **from** **CASSINI** **in** the J2000 frame, **in** kilometers **and** kilometers/second. This vector itself **is not** of **any** particular interest, but it **is** a useful intermediate quantity **in** some geometry calculations.
2. The apparent position of the Earth **as** seen **from** **CASSINI** **in** the

(continues on next page)

(continued from previous page)

- ```
J2000 frame, in kilometers.
```
3. The one-way light time between CASSINI and the apparent position of Earth, in seconds.
  4. The apparent position of the Sun as seen from Phoebe in the J2000 frame (J2000), in kilometers.
  5. The actual (geometric) distance between the Sun and Phoebe, in astronomical units.

and displays the results. Use the program to compute these quantities at “2004 jun 11 19:32:00” UTC.

### 6.2.13 Learning Goals

Understand the anatomy of an `spiceypy.spkezr` call. Discover the difference between `spiceypy.spkezr` and `spiceypy.spkpos`. Familiarity with the Toolkit utility “brief”. Exposure to unit conversion with SpiceyPy.

### 6.2.14 Approach

The solution to the problem can be broken down into a series of simple steps:

- ```
-- Decide which SPICE kernels are necessary. Prepare a meta-kernel
-- listing the kernels and load it into the program.

-- Prompt the user for an input time string.

-- Convert the input time string into ephemeris time expressed as
-- seconds past J2000 TDB.

-- Compute the state of Phoebe relative to CASSINI in the J2000
-- reference frame, corrected for aberrations.

-- Compute the position of Earth relative to CASSINI in the J2000
-- reference frame, corrected for aberrations. (The function in
-- the library that computes this also returns the one-way light
-- time between CASSINI and Earth.)

-- Compute the position of the Sun relative to Phoebe in the J2000
-- reference frame, corrected for aberrations.

-- Compute the position of the Sun relative to Phoebe without
-- correcting for aberration.

-- Compute the length of this vector. This provides the desired
-- distance in kilometers.

-- Convert the distance in kilometers into AU.
```

You may find it useful to consult the permuted index, the headers of various source modules, and the “SPK Required Reading” (`spk.req`) document.

When deciding which SPK files to load, the Toolkit utility “brief” may be of some use.

“brief” is located in the “cspice/exe” directory for C toolkits. Consult its user’s guide available in “cspice/doc/brief.ug” for details.

6.2.15 Solution

Solution Meta-Kernel

The meta-kernel we created for the solution to this exercise is named ‘getsta.tm’. Its contents follow:

KPL/MK

This **is** the meta-kernel used **in** the solution of the "Obtaining Target States and Positions" task **in** the Remote Sensing Hands On Lesson.

The names **and** contents of the kernels referenced by this meta-kernel are **as** follows:

File name	Contents
naif0008.tls	Generic LSK
981005_PLTEPH-DE405S.bsp	Solar System Ephemeris
020514_SE_SAT105.bsp	Saturnian Satellite Ephemeris
030201AP_SK_SM546_T45.bsp	Cassini Spacecraft SPK

```
\begindata
KERNELS_TO_LOAD = ( 'kernels/lsk/naif0008.tls',
                     'kernels/spk/981005_PLTEPH-DE405S.bsp',
                     'kernels/spk/020514_SE_SAT105.bsp',
                     'kernels/spk/030201AP_SK_SM546_T45.bsp' )
\beginntext
```

Solution Source Code

A sample solution to the problem follows:

```
#
# Solution getsta.py
#
from __future__ import print_function
from builtins import input

import spiceypy

def getsta():
    #
    # Local parameters
    #
    METAKR = 'getsta.tm'

    #
    # Load the kernels that this program requires. We
    # will need a leapseconds kernel to convert input
    # UTC time strings into ET. We also will need the
    # necessary SPK files with coverage for the bodies
    # in which we are interested.
```

(continues on next page)

(continued from previous page)

```

#
spiceypy.furnsh( METAKR )

#
#Prompt the user for the input time string.
#
utctim = input( 'Input UTC Time: ' )

print( 'Converting UTC Time: {:s}'.format(utctim) )

#
#Convert utctim to ET.
#
et = spiceypy.str2et( utctim )

print( '    ET seconds past J2000: {:16.3f}'.format(et) )

#
# Compute the apparent state of Phoebe as seen from
# CASSINI in the J2000 frame. All of the ephemeris
# readers return states in units of kilometers and
# kilometers per second.
#
[state, ltime] = spiceypy.spkezr( 'PHOEBE', et,          'J2000',
                                'LT+S',    'CASSINI'    )

print( '    Apparent state of Phoebe as seen '
      'from CASSINI in the J2000\n'
      '    frame (km, km/s):'
      )

print( '        X = {:16.3f}'.format(state[0])      )
print( '        Y = {:16.3f}'.format(state[1])      )
print( '        Z = {:16.3f}'.format(state[2])      )
print( '        VX = {:16.3f}'.format(state[3])      )
print( '        VY = {:16.3f}'.format(state[4])      )
print( '        VZ = {:16.3f}'.format(state[5])      )

#
# Compute the apparent position of Earth as seen from
# CASSINI in the J2000 frame. Note: We could have
# continued using spkezr and simply ignored the
# velocity components.
#
[pos, ltime] = spiceypy.spkpos( 'EARTH', et,          'J2000',
                              'LT+S',    'CASSINI',    )

print( '    Apparent position of Earth as '
      'seen from CASSINI in the J2000\n'
      '    frame (km):'
      )
print( '        X = {:16.3f}'.format(pos[0])      )
print( '        Y = {:16.3f}'.format(pos[1])      )
print( '        Z = {:16.3f}'.format(pos[2])      )

#
# We need only display LTIME, as it is precisely the
# light time in which we are interested.
#

```

(continues on next page)

(continued from previous page)

```

print( '    One way light time between CASSINI and '
      'the apparent position\n'
      '    of Earth (seconds):'
      '    {:16.3f}'.format(ltime) )

#
# Compute the apparent position of the Sun as seen from
# PHOEBE in the J2000 frame.
#
[pos, ltime] = spiceypy.spkpos( 'SUN', et, 'J2000',
                              'LT+S', 'PHOEBE', )

print( '    Apparent position of Sun as '
      'seen from Phoebe in the\n'
      '    J2000 frame (km):'
      '    ' )
print( '    X = {:16.3f}'.format(pos[0]) )
print( '    Y = {:16.3f}'.format(pos[1]) )
print( '    Z = {:16.3f}'.format(pos[2]) )

#
# Now we need to compute the actual distance between
# the Sun and Phoebe. The above spkpos call gives us
# the apparent distance, so we need to adjust our
# aberration correction appropriately.
#
[pos, ltime] = spiceypy.spkpos( 'SUN', et, 'J2000',
                              'NONE', 'PHOEBE' )

#
# Compute the distance between the body centers in
# kilometers.
#
dist = spiceypy.vnorm( pos )

#
# Convert this value to AU using convrt.
#
dist = spiceypy.convrt( dist, 'KM', 'AU' )

print( '    Actual distance between Sun and '
      'Phoebe body centers:\n'
      '    (AU): {:16.3f}'.format(dist) )

spiceypy.unload( METAKR )

if __name__ == '__main__':
    getsta()
    
```

Solution Sample Output

Execute the program:

```

Input UTC Time: 2004 jun 11 19:32:00
Converting UTC Time: 2004 jun 11 19:32:00
ET seconds past J2000: 140254384.185
Apparent state of Phoebe as seen from CASSINI in the J2000
frame (km, km/s):
    
```

(continues on next page)

(continued from previous page)

```

X =      -119.921
Y =      2194.139
Z =      -57.639
VX =      -5.980
VY =      -2.119
VZ =      -0.295
Apparent position of Earth as seen from CASSINI in the J2000
frame (km):
X =      353019393.123
Y =     -1328180352.140
Z =     -568134171.697
One way light time between CASSINI and the apparent position
of Earth (seconds):      4960.427
Apparent position of Sun as seen from Phoebe in the
J2000 frame (km):
X =      376551465.272
Y =     -1190495630.303
Z =     -508438699.110
Actual distance between Sun and Phoebe body centers:
(AU):      9.012

```

6.2.16 Extra Credit

In this “extra credit” section you will be presented with more complex tasks, aimed at improving your understanding of state computations, particularly the application of the different light time and stellar aberration corrections available in the `spiceypy.spkezr` function, and some common errors that may happen when computing these states.

These “extra credit” tasks are provided as task statements, and unlike the regular tasks, no approach or solution source code is provided. In the next section, you will find the numeric solutions (when applicable) and answers to the questions asked in these tasks.

Task statements and questions

1. Remove the Solar System ephemerides SPK from the original meta-kernel and run your program again, using the same inputs as before. Has anything changed? Why?
2. Extend your program to compute the geometric position of Jupiter as seen from Saturn in the J2000 frame (J2000), in kilometers.
3. Extend, or modify, your program to compute the position of the Sun as seen from Saturn in the J2000 frame (J2000), in kilometers, using the following light time and aberration corrections: NONE, LT and LT+S. Explain the differences.
4. Examine the CASSINI frames definition kernel and the ISS instrument kernel to find the SPICE ID/name definitions.

Solutions and answers

1. When running the original program without the Solar System ephemerides SPK, an error is produced by `spiceypy.spkezr`:

Traceback (most recent call last):

(continues on next page)

(continued from previous page)

```

File "getsta.py", line 128, in <module>
    getsta()
File "getsta.py", line 47, in getsta
    'LT+S', 'CASSINI' )
File "/home/bsemenov/local/lib/python3.5/site-packages/spiceypy/spi
ceypy.py", line 76, in with_errcheck
    check_for_spice_error(f)
File "/home/bsemenov/local/lib/python3.5/site-packages/spiceypy/spi
ceypy.py", line 59, in check_for_spice_error
    raise stypes.SpiceyError(msg)
spiceypy.utils.support_types.SpiceyError:
=====
=====

Toolkit version: N0066

SPICE(SPKINSUFFDATA) --

Insufficient ephemeris data has been loaded to compute the state of -
82 (CASSINI) relative to 0 (SOLAR SYSTEM BARYCENTER) at the ephemeris
epoch 2004 JUN 11 19:33:04.184.

spkezr_c --> SPKEZR --> SPKEZ --> SPKACS --> SPKGEO

=====
=====

This error is generated when trying to compute the apparent
state of Phoebe as seen from CASSINI in the J2000 frame because
despite both Phoebe and CASSINI ephemeris data being relative
to the Saturn Barycenter, the state of the spacecraft with
respect to the solar system barycenter is required to compute
the light time and stellar aberrations. The loaded SPK data are
enough to compute geometric states of CASSINI with respect to
the Saturn Barycenter, and geometric states of Phoebe with
respect to the Saturn Barycenter, but insufficient to compute
the state of the spacecraft relative to the Solar System
Barycenter because the SPK data needed to compute geometric
states of Saturn Barycenter relative to the Solar System
barycenter are no longer loaded. Run "brief" on the SPKs used
in the original task to find out which ephemeris objects are
available from those kernels. If you want to find out what is
the 'center of motion' for the ephemeris object(s) included in
an SPK, use the -c option when running "brief":

BRIEF -- Version 4.0.0, September 8, 2010 -- Toolkit Version N0066

Summary for: kernels/spk/981005_PLTEPH-DE405S.bsp

Bodies: MERCURY BARYCENTER (1) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        VENUS BARYCENTER (2) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        EARTH BARYCENTER (3) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        MARS BARYCENTER (4) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        JUPITER BARYCENTER (5) w.r.t. SOLAR SYSTEM BARYCENTER (0)
        SATURN BARYCENTER (6) w.r.t. SOLAR SYSTEM BARYCENTER (0)

```

(continues on next page)

(continued from previous page)

```

URANUS BARYCENTER (7) w.r.t. SOLAR SYSTEM BARYCENTER (0)
NEPTUNE BARYCENTER (8) w.r.t. SOLAR SYSTEM BARYCENTER (0)
PLUTO BARYCENTER (9) w.r.t. SOLAR SYSTEM BARYCENTER (0)
SUN (10) w.r.t. SOLAR SYSTEM BARYCENTER (0)
MERCURY (199) w.r.t. MERCURY BARYCENTER (1)
VENUS (299) w.r.t. VENUS BARYCENTER (2)
MOON (301) w.r.t. EARTH BARYCENTER (3)
EARTH (399) w.r.t. EARTH BARYCENTER (3)
MARS (499) w.r.t. MARS BARYCENTER (4)
Start of Interval (UTC)          End of Interval (UTC)
-----
-----
----
2004-JUN-11 05:00:00.000          2004-JUN-12 12:00:00.000

Summary for: kernels/spk/020514_SE_SAT105.bsp

Bodies: MIMAS (601) w.r.t. SATURN BARYCENTER (6)
ENCCELADUS (602) w.r.t. SATURN BARYCENTER (6)
TETHYS (603) w.r.t. SATURN BARYCENTER (6)
DIONE (604) w.r.t. SATURN BARYCENTER (6)
RHEA (605) w.r.t. SATURN BARYCENTER (6)
TITAN (606) w.r.t. SATURN BARYCENTER (6)
HYPERION (607) w.r.t. SATURN BARYCENTER (6)
IAPETUS (608) w.r.t. SATURN BARYCENTER (6)
PHOEBE (609) w.r.t. SATURN BARYCENTER (6)
SATURN (699) w.r.t. SATURN BARYCENTER (6)
Start of Interval (UTC)          End of Interval (UTC)
-----
-----
----
2004-JUN-11 05:00:00.000          2004-JUN-12 12:00:00.000

Summary for: kernels/spk/030201AP_SK_SM546_T45.bsp

Body: CASSINI (-82) w.r.t. SATURN BARYCENTER (6)
Start of Interval (UTC)          End of Interval (UTC)
-----
-----
--
2004-JUN-11 05:00:00.000          2004-JUN-12 12:00:00.000

2.  If you run your extended program with the original meta-kernel,
the SPICE(SPKINSUFFDATA) error should be produced by the
spiceypy.spkpos function because you have not loaded enough
ephemeris data to compute the position of Jupiter with respect
to Saturn. The loaded SPKs contain data for Saturn relative to
the Solar System Barycenter, and for the Jupiter System
Barycenter relative to the Solar System Barycenter, but the
data for Jupiter relative to the Jupiter System Barycenter are
missing:

Additional kernels required for this task:

File name          Contents

```

(continues on next page)

(continued from previous page)

```
-----
jup310_2004.bsp          Generic Jovian Satellite Ephemeris
```

available in the NAIF server at:

https://naif.jpl.nasa.gov/pub/naif/generic_kernels/spk/satellites/

Download the relevant SPK, add it to the meta-kernel and run again your extended program. The solution for the input UTC time "2004 jun 11 19:32:00" when using the downloaded Jovian Satellite Ephemeris SPK:

Actual position of Jupiter as seen from Saturn in the
J2000 frame (km):
X = -436016583.291
Y = -1094176737.323
Z = -446585337.431

3. When using 'NONE' aberration corrections, `spiceypy.spkpos` returns the geometric position of the target body relative to the observer. If 'LT' is used, the returned vector corresponds to the position of the target at the moment it emitted photons arriving at the observer at 'et'. If 'LT+S' is used instead, the returned vector takes into account the observer's velocity relative to the solar system barycenter. The solution for the input UTC time "2004 jun 11 19:32:00" is:

Actual (geometric) position of Sun as seen from Saturn in the
J2000 frame (km):
X = 367770592.367
Y = -1197330367.359
Z = -510369088.677

Light-time corrected position of Sun as seen from Saturn in the
J2000 frame (km):
X = 367770572.921
Y = -1197330417.733
Z = -510369109.509

Apparent position of Sun as seen from Saturn in the
J2000 frame (km):
X = 367726456.168
Y = -1197342627.879
Z = -510372252.747

6.2.17 Spacecraft Orientation and Reference Frames (xform)

6.2.18 Task Statement

Write a program that prompts the user for an input time string, computes and displays the following at the epoch of interest:

1. The apparent state of Phoebe **as** seen **from CASSINI** **in** the IAU_PHOEBE body-fixed frame. This vector itself **is not** of any particular interest, but it **is** a useful intermediate quantity **in** some geometry calculations.
 2. The angular separation between the apparent position of Earth **as** seen **from CASSINI** **and** the nominal boresight of the CASSINI high gain antenna (HGA).
- The HGA boresight direction **is** provided by the kernel variable TKFRAME_-82101_BORESIGHT, which **is** defined **in** the Cassini frame kernel cited above **in** the section "Kernels Used." In this kernel, the HGA boresight vector **is** expressed relative to the CASSINI_HGA reference frame.

Use the program to compute these quantities at the epoch “2004 jun 11 19:32:00” UTC.

6.2.19 Learning Goals

Familiarity with the different types of kernels involved in chaining reference frames together, both inertial and non-inertial. Discover some of the matrix and vector math functions. Understand the difference between spiceypy.pxform and spiceypy.sxform.

6.2.20 Approach

The solution to the problem can be broken down into a series of simple steps:

- Decide which SPICE kernels are necessary. Prepare a meta-kernel listing the kernels **and** load it into the program.
- Prompt the user **for** an **input** time string.
- Convert the **input** time string into ephemeris time expressed **as** seconds past J2000 TDB.
- Compute the state of Phoebe relative to CASSINI **in** the J2000 reference frame, corrected **for** aberrations.
- Compute the state transformation matrix **from J2000** to IAU_PHOEBE at the epoch, adjusted **for** light time.
- Multiply the state of Phoebe relative to CASSINI **in** the J2000 reference frame by the state transformation matrix computed **in** the previous step.
- Compute the position of Earth relative to CASSINI **in** the J2000 reference frame, corrected **for** aberrations.

(continues on next page)

(continued from previous page)

```
-- Determine what the nominal boresight of the CASSINI high gain
antenna is by examining the frame kernel's content.

-- Compute the rotation matrix from the CASSINI high gain antenna
frame to J2000.

-- Multiply the nominal boresight expressed in the CASSINI high
gain antenna frame by the rotation matrix from the previous
step.

-- Compute the separation between the result of the previous step
and the apparent position of the Earth relative to CASSINI in
the J2000 frame.
```

HINT: Several of the steps above may be compressed into a single step using SpiceyPy functions with which you are already familiar. The “long way” presented above is intended to facilitate the introduction of the functions `spiceypy.pxform` and `spiceypy.sxform`.

You may find it useful to consult the permuted index, the headers of various source modules, and the following toolkit documentation:

1. Frames Required Reading (`frames.req`)
2. PCK Required Reading (`pck.req`)
3. SPK Required Reading (`spk.req`)
4. CK Required Reading (`ck.req`)

This particular example makes use of many of the different types of SPICE kernels. You should spend a few moments thinking about which kernels you will need and what data they provide.

6.2.21 Solution

Solution Meta-Kernel

The meta-kernel we created for the solution to this exercise is named ‘`xform.tm`’. Its contents follow:

```
KPL/MK

This is the meta-kernel used in the solution of the "Spacecraft
Orientation and Reference Frames" task in the Remote Sensing
Hands On Lesson.

The names and contents of the kernels referenced by this
meta-kernel are as follows:

File name                      Contents
-----
naif0008.tls                   Generic LSK
cas00084.tsc                   Cassini SCLK
981005_PLTEPH-DE405S.bsp       Solar System Ephemeris
020514_SE_SAT105.bsp          Saturnian Satellite Ephemeris
030201AP_SK_SM546_T45.bsp      Cassini Spacecraft SPK
cas_v37.tf                     Cassini FK
```

(continues on next page)

(continued from previous page)

```

04135_04171pc_psiv2.bc      Cassini Spacecraft CK
cpck05Mar2004.tpc          Cassini Project PCK

\begindata
KERNELS_TO_LOAD = ( 'kernels/lsk/naif0008.tls',
                    'kernels/sclk/cas00084.tsc',
                    'kernels/spk/981005_PLTEPH-DE405S.bsp',
                    'kernels/spk/020514_SE_SAT105.bsp',
                    'kernels/spk/030201AP_SK_SM546_T45.bsp',
                    'kernels/fk/cas_v37.tf',
                    'kernels/ck/04135_04171pc_psiv2.bc',
                    'kernels/pck/cpck05Mar2004.tpc' )

\beginext

```

Solution Source Code

A sample solution to the problem follows:

```

#
# Solution xform.py
#
from __future__ import print_function
from builtins import input

import spiceypy

def xform():
    #
    # Local parameters
    #
    METAKR = 'xform.tm'

    #
    # Load the kernels that this program requires. We
    # will need:
    #
    #   A leapseconds kernel
    #   A spacecraft clock kernel for CASSINI
    #   The necessary ephemerides
    #   A planetary constants file (PCK)
    #   A spacecraft orientation kernel for CASSINI (CK)
    #   A frame kernel (TF)
    #
    spiceypy.furnsh( METAKR )

    #
    # Prompt the user for the input time string.
    #
    utctim = input( 'Input UTC Time: ' )

    print( 'Converting UTC Time: {:s}'.format(utctim) )

    #
    # Convert utctim to ET.
    #
    et = spiceypy.str2et( utctim )

```

(continues on next page)

(continued from previous page)

```

print( '    ET seconds past J2000: {:.16.3f}'.format(et) )

#
# Compute the apparent state of Phoebe as seen from
# CASSINI in the J2000 frame.
#
[state, ltime] = spiceypy.spkezr( 'PHOEBE', et,          'J2000',
                                'LT+S',    'CASSINI'      )

#
# Now obtain the transformation from the inertial
# J2000 frame to the non-inertial body-fixed IAU_PHOEBE
# frame. Since we want the apparent position, we
# need to subtract ltime from et.
#
sform = spiceypy.sxform( 'J2000', 'IAU_PHOEBE', et-ltime )

#
# Now rotate the apparent J2000 state into IAU_PHOEBE
# with the following matrix multiplication:
#
bfixst = spiceypy.mxvg ( sform, state, 6, 6 )

#
# Display the results.
#
print( '    Apparent state of Phoebe as seen '
      'from CASSINI in the IAU_PHOEBE\n'
      '    body-fixed frame (km, km/s):'      )
print( '        X = {:.19.6f}'.format(bfixst[0]) )
print( '        Y = {:.19.6f}'.format(bfixst[1]) )
print( '        Z = {:.19.6f}'.format(bfixst[2]) )
print( '        VX = {:.19.6f}'.format(bfixst[3]) )
print( '        VY = {:.19.6f}'.format(bfixst[4]) )
print( '        VZ = {:.19.6f}'.format(bfixst[5]) )

#
# It is worth pointing out, all of the above could
# have been done with a single use of spkezr:
#
[state, ltime] = spiceypy.spkezr(
    'PHOEBE', et,          'IAU_PHOEBE',
    'LT+S',    'CASSINI'      )

#
# Display the results.
#
print( '    Apparent state of Phoebe as seen '
      'from CASSINI in the IAU_PHOEBE\n'
      '    body-fixed frame (km, km/s) '
      'obtained using spkezr directly:'      )
print( '        X = {:.19.6f}'.format(state[0]) )
print( '        Y = {:.19.6f}'.format(state[1]) )
print( '        Z = {:.19.6f}'.format(state[2]) )
print( '        VX = {:.19.6f}'.format(state[3]) )
print( '        VY = {:.19.6f}'.format(state[4]) )
print( '        VZ = {:.19.6f}'.format(state[5]) )

```

(continues on next page)

(continued from previous page)

```

#
# Note that the velocity found by using spekr
# to compute the state in the IAU_PHOEBE frame differs
# at the few mm/second level from that found previously
# by calling spekr and then sxform. Computing
# velocity via a single call to spekr as we've
# done immediately above is slightly more accurate because
# it accounts for the effect of the rate of change of
# light time on the apparent angular velocity of the
# target's body-fixed reference frame.
#
# Now we are to compute the angular separation between
# the apparent position of the Earth as seen from the
# orbiter and the nominal boresight of the high gain
# antenna. First, compute the apparent position of
# the Earth as seen from CASSINI in the J2000 frame.
#
[pos, ltime] = spiceypy.spkpos( 'EARTH', et, 'J2000',
                                'LT+S', 'CASSINI' )

#
# Now compute the location of the antenna boresight
# at this same epoch. From reading the frame kernel
# we know that the antenna boresight is nominally the
# +Z axis of the CASSINI_HGA frame defined there.
#
bsight = [ 0.0, 0.0, 1.0]

#
# Now compute the rotation matrix from CASSINI_HGA into
# J2000.
#
pform = spiceypy.pxform( 'CASSINI_HGA', 'J2000', et )

#
# And multiply the result to obtain the nominal
# antenna boresight in the J2000 reference frame.
#
bsight = spiceypy.mxd( pform, bsight )

#
# Lastly compute the angular separation.
#
sep = spiceypy.convt( spiceypy.vsep(bsight, pos),
                      'RADIANS', 'DEGREES' )

print( ' Angular separation between the '
      'apparent position of\n'
      ' Earth and the CASSINI high '
      'gain antenna boresight (degrees):\n'
      '      {:.16.3f}'.format(sep) )

#
# Or alternatively we can work in the antenna
# frame directly.
#
[pos, ltime] = spiceypy.spkpos(

```

(continues on next page)

(continued from previous page)

```

        'EARTH', et, 'CASSINI_HGA',
        'LT+S', 'CASSINI' )

#
# The antenna boresight is the Z-axis in the
# CASSINI_HGA frame.
#
bsight = [ 0.0, 0.0, 1.0 ]

#
# Lastly compute the angular separation.
#
sep = spiceypy.convrt( spiceypy.vsep(bsight, pos),
                      'RADIANS', 'DEGREES' )

print( '    Angular separation between the '
      'apparent position of\n'
      '    Earth and the CASSINI high '
      'gain antenna boresight computed\n'
      '    using vectors in the CASSINI_HGA '
      'frame (degrees):\n'
      '    {:16.3f}'.format(sep) )

spiceypy.unload( METAKR )

if __name__ == '__main__':
    xform()

```

Solution Sample Output

Execute the program:

```

Input UTC Time: 2004 jun 11 19:32:00
Converting UTC Time: 2004 jun 11 19:32:00
ET seconds past J2000: 140254384.185
Apparent state of Phoebe as seen from CASSINI in the IAU_PHOEBE
body-fixed frame (km, km/s):
X = -1982.639762
Y = -934.530471
Z = -166.562595
VX = 3.970833
VY = -3.812498
VZ = -2.371663
Apparent state of Phoebe as seen from CASSINI in the IAU_PHOEBE
body-fixed frame (km, km/s) obtained using spkezzr directly:
X = -1982.639762
Y = -934.530471
Z = -166.562595
VX = 3.970832
VY = -3.812496
VZ = -2.371663
Angular separation between the apparent position of
Earth and the CASSINI high gain antenna boresight (degrees):
71.924
Angular separation between the apparent position of
Earth and the CASSINI high gain antenna boresight computed
using vectors in the CASSINI_HGA frame (degrees):

```

(continues on next page)

71.924

6.2.22 Extra Credit

In this “extra credit” section you will be presented with more complex tasks, aimed at improving your understanding of frame transformations, and some common errors that may happen when computing them.

These “extra credit” tasks are provided as task statements, and unlike the regular tasks, no approach or solution source code is provided. In the next section, you will find the numeric solutions (when applicable) and answers to the questions asked in these tasks.

Task statements and questions

1. Run the original program using the input UTC time "2004 jun 11 18:25:00". Explain what happens.
2. Compute the angular separation between the apparent position of the Sun as seen from CASSINI and the nominal boresight of the CASSINI high gain antenna (HGA). Is the HGA illuminated?

Solutions and answers

```
1. When running the original software using as input the UTC time
   string "2004 jun 11 18:25:00":

Traceback (most recent call last):
  File "xform.py", line 183, in <module>
    xform()
  File "xform.py", line 130, in xform
    pform = spiceypy.pxform( 'CASSINI_HGA', 'J2000', et )
  File "/home/bsemenov/local/lib/python3.5/site-packages/spiceypy/spi
ceypy.py", line 76, in with_errcheck
    check_for_spice_error(f)
  File "/home/bsemenov/local/lib/python3.5/site-packages/spiceypy/spi
ceypy.py", line 59, in check_for_spice_error
    raise stypes.SpiceyError(msg)
spiceypy.utils.support_types.SpiceyError:
=====
=====

Toolkit version: N0066

SPICE(NOFRAMECONNECT) --

At epoch 1.4025036418463E+08 TDB (2004 JUN 11 18:26:04.184 TDB), ther
e is insufficient information available to transform from reference f
rame -82101 (CASSINI_HGA) to reference frame 1 (J2000). Frame CASSINI
_HGA could be transformed to frame -82000 (CASSINI_SC_COORD). The lat
ter is a CK frame; a CK file containing data

pxform_c --> PXFORM --> REFCHG

=====
=====
```

(continues on next page)

(continued from previous page)

spiceypy.pxform returns the SPICE(NOFRAMECONNECT) error, which indicates that there are **not** sufficient data to perform the transformation **from the** CASSINI_HGA frame to J2000 at the requested epoch. If you summarize the CASSINI spacecraft CK using the "ckbrief" utility program **with** the -dump option (display interpolation intervals boundaries) you will find that the CK contains gaps within its segment:

```
CKBRIEF -- Version 6.1.0, June 27, 2014 -- Toolkit Version N0066
```

```
Summary for: kernels/ck/04135_04171pc_psiv2.bc
```

```
Segment No.: 1
```

```
Object: -82000
```

Interval Begin UTC	Interval End UTC	AV
2004-JUN-11 05:00:00.000	2004-JUN-11 09:25:02.019	Y
2004-JUN-11 09:26:14.019	2004-JUN-11 18:24:37.152	Y
2004-JUN-11 18:26:13.152	2004-JUN-12 05:53:26.012	Y
2004-JUN-12 05:54:56.012	2004-JUN-12 10:32:08.016	Y
2004-JUN-12 10:33:26.016	2004-JUN-12 11:59:59.998	Y

whereas **if** you had used ckbrief without -dump you would have gotten the following information (only CK segment begin/end times):

```
CKBRIEF -- Version 6.1.0, June 27, 2014 -- Toolkit Version N0066
```

```
Summary for: kernels/ck/04135_04171pc_psiv2.bc
```

```
Object: -82000
```

Interval Begin UTC	Interval End UTC	AV
2004-JUN-11 05:00:00.000	2004-JUN-12 11:59:59.998	Y

which has insufficient detail to reveal the problem.

- By computing the apparent position of the Sun **as** seen **from** **CASSINI in** the CASSINI_HGA frame, **and** the angular separation between this vector **and** the nominal boresight of the CASSINI high gain antenna (+Z-axis of the CASSINI_HGA frame), you will find whether the HGA **is** illuminated. The solution **for** the input UTC time "2004 jun 11 19:32:00" **is**:

Angular separation between the apparent position of the Sun **and** the nominal boresight of the CASSINI high gain antenna (degrees):
73.130

(continues on next page)

(continued from previous page)

```
HGA illumination:
    CASSINI high gain antenna IS illuminated.

    since the angular separation is smaller than 90 degrees.
```

6.2.23 Computing Sub-s/c and Sub-solar Points on an Ellipsoid and a DSK (subpts)

6.2.24 Task Statement

Write a program that prompts the user for an input UTC time string and computes the following quantities at that epoch:

1. The apparent sub-observer point of CASSINI on Phoebe, **in** the body fixed frame IAU_PHOEBE, **in** kilometers.
2. The apparent sub-solar point on Phoebe, **as** seen **from** CASSINI **in** the body fixed frame IAU_PHOEBE, **in** kilometers.

The program computes each point twice: once using an ellipsoidal shape model and the

```
near point/ellipsoid
```

definition, and once using a DSK shape model and the

```
nadir/dsk/unprioritized
```

definition.

The program displays the results. Use the program to compute these quantities at “2004 jun 11 19:32:00” UTC.

6.2.25 Learning Goals

Discover higher level geometry calculation functions in SpiceyPy and their usage as it relates to CASSINI.

6.2.26 Approach

This particular problem is more of an exercise in searching the permuted index to find the appropriate functions and then reading their headers to understand how to call them.

One point worth considering: how would the results change if the sub-solar and sub-observer points were computed using the

```
intercept/ellipsoid
```

and

```
intercept/dsk/unprioritized
```

definitions? Which definition is appropriate?

6.2.27 Solution

Solution Meta-Kernel

The meta-kernel we created for the solution to this exercise is named 'subpts.tm'. Its contents follow:

KPL/MK

This **is** the meta-kernel used **in** the solution of the "Computing Sub-spacecraft and Sub-solar Points" task **in** the Remote Sensing Hands On Lesson.

The names **and** contents of the kernels referenced by this meta-kernel are **as** follows:

File name	Contents
naif0008.tls	Generic LSK
981005_PLTEPH-DE405S.bsp	Solar System Ephemeris
020514_SE_SAT105.bsp	Saturnian Satellite Ephemeris
030201AP_SK_SM546_T45.bsp	Cassini Spacecraft SPK
cpck05Mar2004.tpc	Cassini Project PCK
phoebe_64q.bds	Phoebe DSK

```
\begindata
KERNELS_TO_LOAD = ( 'kernels/lsk/naif0008.tls',
                    'kernels/spk/981005_PLTEPH-DE405S.bsp',
                    'kernels/spk/020514_SE_SAT105.bsp',
                    'kernels/spk/030201AP_SK_SM546_T45.bsp',
                    'kernels/pck/cpck05Mar2004.tpc'
                    'kernels/dsk/phoebe_64q.bds' )

\begin text
```

Solution Source Code

A sample solution to the problem follows:

```
#
# Solution subpts.py
#
from __future__ import print_function
from builtins import input

#
# SpiceyPy package:
#
import spiceypy

def subpts():
    #
    # Local parameters
    #
    METAKR = 'subpts.tm'

    #
    # Load the kernels that this program requires. We
```

(continues on next page)

(continued from previous page)

```

# will need:
#
#   A leapseconds kernel
#   The necessary ephemerides
#   A planetary constants file (PCK)
#   A DSK file containing Phoebe shape data
#
spiceypy.furnsh( METAKR )

#
#Prompt the user for the input time string.
#
utctim = input( 'Input UTC Time: ' )

print( ' Converting UTC Time: {:s}'.format(utctim) )

#
#Convert utctim to ET.
#
et = spiceypy.str2et( utctim )

print( '   ET seconds past J2000: {:16.3f}'.format(et) )

for i in range(2):

    if i == 0:
        #
        # Use the "near point" sub-point definition
        # and an ellipsoidal model.
        #
        method = 'NEAR POINT/Ellipsoid'

    else:
        #
        # Use the "nadir" sub-point definition
        # and a DSK model.
        #
        method = 'NADIR/DSK/Unprioritized'

    print( '\n Sub-point/target shape model: {:s}\n'.format(
        method ) )

    #
    # Compute the apparent sub-observer point of CASSINI
    # on Phoebe.
    #
    [spoint, trgepc, srfvec] = spiceypy.subpnt(
        method, 'PHOEBE', et,
        'IAU_PHOEBE', 'LT+S', 'CASSINI' )

    print( '   Apparent sub-observer point of CASSINI '
        'on Phoebe in the\n'
        '   IAU_PHOEBE frame (km):' )
    print( '       X = {:16.3f}'.format(spoint[0]) )
    print( '       Y = {:16.3f}'.format(spoint[1]) )
    print( '       Z = {:16.3f}'.format(spoint[2]) )
    print( '       ALT = {:16.3f}'.format(spiceypy.vnorm(srfvec)) )

```

(continues on next page)

(continued from previous page)

```
#
# Compute the apparent sub-solar point on Phoebe
# as seen from CASSINI.
#
[spoint, trgepc, srfvec] = spiceypy.subslr(
    method,      'PHOEBE', et,
    'IAU_PHOEBE', 'LT+S', 'CASSINI' )

print( '    Apparent sub-solar point on Phoebe '
      'as seen from CASSINI in\n'
      'the IAU_PHOEBE frame (km):' )
print( '        X = {:.16.3f}'.format(spoint[0]) )
print( '        Y = {:.16.3f}'.format(spoint[1]) )
print( '        Z = {:.16.3f}'.format(spoint[2]) )

#
# End of computation block for "method"
#
print( " )

spiceypy.unload( METAKR )

if __name__ == '__main__':
    subpts()
```

Solution Sample Output

Execute the program:

```
Input UTC Time: 2004 jun 11 19:32:00
Converting UTC Time: 2004 jun 11 19:32:00
ET seconds past J2000: 140254384.185

Sub-point/target shape model: NEAR POINT/Ellipsoid

Apparent sub-observer point of CASSINI on Phoebe in the
IAU_PHOEBE frame (km):
X = 104.498
Y = 45.269
Z = 7.383
ALT = 2084.116
Apparent sub-solar point on Phoebe as seen from CASSINI in
the IAU_PHOEBE frame (km):
X = 78.681
Y = 76.879
Z = -21.885

Sub-point/target shape model: NADIR/DSK/Unprioritized

Apparent sub-observer point of CASSINI on Phoebe in the
IAU_PHOEBE frame (km):
X = 95.373
Y = 40.948
Z = 6.610
ALT = 2094.242
Apparent sub-solar point on Phoebe as seen from CASSINI in
```

(continues on next page)

(continued from previous page)

```

the IAU_PHOEBE frame (km):
X =          79.111
Y =          77.338
Z =         -22.028

```

6.2.28 Extra Credit

In this “extra credit” section you will be presented with more complex tasks, aimed at improving your understanding of `spiceypy.subpnt` and `spiceypy.subslr` functions.

These “extra credit” tasks are provided as task statements, and unlike the regular tasks, no approach or solution source code is provided. In the next section, you will find the numeric solutions (when applicable) and answers to the questions asked in these tasks.

Task statements and questions

1. Recompute the apparent sub-solar point on Phoebe **as** seen **from** **CASSINI** **in** the body fixed frame IAU_PHOEBE **in** kilometers using the `'Intercept/ellipsoid'` method at `"2004 jun 11 19:32:00"`. Explain the differences.
2. Compute the geometric sub-spacecraft point of CASSINI on Phoebe **in** the body fixed frame IAU_PHOEBE **in** kilometers using the `'Near point/ellipsoid'` method at `"2004 jun 11 19:32:00"`.
3. Transform the sub-spacecraft Cartesian coordinates obtained **in** the previous task to planetocentric **and** planetographic coordinates. When computing planetographic coordinates, retrieve Phoebe's radii by calling `spiceypy.bodvrd` and use the first element of the returned radii values **as** Phoebe's equatorial radius. Explain why planetocentric **and** planetographic latitudes **and** longitudes are different. Explain why the planetographic altitude **for** a point on the surface of Phoebe **is not** zero **and** whether this **is** correct **or not**.

Solutions and answers

1. The differences observed are due to the computation method. The `"Intercept/ellipsoid"` method defines the sub-solar point **as** the target surface intercept of the line containing the Sun **and** the target's center, while the `"Near point/ellipsoid"` method defines the sub-solar point **as** the nearest point on the target relative to the Sun. Since Phoebe **is not** spherical, these two points are **not** the same:

```

Apparent sub-solar point on Phoebe as seen from CASSINI in
the IAU_PHOEBE frame using the 'Near Point: ellipsoid' method
(km):
X =          78.681
Y =          76.879
Z =         -21.885

```

```

Apparent sub-solar point on Phoebe as seen from CASSINI in
the IAU_PHOEBE frame using the 'Intercept: ellipsoid' method
(km):

```

(continues on next page)

(continued from previous page)

```
X = 74.542
Y = 79.607
Z = -24.871
```

2. The geometric sub-spacecraft point of CASSINI on Phoebe **in** the body fixed frame IAU_PHOEBE **in** kilometers at "2004 jun 11 19:32:00" UTC epoch is:

Geometric sub-spacecraft point of CASSINI on Phoebe **in** the IAU_PHOEBE frame using the 'Near Point: ellipsoid' method (km):

```
X = 104.497
Y = 45.270
Z = 7.384
```

3. The sub-spacecraft point of CASSINI on Phoebe **in** planetocentric **and** planetographic coordinates at "2004 jun 11 19:32:00" UTC epoch **is**:

Planetocentric coordinates of the CASSINI sub-spacecraft point on Phoebe (degrees, km):

```
LAT = 3.710
LON = 23.423
R = 114.121
```

Planetographic coordinates of the CASSINI sub-spacecraft point on Phoebe (degrees, km):

```
LAT = 4.454
LON = 336.577
ALT = -0.831
```

The planetocentric **and** planetographic longitudes are different ("graphic" = 360 - "centric") because planetographic longitudes on Phoebe are measured positive west **as** defined by Phoebe's rotation direction.

The planetocentric **and** planetographic latitudes are different because the planetocentric latitude was computed **as** the angle between the direction **from the** center of the body to the point **and** the equatorial plane, **while** the planetographic latitude was computed **as** the angle between the surface normal at the point **and** the equatorial plane.

The planetographic altitude **is** non zero because it was computed using a different **and** incorrect Phoebe surface model: a spheroid **with** equal equatorial radii. The surface point returned by spiceypy.subpnt was computed by treating Phoebe **as** a triaxial ellipsoid **with** different equatorial radii. The planetographic latitude **is** also incorrect because it **is** based on the normal to the surface of the spheroid rather than the ellipsoid, In general planetographic coordinates cannot be used **for** bodies **with** shapes modeled **as** triaxial ellipsoids.

6.2.29 Intersecting Vectors with an Ellipsoid and a DSK (fovint)

6.2.30 Task Statement

Write a program that prompts the user for an input UTC time string and, for that time, computes the intersection of the CASSINI ISS NAC camera boresight and field of view (FOV) boundary vectors with the surface of Phoebe. Compute each intercept twice: once with Phoebe's shape modeled as an ellipsoid, and once with Phoebe's shape modeled by DSK data. The program presents each point of intersection as

1. A Cartesian vector **in** the IAU_PHOEBE frame
2. Planetocentric (latitudinal) coordinates **in** the IAU_PHOEBE frame.

For each of the camera FOV boundary and boresight vectors, if an intersection is found, the program displays the results of the above computations, otherwise it indicates no intersection exists.

At each point of intersection compute the following:

3. Phase angle
4. Solar incidence angle
5. Emission angle

These angles should be computed using both ellipsoidal and DSK shape models.

Additionally compute the local solar time at the intercept of the camera boresight with the surface of Phoebe, using both ellipsoidal and DSK shape models.

Use this program to compute values at the epoch:

```
"2004 jun 11 19:32:00" UTC
```

6.2.31 Learning Goals

Understand how field of view parameters are retrieved from instrument kernels. Learn how various standard planetary constants are retrieved from text PCKs. Discover how to compute the intersection of field of view vectors with target bodies whose shapes are modeled as ellipsoids or provided by DSKs. Discover another high level geometry function and another time conversion function in SpiceyPy.

6.2.32 Approach

This problem can be broken down into several simple, small steps:

- Decide which SPICE kernels are necessary. Prepare a meta-kernel listing the kernels **and** load it into the program. Remember, you will need to find a kernel **with** information about the CASSINI NAC camera.
- Prompt the user **for** an **input** time string.
- Convert the **input** time string into ephemeris time expressed **as** seconds past J2000 TDB.

(continues on next page)

(continued from previous page)

```
-- Retrieve the FOV (field of view) configuration for the CASSINI
   NAC camera.
```

For each vector in the set of boundary corner vectors, and for the boresight vector, perform the following operations:

```
-- Compute the intercept of the vector with Phoebe modeled as an
   ellipsoid or using DSK data

-- If this intercept is found, convert the position vector of the
   intercept into planetocentric coordinates.

   Then compute the phase, solar incidence, and emission angles at
   the intercept. Otherwise indicate to the user no intercept was
   found for this vector.

-- Compute the planetocentric longitude of the boresight
   intercept.
```

Finally

```
-- Compute the local solar time at the boresight intercept
   longitude on a 24-hour clock. The input time for this
   computation should be the TDB observation epoch minus one-way
   light time from the boresight intercept to the spacecraft.
```

It may be useful to consult the CASSINI ISS instrument kernel to determine the name of the NAC camera as well as its configuration. This exercise may make use of some of the concepts and (loosely) code from the “Spacecraft Orientation and Reference Frames” task.

6.2.33 Solution

Solution Meta-Kernel

The meta-kernel we created for the solution to this exercise is named ‘fovint.tm’. Its contents follow:

```
KPL/MK

This is the meta-kernel used in the solution of the
"Intersecting Vectors with a Triaxial Ellipsoid" task
in the Remote Sensing Hands On Lesson.

The names and contents of the kernels referenced by this
meta-kernel are as follows:

File name                               Contents
-----
naif0008.tls                            Generic LSK
cas00084.tsc                             Cassini SCLK
981005_PLTEPH-DE405S.bsp                 Solar System Ephemeris
020514_SE_SAT105.bsp                     Saturnian Satellite Ephemeris
030201AP_SK_SM546_T45.bsp                Cassini Spacecraft SPK
cas_v37.tf                               Cassini FK
04135_04171pc_psiv2.bc                   Cassini Spacecraft CK
cpck05Mar2004.tpc                         Cassini Project PCK
cas_iss_v09.ti                           ISS Instrument Kernel
```

(continues on next page)

(continued from previous page)

```

phoebe_64q.bds          Phoebe DSK

\begindata
KERNELS_TO_LOAD = ( 'kernels/lsk/naif0008.tls',
                    'kernels/sclk/cas00084.tsc',
                    'kernels/spk/981005_PLTEPH-DE405S.bsp',
                    'kernels/spk/020514_SE_SAT105.bsp',
                    'kernels/spk/030201AP_SK_SM546_T45.bsp',
                    'kernels/fk/cas_v37.tf',
                    'kernels/ck/04135_04171pc_psiv2.bc',
                    'kernels/pck/cpck05Mar2004.tpc',
                    'kernels/ik/cas_iss_v09.ti'
                    'kernels/dsk/phoebe_64q.bds' )

\begintext

```

Solution Source Code

A sample solution to the problem follows:

```

#
# Solution fovint.py
#
from __future__ import print_function
from builtins import input

#
# SpiceyPy package:
#
import spiceypy
from spiceypy.utils.support_types import SpiceyError

def fovint():
    #
    # Local parameters
    #
    METAKR = 'fovint.tm'
    ROOM   = 4

    #
    # Load the kernels that this program requires. We
    # will need:
    #
    #     A leapseconds kernel.
    #     A SCLK kernel for CASSINI.
    #     Any necessary ephemerides.
    #     The CASSINI frame kernel.
    #     A CASSINI C-kernel.
    #     A PCK file with Phoebe constants.
    #     The CASSINI ISS I-kernel.
    #     A DSK file containing Phoebe shape data.
    #
    spiceypy.furnsh( METAKR )

    #
    # Prompt the user for the input time string.
    #

```

(continues on next page)

(continued from previous page)

```

utctim = input( 'Input UTC Time: ' )

print( 'Converting UTC Time: {:s}'.format(utctim) )

#
#Convert utctim to ET.
#
et = spiceypy.str2et( utctim )

print( ' ET seconds past J2000: {:.16.3f}\n'.format(et) )

#
# Now we need to obtain the FOV configuration of
# the ISS NAC camera. To do this we will need the
# ID code for CASSINI_ISS_NAC.
#
try:
    nacid = spiceypy.bodn2c( 'CASSINI_ISS_NAC' )

except SpiceyError:
    #
    # Stop the program if the code was not found.
    #
    print( 'Unable to locate the ID code for '
           'CASSINI_ISS_NAC' )
    raise

#
# Now retrieve the field of view parameters.
#
[ shape, insfrm,
  bsight, n,      bounds ] = spiceypy.getfov( nacid, ROOM )

#
# `bounds' is a numpy array. We'll convert it to a list.
#
# Rather than treat BSIGHT as a separate vector,
# copy it into the last slot of BOUNDS.
#
bounds = bounds.tolist()
bounds.append( bsight )

#
# Set vector names to be used for output.
#
vecnam = [ 'Boundary Corner 1',
           'Boundary Corner 2',
           'Boundary Corner 3',
           'Boundary Corner 4',
           'Cassini NAC Boresight' ]

#
# Set values of "method" string that specify use of
# ellipsoidal and DSK (topographic) shape models.
#
# In this case, we can use the same methods for calls to both
# spiceypy.sincpt and spiceypy.ilumin. Note that some SPICE

```

(continues on next page)

(continued from previous page)

```

# routines require different "method" inputs from those
# shown here. See the API documentation of each routine
# for details.
#
method = [ 'Ellipsoid', 'DSK/Unprioritized']

#
# Get ID code of Phoebe. We'll use this ID code later, when we
# compute local solar time.
#
try:
    phoeid = spiceypy.bodn2c( 'PHOEBE' )
except:
    #
    # The ID code for PHOEBE is built-in to the library.
    # However, it is good programming practice to get
    # in the habit of handling exceptions that may
    # be thrown when a quantity is not found.
    #
    print( 'Unable to locate the body ID code '
           'for Phoebe.' )
    raise

#
# Now perform the same set of calculations for each
# vector listed in the BOUNDS array. Use both
# ellipsoidal and detailed (DSK) shape models.
#
for i in range(5):
    #
    # Call sincpt to determine coordinates of the
    # intersection of this vector with the surface
    # of Phoebe.
    #
    print( 'Vector: {:s}\n'.format( vecnam[i] ) )

    for j in range(2):

        print ( ' Target shape model: {:s}\n'.format(
                method[j] ) )

        try:

            [point, trgepc, srfvec ] = spiceypy.sincpt(
                method[j], 'PHOEBE', et,
                'IAU_PHOEBE', 'LT+S', 'CASSINI',
                insfrm, bounds[i] )

            #
            # Now, we have discovered a point of intersection.
            # Start by displaying the position vector in the
            # IAU_PHOEBE frame of the intersection.
            #
            print( ' Position vector of surface intercept '
                  'in the IAU_PHOEBE frame (km):' )
            print( '      X = {:.16.3f}'.format( point[0] ) )
            print( '      Y = {:.16.3f}'.format( point[1] ) )
            print( '      Z = {:.16.3f}'.format( point[2] ) )

```

(continues on next page)

(continued from previous page)

```

#
# Display the planetocentric latitude and longitude
# of the intercept.
#
[radius, lon, lat] = spiceypy.reclat( point )

print( ' Planetocentric coordinates of '
      'the intercept (degrees):'
      )
print( '      LAT = {:.16.3f}'.format(
      lat * spiceypy.dpr() ) )
print( '      LON = {:.16.3f}'.format(
      lon * spiceypy.dpr() ) )

#
# Compute the illumination angles at this
# point.
#
[ trgepc, srfvec, phase, solar,
  emissn, visibl, lit ] = \
    spiceypy.illumf(
        method[j], 'PHOEBE', 'SUN', et,
        'IAU_PHOEBE', 'LT+S', 'CASSINI', point )

print( ' Phase angle (degrees): '
      '{:.16.3f}'.format( phase*spiceypy.dpr() ) )
print( ' Solar incidence angle (degrees): '
      '{:.16.3f}'.format( solar*spiceypy.dpr() ) )
print( ' Emission angle (degrees): '
      '{:.16.3f}'.format( emissn*spiceypy.dpr() ) )
print( ' Observer visible: {:.s}'.format(
      str(visibl) ) )
print( ' Sun visible: {:.s}'.format(
      str(lit) ) )

if i == 4:
    #
    # Compute local solar time corresponding
    # to the light time corrected TDB epoch
    # at the boresight intercept.
    #
    [hr, mn, sc, time, ampm] = spiceypy.et2lst(
        trgepc,
        phoeid,
        lon,
        'PLANETOCENTRIC' )

    print( '\n Local Solar Time at boresight '
          'intercept (24 Hour Clock):\n'
          '{:.s}'.format( time ) )

#
# End of LST computation block.
#

except SpiceyError as exc:
    #
    # Display a message if an exception was thrown.
    # For simplicity, we treat this as an indication

```

(continues on next page)

(continued from previous page)

```

        # that the point of intersection was not found,
        # although it could be due to other errors.
        # Otherwise, continue with the calculations.
        #
        print( 'Exception message is: {:s}'.format(
            exc.value ))

    #
    # End of SpiceyError try-catch block.
    #
    print( " )

#
# End of target shape model loop.
#

#
# End of vector loop.
#

spiceypy.unload( METAKR )

if __name__ == '__main__':
    fovint()

```

Solution Sample Output

Execute the program:

```

Input UTC Time: 2004 jun 11 19:32:00
Converting UTC Time: 2004 jun 11 19:32:00
    ET seconds past J2000:    140254384.185

Vector: Boundary Corner 1

Target shape model: Ellipsoid

Position vector of surface intercept in the IAU_PHOEBE frame (km):
    X   =    91.026
    Y   =    67.190
    Z   =     2.030
Planetocentric coordinates of the intercept (degrees):
    LAT =     1.028
    LON =    36.432
Phase angle (degrees):                28.110
Solar incidence angle (degrees):      16.121
Emission angle (degrees):             14.627
Observer visible: true
Sun visible: true

Target shape model: DSK/Unprioritized

Position vector of surface intercept in the IAU_PHOEBE frame (km):
    X   =    78.770
    Y   =    61.570
    Z   =     0.964
Planetocentric coordinates of the intercept (degrees):
    LAT =     0.552
    LON =    38.013
Phase angle (degrees):                28.110

```

(continues on next page)

(continued from previous page)

```

Solar incidence angle (degrees):      31.132
Emission angle (degrees):            16.539
Observer visible: true
Sun visible:      true

```

Vector: Boundary Corner 2

Target shape model: Ellipsoid

Position vector of surface intercept **in** the IAU_PHOEBE frame (km):

```

X  =      89.991
Y  =      66.726
Z  =      14.733

```

Planetocentric coordinates of the intercept (degrees):

```

LAT =      7.492
LON =     36.556

```

Phase angle (degrees): 27.894

Solar incidence angle (degrees): 22.894

Emission angle (degrees): 14.988

Observer visible: true

Sun visible: true

Target shape model: DSK/Unprioritized

Position vector of surface intercept **in** the IAU_PHOEBE frame (km):

```

X  =      76.586
Y  =      60.579
Z  =      13.657

```

Planetocentric coordinates of the intercept (degrees):

```

LAT =      7.962
LON =     38.344

```

Phase angle (degrees): 27.894

Solar incidence angle (degrees): 32.013

Emission angle (degrees): 11.845

Observer visible: true

Sun visible: true

Vector: Boundary Corner 3

Target shape model: Ellipsoid

Position vector of surface intercept **in** the IAU_PHOEBE frame (km):

```

X  =      80.963
Y  =      76.643
Z  =      14.427

```

Planetocentric coordinates of the intercept (degrees):

```

LAT =      7.373
LON =     43.430

```

Phase angle (degrees): 28.171

Solar incidence angle (degrees): 21.315

Emission angle (degrees): 21.977

Observer visible: true

Sun visible: true

Target shape model: DSK/Unprioritized

Position vector of surface intercept **in** the IAU_PHOEBE frame (km):

(continues on next page)

(continued from previous page)

```

X      =      68.677
Y      =      71.100
Z      =      13.444
Planetocentric coordinates of the intercept (degrees):
LAT =      7.745
LON =     45.993
Phase angle (degrees):      28.171
Solar incidence angle (degrees): 36.039
Emission angle (degrees): 14.474
Observer visible: true
Sun visible:      true

Vector: Boundary Corner 4

Target shape model: Ellipsoid

Position vector of surface intercept in the IAU_PHOEBE frame (km):
X      =      81.997
Y      =      77.106
Z      =      1.698
Planetocentric coordinates of the intercept (degrees):
LAT =      0.865
LON =     43.239
Phase angle (degrees):      28.385
Solar incidence angle (degrees): 13.882
Emission angle (degrees): 21.763
Observer visible: true
Sun visible:      true

Target shape model: DSK/Unprioritized

Position vector of surface intercept in the IAU_PHOEBE frame (km):
X      =      73.186
Y      =      73.131
Z      =      0.934
Planetocentric coordinates of the intercept (degrees):
LAT =      0.517
LON =     44.978
Phase angle (degrees):      28.385
Solar incidence angle (degrees): 41.268
Emission angle (degrees): 17.493
Observer visible: true
Sun visible:      true

Vector: Cassini NAC Boresight

Target shape model: Ellipsoid

Position vector of surface intercept in the IAU_PHOEBE frame (km):
X      =      86.390
Y      =      72.089
Z      =      8.255
Planetocentric coordinates of the intercept (degrees):
LAT =      4.196
LON =     39.844
Phase angle (degrees):      28.139
Solar incidence angle (degrees): 18.247

```

(continues on next page)

(continued from previous page)

```
Emission angle (degrees):          17.858
Observer visible:  true
Sun visible:      true

Local Solar Time at boresight intercept (24 Hour Clock):
  11:31:50

Target shape model: DSK/Unprioritized

Position vector of surface intercept in the IAU_PHOEBE frame (km):
  X   =      74.326
  Y   =      66.602
  Z   =       7.247
Planetocentric coordinates of the intercept (degrees):
  LAT =       4.153
  LON =      41.863
Phase angle (degrees):             28.139
Solar incidence angle (degrees):    33.200
Emission angle (degrees):           9.230
Observer visible:  true
Sun visible:      true

Local Solar Time at boresight intercept (24 Hour Clock):
  11:39:55
```

6.2.34 Extra Credit

There are no “extra credit” tasks for this step of the lesson.

6.3 Geometric Event Finding Hands-On Lesson, using MEX (Python)

November 20, 2017

6.3.1 Overview

This lesson illustrates how the Geometry Finder (GF) subsystem of the SpiceyPy Toolkit can be used to find time intervals when specified geometric conditions are satisfied.

In this lesson the student is asked to construct a program that finds the time intervals, within a specified time range, when the Mars Express Orbiter (MEX) is visible from the DSN station DSS-14. Possible occultation of the spacecraft by Mars is to be considered.

6.3.2 References

This section lists SPICE documents referred to in this lesson.

In some cases the lesson explanations also refer to the information provided in the meta-data area of the kernels used in the lesson examples. It is especially true in case of the FK and IK files, which often contain comprehensive descriptions of the frames, instrument FOVs, etc. Since both FK and IK are text kernels, the information provided in them can be viewed using any text editor, while the meta information provided in binary kernels – SPKs and CKs – can be viewed using “commnt” or “spacit” utility programs located in “cspice/exe” of Toolkit installation tree.

The following SPICE tutorials serve as references for the discussions in this lesson:

Name	Lesson steps/functions it describes
-----	-----
Time	Time Conversion
SCLK and LSK	Time Conversion
SPK	Obtaining Ephemeris Data
Frames	Reference Frames
Using Frames	Reference Frames
PCK	Planetary Constants Data
Lunar-Earth PCK	Lunar and Earth Orientation Data
GF	The SPICE Geometry Finder (GF) subsystem

These tutorials are available from the NAIF ftp server at JPL:

<http://naif.jpl.nasa.gov/naif/tutorials.html>

Required Readings

The Required Reading documents are provided with the Toolkit and are located under the “cspice/doc” directory in the CSPICE Toolkit installation tree.

Name	Lesson steps/functions that it describes
-----	-----
cells.req	Cell/window initialization
frames.req	Using reference frames
gf.req	The SPICE geometry finder (GF) subsystem
kernel.req	Loading SPICE kernels
naif_ids.req	Body and reference frame names
pck.req	Obtaining planetary constants data
spk.req	Computing positions and velocities
time.req	UTC to ET time conversion
windows.req	The SPICE window data type

The Permuted Index

Another useful document distributed with the Toolkit is the permuted index. This is located under the “cspice/doc” directory in the C installation tree.

This text document provides a simple mechanism by which users can discover which SpiceyPy functions perform functions of interest, as well as the names of the source files that contain these functions.

SpiceyPy API Documentation

A SpiceyPy function's parameters specification is available using the built-in Python help system.

For example, the Python help function

```
>>> import spiceypy
>>> help(spiceypy.str2et)
```

describes of the str2et function's parameters, while the document

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/str2et_c.html

describes extensively the str2et functionality.

6.3.3 Kernels Used

The following kernels are used in examples provided in this lesson:

#	FILE NAME	TYPE	DESCRIPTION
1	de405xs.bsp	SPK	Planetary ephemeris SPK, subsetting to cover only time range of interest
2	earthstns_itrf93_050714.bsp	SPK	DSN station SPK
3	earth_topo_050714.tf	FK	DSN station frame definitions
4	earth_000101_060525_060303.bpc	PCK	Binary PCK for Earth
5	naif0008.tls	LSK	Generic LSK
6	ORMM_040501000000_00076XS.BSP	SPK	MEX Orbiter trajectory SPK, subsetting to cover only time range of interest
7	pck00008.tpc	PCK	Generic PCK
8	mars_lowres.bds	DSK	Low-resolution Mars DSK

These SPICE kernels are included in the lesson package available from the NAIF server at JPL:

ftp://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Lessons/

6.3.4 SpiceyPy Modules Used

This section provides a complete list of the functions and kernels that are suggested for usage in each of the exercises in this lesson. (You may wish to not look at this list unless/until you “get stuck” while working on your own.)

CHAPTER	EXERCISE	FUNCTIONS	NON-VOID	KERNELS
1	viewpr	spiceypy.furnsh spiceypy.wninsd spiceypy.gfposc spiceypy.unload	spiceypy.rpd spiceypy.str2et spiceypy.timeout spiceypy.wncard spiceypy.wnfetd	1-7
2	visibl	spiceypy.furnsh spiceypy.wninsd	spiceypy.rpd spiceypy.str2et	1-8

(continues on next page)

(continued from previous page)

	spiceypy.gfposc	spiceypy.timeout	
	spiceypy.gfoclt	spiceypy.wndifd	
	spiceypy.unload	spiceypy.wncard	
		spiceypy.wnfetd	
extra (*)	spiceypy.gfdist	spiceypy.repmc	1,5-7
	spiceypy.kclear	spiceypy.repmf	

(*) Additional APIs **and** kernels used **in** Extra Credit tasks.

Use the Python built-in help system on the various functions listed above for the API parameters' description, and refer to the headers of their corresponding CSPICE versions for detailed interface specifications.

6.3.5 Find View Periods

6.3.6 Task Statement

Write a program that finds the set of time intervals, within the time range

```
2004 MAY 2 TDB
2004 MAY 6 TDB
```

when the Mars Express Orbiter (MEX) is visible from the DSN station DSS-14. These time intervals are frequently called “view periods.”

The spacecraft is considered visible if its apparent position (that is, its position corrected for light time and stellar aberration) has elevation of at least 6 degrees in the topocentric reference frame DSS-14_TOPO. In this exercise, we ignore the possibility of occultation of the spacecraft by Mars.

Use a search step size that ensures that no view periods of duration 5 minutes or longer will be missed by the search.

Display the start and stop times of these intervals using TDB calendar dates and millisecond precision.

6.3.7 Learning Goals

Exposure to SPICE GF event finding routines. Familiarity with SPICE windows and routines that manipulate them. Exposure to SPICE time parsing and output formatting routines.

6.3.8 Approach

Solution steps

A possible solution could consist of the following steps:

Preparation:

1. Decide what SPICE kernels are necessary. Use the SPICE summary tool BRIEF to examine the coverage of the binary kernels **and** verify the availability of required data.
2. Create a meta-kernel listing the SPICE kernels to be loaded. (Hint: consult a programming example tutorial, **or** the Introduction to Kernels tutorial, **for** a reminder of how to do

(continues on next page)

(continued from previous page)

this.)

Name the meta-kernel 'viewpr.tm'.

Next, write a program that performs the following steps:

1. Use `spiceypy.furnsh` to load the meta-kernel.
2. Create confinement **and** output SpiceyPy windows using `stypes.SPICEDOUBLE_CELL`.
3. Insert the given time bounds into the confinement window using `spiceypy.wninsd`.
4. Select a step size **for** searching **for** visibility state transitions: **in** this case, each target rise **or** set event **is** a state transition.

The step size must be large enough so the search proceeds **with** reasonable speed, but small enough so that no visibility transition events---that **is**, target rise **or** set events---are missed.
5. Use the GF routine `spiceypy.gfposc` to find the window of times, within the confinement window, during which the MEX spacecraft **is** above the elevation limit **as** seen **from** DSN station DSS-14, **in** the reference frame DSS-14_TOPO.

Use light time **and** stellar aberration corrections **for** the apparent position of the spacecraft **as** seen **from** the station.
6. Fetch **and** display the contents of the result window. Use `spiceypy.wnfetd` to extract **from** the result window the start **and** stop times of each time interval. Display each of the intervals **in** the result window **as** a pair of start **and** stop times. Express each time **as** a TDB calendar date using the routine `spiceypy.timout`.

You may find it useful to consult the references listed above. In particular, the header of the SPICE GF function `spiceypy.gfposc` contains pertinent documentation.

6.3.9 Solution

Solution Meta-Kernel

The meta-kernel we created for the solution to this exercise is named 'viewpr.tm'. Its contents follow:

KPL/MK

Example meta-kernel **for** geometric event finding hands-on coding lesson.

Version 2.0.0 13-JUL-2017 (JDR)

The names **and** contents of the kernels referenced by this

(continues on next page)

(continued from previous page)

```

meta-kernel are as follows:

File Name                Description
-----
de405xs.bsp              Planetary ephemeris SPK,
                          subsetting to cover only
                          time range of interest.
earthstns_itrf93_050714.bsp DSN station SPK.
earth_topo_050714.tf      DSN station frame definitions.
earth_000101_060525_060303.bpc Binary PCK for Earth.
naif0008.tls              Generic LSK.
ORMM__040501000000_00076XS.BSP MEX Orbiter trajectory SPK,
                          subsetting to cover only
                          time range of interest.
pck00008.tpc              Generic PCK.

\begindata

    KERNELS_TO_LOAD = (

        'kernels/spk/de405xs.bsp'
        'kernels/spk/earthstns_itrf93_050714.bsp'
        'kernels/fk/earth_topo_050714.tf'
        'kernels/pck/earth_000101_060525_060303.bpc'
        'kernels/lsk/naif0008.tls'
        'kernels/spk/ORMM__040501000000_00076XS.BSP'
        'kernels/pck/pck00008.tpc'
    )

\beginext

```

Solution Code

The example program below shows one possible solution.

```

#
# Solution viewpr
#
from __future__ import print_function
import spiceypy.utils.support_types as stypes
import spiceypy

def viewpr():
    #
    # Local Parameters
    #
    METAKR = 'viewpr.tm'
    TDBFMT = 'YYYY MON DD HR:MN:SC.### (TDB) ::TDB'
    MAXIVL = 1000
    MAXWIN = 2 * MAXIVL

    #
    # Load the meta-kernel.
    #
    spiceypy.furnsh( METAKR )

```

(continues on next page)

(continued from previous page)

```

#
# Assign the inputs for our search.
#
# Since we're interested in the apparent location of the
# target, we use light time and stellar aberration
# corrections. We use the "converged Newtonian" form
# of the light time correction because this choice may
# increase the accuracy of the occultation times we'll
# compute using gfoclt.
#
srfpt = 'DSS-14'
obsfrm = 'DSS-14_TOPO'
target = 'MEX'
abcorr = 'CN+S'
start = '2004 MAY 2 TDB'
stop = '2004 MAY 6 TDB'
elvlim = 6.0

#
# The elevation limit above has units of degrees; we convert
# this value to radians for computation using SPICE routines.
# We'll store the equivalent value in radians in revlim.
#
revlim = spiceypy.rpd() * elvlim

#
# Since SPICE doesn't directly support the AZ/EL coordinate
# system, we use the equivalent constraint
#
#     latitude > revlim
#
# in the latitudinal coordinate system, where the reference
# frame is topocentric and is centered at the viewing location.
#
crdsys = 'LATITUDINAL'
coord = 'LATITUDE'
relate = '>'

#
# The adjustment value only applies to absolute extrema
# searches; simply give it an initial value of zero
# for this inequality search.
#
adjust = 0.0

#
# stepsz is the step size, measured in seconds, used to search
# for times bracketing a state transition. Since we don't expect
# any events of interest to be shorter than five minutes, and
# since the separation between events is well over 5 minutes,
# we'll use this value as our step size. Units are seconds.
#
stepsz = 300.0

#
# Display a banner for the output report:
#

```

(continues on next page)

(continued from previous page)

```

print( '\n{:s}\n'.format(
    'Inputs for target visibility search:' ) )

print( '    Target                                = '
    '{:s}'.format( target ) )
print( '    Observation surface location = '
    '{:s}'.format( srftp ) )
print( '    Observer\'s reference frame   = '
    '{:s}'.format( obsfrm ) )
print( '    Elevation limit (degrees)     = '
    '{:f}'.format( elvlim ) )
print( '    Aberration correction           = '
    '{:s}'.format( abcorr ) )
print( '    Step size (seconds)            = '
    '{:f}'.format( stepsz ) )

#
# Convert the start and stop times to ET.
#
etbeg = spiceypy.str2et( start )
etend = spiceypy.str2et( stop )

#
# Display the search interval start and stop times
# using the format shown below.
#
#    2004 MAY 06 20:15:00.000 (TDB)
#
timstr = spiceypy.timeout( etbeg, TDBFMT )
print( '    Start time                    = '
    '{:s}'.format(timstr) )

timstr = spiceypy.timeout( etend, TDBFMT )
print( '    Stop time                        = '
    '{:s}'.format(timstr) )

print( ' ' )

#
# Initialize the "confinement" window with the interval
# over which we'll conduct the search.
#
cnfine = stypes.SPICEDOUBLE_CELL(2)
spiceypy.wninsd( etbeg, etend, cnfine )

#
# In the call below, the maximum number of window
# intervals gfposc can store internally is set to MAXIVL.
# We set the cell size to MAXWIN to achieve this.
#
riswin = stypes.SPICEDOUBLE_CELL( MAXWIN )

#
# Now search for the time period, within our confinement
# window, during which the apparent target has elevation
# at least equal to the elevation limit.
#

```

(continues on next page)

(continued from previous page)

```

spiceypy.gfposc( target, obsfrm, abcorr, srfpt,
                crdsys, coord, relate, revlim,
                adjust, stepsz, MAXIVL, cnfine, riswin )

#
# The function wncard returns the number of intervals
# in a SPICE window.
#
winsiz = spiceypy.wncard( riswin )

if winsiz == 0:

    print( 'No events were found.' )

else:

    #
    # Display the visibility time periods.
    #
    print( 'Visibility times of {0:s} '
          'as seen from {1:s}:\n'.format(
            target, srfpt )
          )

    for i in range(winsiz):
        #
        # Fetch the start and stop times of
        # the ith interval from the search result
        # window riswin.
        #
        [intbeg, intend] = spiceypy.wnfetd( riswin, i )

        #
        # Convert the rise time to a TDB calendar string.
        #
        timstr = spiceypy.timeout( intbeg, TDBFMT )

        #
        # Write the string to standard output.
        #
        if i == 0:

            print( 'Visibility or window start time:'
                  '{:s}'.format( timstr )
                  )

        else:

            print( 'Visibility start time:           '
                  '{:s}'.format( timstr )
                  )

        #
        # Convert the set time to a TDB calendar string.
        #
        timstr = spiceypy.timeout( intend, TDBFMT )

        #
        # Write the string to standard output.
        #
        if i == (winsiz-1):

```

(continues on next page)

(continued from previous page)

```

        print( 'Visibility or window stop time: '
              '   {:s}'.format( timstr )
              )
    else:

        print( 'Visibility stop time:           '
              '   {:s}'.format( timstr )
              )

    print( ' ' )

    spiceypy.unload( METAKR )

if __name__ == '__main__':
    viewpr()

```

Solution Sample Output

Numerical results shown for this example may differ across platforms since the results depend on the SPICE kernels used as input and on the host platform's arithmetic implementation.

Execute the program. The output is:

```

Inputs for target visibility search:

Target                = MEX
Observation surface location = DSS-14
Observer's reference frame = DSS-14_TOPO
Elevation limit (degrees)  = 6.000000
Aberration correction     = CN+S
Step size (seconds)       = 300.000000
Start time              = 2004 MAY 02 00:00:00.000 (TDB)
Stop time               = 2004 MAY 06 00:00:00.000 (TDB)

Visibility times of MEX as seen from DSS-14:

Visibility or window start time: 2004 MAY 02 00:00:00.000 (TDB)
Visibility stop time:             2004 MAY 02 05:35:03.096 (TDB)

Visibility start time:            2004 MAY 02 16:09:14.078 (TDB)
Visibility stop time:            2004 MAY 03 05:33:57.257 (TDB)

Visibility start time:            2004 MAY 03 16:08:02.279 (TDB)
Visibility stop time:            2004 MAY 04 05:32:50.765 (TDB)

Visibility start time:            2004 MAY 04 16:06:51.259 (TDB)
Visibility stop time:            2004 MAY 05 05:31:43.600 (TDB)

Visibility start time:            2004 MAY 05 16:05:40.994 (TDB)
Visibility or window stop time: 2004 MAY 06 00:00:00.000 (TDB)

```

6.3.10 Find Times when Target is Visible

6.3.11 Task Statement

Extend the program of the previous chapter to find times when the MEX orbiter is:

```
-- Above the elevation limit in the DSS-14_TOPO topocentric
   reference frame.

-- and is not occulted by Mars
```

Finding time intervals that satisfy the second condition requires a search for occultations of the spacecraft by Mars. Perform this search twice: once using an ellipsoidal shape model for Mars, and once using a DSK shape model.

Compute the final results twice as well, using the results of both occultation searches.

For each of the two shape model cases, store the set of time intervals when the spacecraft is visible in a SpiceyPy window. We'll call this the "result window."

Display each of the intervals in each result window as a pair of start and stop times. Express each time as a TDB calendar date using the same format as in the previous program.

6.3.12 Learning Goals

Familiarity with the GF occultation finding routine `spiceypy.gfoclt`. Experience with Digital Shape Kernel (DSK) shape models. Further experience with the SpiceyPy window functions.

6.3.13 Approach

Solution steps

A possible solution would consist of the following steps:

1. Use the meta-kernel **from the** previous chapter **as** the starting point. Add more kernels to it **as** needed.

Name the meta-kernel `'visibl.tm'`.
2. Include the code **from the** program of the previous chapter **in** a new source file; modify this code to create the new program.
3. Your program will need additional windows to capture the results of occultation searches performed using both ellipsoidal **and** DSK shape models. Additional windows will be needed to compute the **set** differences of the elevation search ("`view period`") window **and** each of the occultation search windows. Further details are provided below.

Create additional output SpiceyPy windows using `stypes.SPICEDOUBLE_CELL`.
4. The remaining steps can be performed twice: once using an ellipsoidal shape model **for** Mars, **and** once using a DSK Mars shape model. Alternatively, two copies of the entire solution program can be created: one **for** each shape model.

(continues on next page)

(continued from previous page)

5. Search **for** occultations of the MEX orbiter **as** seen **from** DSS-14 using `spiceypy.gfoclt`. Use **as** the confinement window **for** this search the result window **from the** elevation search performed by `spiceypy.gfposc`.

Since occultations occur when the apparent MEX spacecraft position **is** behind the apparent figure of Mars, light time correction must be performed **for** the occultation search. To improve accuracy of the occultation state determination, use `"converged Newtonian"` light time correction.
6. Use the SpiceyPy window subtraction routine `spiceypy.wndifd` to subtract the window of times when the spacecraft **is** occulted **from the** window of times when the spacecraft **is** above the elevation limit. The difference window **is** the final result.
7. Modify the code to display the contents of the difference window.

This completes the assignment.

6.3.14 Solution

Solution Meta-Kernel

The meta-kernel we created for the solution to this exercise is named 'visibl.tm'. Its contents follow:

KPL/MK

Example meta-kernel **for** geometric event finding hands-on coding lesson.

Version 3.0.0 26-OCT-2017 (BVS)

The names **and** contents of the kernels referenced by this meta-kernel are **as** follows:

File Name	Description
de405xs.bsp	Planetary ephemeris SPK, subsetting to cover only time range of interest.
earthstns_itrf93_050714.bsp	DSN station SPK.
earth_topo_050714.tf	DSN station frame definitions.
earth_000101_060525_060303.bpc	Binary PCK for Earth.
naif0008.tls	Generic LSK.
ORMM__040501000000_00076XS.BSP	MEX Orbiter trajectory SPK, subsetting to cover only time range of interest.
pck00008.tpc	Generic PCK.
mars_lowres.bds	Low-resolution Mars DSK.

\begindata

KERNELS_TO_LOAD = (

(continues on next page)

(continued from previous page)

```

'kernels/spk/de405xs.bsp'
'kernels/spk/earthstns_itrf93_050714.bsp'
'kernels/fk/earth_topo_050714.tf'
'kernels/pck/earth_000101_060525_060303.bpc'
'kernels/lsk/naif0008.tls'
'kernels/spk/ORMM__040501000000_00076XS.BSP'
'kernels/pck/pck00008.tpc'
'kernels/dsk/mars_lowres.bds'
    )

```

\begin{code}

Solution Code

```

#
# Solution visibl
#
from __future__ import print_function

#
# SpiceyPy package:
#
import spiceypy.utils.support_types as stypes
import spiceypy

def visibl():
    #
    # Local Parameters
    #
    METAKR = 'visibl.tm'
    SCLKID = -82
    TDBFMT = 'YYYY MON DD HR:MN:SC.### TDB ::TDB'
    MAXIVL = 1000
    MAXWIN = 2 * MAXIVL

    #
    # Load the meta-kernel.
    #
    spiceypy.furnsh( METAKR )

    #
    # Assign the inputs for our search.
    #
    # Since we're interested in the apparent location of the
    # target, we use light time and stellar aberration
    # corrections. We use the "converged Newtonian" form
    # of the light time correction because this choice may
    # increase the accuracy of the occultation times we'll
    # compute using gfoclt.
    #
    srfpt = 'DSS-14'
    obsfrm = 'DSS-14_TOPO'
    target = 'MEX'
    abcorr = 'CN+S'
    start = '2004 MAY 2 TDB'
    stop = '2004 MAY 6 TDB'

```

(continues on next page)

(continued from previous page)

```

elvlim = 6.0

#
# The elevation limit above has units of degrees; we convert
# this value to radians for computation using SPICE routines.
# We'll store the equivalent value in revlim.
#
revlim = spiceypy.rpd() * elvlim

#
# We model the target shape as a point. We either model the
# blocking body's shape as an ellipsoid, or we represent
# its shape using actual topographic data. No body-fixed
# reference frame is required for the target since its
# orientation is not used.
#
back = target
bshape = 'POINT'
bframe = ' '
front = 'MARS'
fshape = 'ELLIPSOID'
fframe = 'IAU_MARS'

#
# The occultation type should be set to 'ANY' for a point
# target.
#
occtyp = 'any'

#
# Since SPICE doesn't directly support the AZ/EL coordinate
# system, we use the equivalent constraint
#
# latitude > revlim
#
# in the latitudinal coordinate system, where the reference
# frame is topocentric and is centered at the viewing location.
#
crdsys = 'LATITUDINAL'
coord = 'LATITUDE'
relate = '>'

#
# The adjustment value only applies to absolute extrema
# searches; simply give it an initial value of zero
# for this inequality search.
#
adjust = 0.0

#
# stepsz is the step size, measured in seconds, used to search
# for times bracketing a state transition. Since we don't expect
# any events of interest to be shorter than five minutes, and
# since the separation between events is well over 5 minutes,
# we'll use this value as our step size. Units are seconds.
#
stepsz = 300.0

```

(continues on next page)

(continued from previous page)

```

#
# Display a banner for the output report:
#
print( '\n{:s}\n'.format(
    'Inputs for target visibility search:' ) )

print( '    Target                                = '
    '{:s}'.format( target ) )
print( '    Observation surface location = '
    '{:s}'.format( srfpt ) )
print( '    Observer\'s reference frame   = '
    '{:s}'.format( obsfrm ) )
print( '    Blocking body                  = '
    '{:s}'.format( front ) )
print( '    Blocker\'s reference frame     = '
    '{:s}'.format( fframe ) )
print( '    Elevation limit (degrees)      = '
    '{:f}'.format( elvlim ) )
print( '    Aberration correction           = '
    '{:s}'.format( abcorr ) )
print( '    Step size (seconds)            = '
    '{:f}'.format( stepsz ) )

#
# Convert the start and stop times to ET.
#
etbeg = spiceypy.str2et( start )
etend = spiceypy.str2et( stop )

#
# Display the search interval start and stop times
# using the format shown below.
#
#    2004 MAY 06 20:15:00.000 (TDB)
#
btmstr = spiceypy.timeout( etbeg, TDBFMT )
print( '    Start time                    = '
    '{:s}'.format( btmstr ) )

etmstr = spiceypy.timeout( etend, TDBFMT )
print( '    Stop time                      = '
    '{:s}'.format( etmstr ) )

print( ' ' )

#
# Initialize the "confinement" window with the interval
# over which we'll conduct the search.
#
cnfine = stypes.SPICEDOUBLE_CELL(2)
spiceypy.wninsd( etbeg, etend, cnfine )

#
# In the call below, the maximum number of window
# intervals gfposc can store internally is set to MAXIVL.
# We set the cell size to MAXWIN to achieve this.

```

(continues on next page)

(continued from previous page)

```

#
riswin = stypes.SPICEDOUBLE_CELL( MAXWIN )

#
# Now search for the time period, within our confinement
# window, during which the apparent target has elevation
# at least equal to the elevation limit.
#
spiceypy.gfposc( target, obsfrm, abcorr, srft,
                crdsys, coord, relate, revlim,
                adjust, stepsz, MAXIVL, cnfine, riswin )

#
# Now find the times when the apparent target is above
# the elevation limit and is not occulted by the
# blocking body (Mars). We'll find the window of times when
# the target is above the elevation limit and is occulted,
# then subtract that window from the view period window
# riswin found above.
#
# For this occultation search, we can use riswin as
# the confinement window because we're not interested in
# occultations that occur when the target is below the
# elevation limit.
#
# Find occultations within the view period window.
#
print( ' Searching using ellipsoid target shape model...' )

eocwin = stypes.SPICEDOUBLE_CELL( MAXWIN )

fshape = 'ELLIPSOID'

spiceypy.gfoclt( occtyp, front, fshape, fframe,
                back, bshape, bframe, abcorr,
                srft, stepsz, riswin, eocwin )
print( ' Done.' )

#
# Subtract the occultation window from the view period
# window: this yields the time periods when the target
# is visible.
#
evswin = spiceypy.wndifd( riswin, eocwin )

#
# Repeat the search using low-resolution DSK data
# for the front body.
#
print( ' Searching using DSK target shape model...' )

docwin = stypes.SPICEDOUBLE_CELL( MAXWIN )

fshape = 'DSK/UNPRIORITIZED'

spiceypy.gfoclt( occtyp, front, fshape, fframe,
                back, bshape, bframe, abcorr,

```

(continues on next page)

(continued from previous page)

```

        srfpt, stepsz, riswin, docwin )
print( ' Done.\n' )

dvswin = spiceypy.wndifd( riswin, docwin )

#
# The function wncard returns the number of intervals
# in a SPICE window.
#
winsiz = spiceypy.wncard( evswin )

if winsiz == 0:

    print( 'No events were found.' )

else:
    #
    # Display the visibility time periods.
    #
    print( 'Visibility start and stop times of '
           '{0:s} as seen from {1:s}\n'
           'using both ellipsoidal and DSK '
           'target shape models:\n'.format(
               target, srfpt )
           )

    for i in range(winsiz):
        #
        # Fetch the start and stop times of
        # the ith interval from the ellipsoid
        # search result window evswin.
        #
        [intbeg, intend] = spiceypy.wnfetd( evswin, i )

        #
        # Convert the rise time to TDB calendar strings.
        # Write the results.
        #
        btmstr = spiceypy.timeout( intbeg, TDBFMT )
        etmstr = spiceypy.timeout( intend, TDBFMT )

        print( ' Ell: {0:s} : {1:s}'.format( btmstr, etmstr ) )

        #
        # Fetch the start and stop times of
        # the ith interval from the DSK
        # search result window dvswin.
        #
        [dintbg, dinten] = spiceypy.wnfetd( dvswin, i )

        #
        # Convert the rise time to TDB calendar strings.
        # Write the results.
        #
        btmstr = spiceypy.timeout( dintbg, TDBFMT )
        etmstr = spiceypy.timeout( dinten, TDBFMT )

        print( ' DSK: {0:s} : {1:s}\n'.format( btmstr, etmstr ) )

```

(continues on next page)

(continued from previous page)

```

#
# End of result display loop.
#

spiceypy.unload( METAKR )

if __name__ == '__main__':
    visibl()

```

Solution Sample Output

Numerical results shown for this example may differ across platforms since the results depend on the SPICE kernels used as input and on the host platform's arithmetic implementation.

Execute the program. The output is:

```

Inputs for target visibility search:

Target                = MEX
Observation surface location = DSS-14
Observer's reference frame = DSS-14_TOPO
Blocking body         = MARS
Blocker's reference frame = IAU_MARS
Elevation limit (degrees) = 6.000000
Aberration correction   = CN+S
Step size (seconds)    = 300.000000
Start time             = 2004 MAY 02 00:00:00.000 TDB
Stop time              = 2004 MAY 06 00:00:00.000 TDB

Searching using ellipsoid target shape model...
Done.
Searching using DSK target shape model...
Done.

Visibility start and stop times of MEX as seen from DSS-14
using both ellipsoidal and DSK target shape models:

E11: 2004 MAY 02 00:00:00.000 TDB : 2004 MAY 02 04:49:30.827 TDB
DSK: 2004 MAY 02 00:00:00.000 TDB : 2004 MAY 02 04:49:32.645 TDB

E11: 2004 MAY 02 16:09:14.078 TDB : 2004 MAY 02 20:00:22.514 TDB
DSK: 2004 MAY 02 16:09:14.078 TDB : 2004 MAY 02 20:00:23.980 TDB

E11: 2004 MAY 02 21:01:38.222 TDB : 2004 MAY 03 03:35:42.256 TDB
DSK: 2004 MAY 02 21:01:43.195 TDB : 2004 MAY 03 03:35:44.140 TDB

E11: 2004 MAY 03 04:36:42.484 TDB : 2004 MAY 03 05:33:57.257 TDB
DSK: 2004 MAY 03 04:36:46.856 TDB : 2004 MAY 03 05:33:57.257 TDB

E11: 2004 MAY 03 16:08:02.279 TDB : 2004 MAY 03 18:46:26.013 TDB
DSK: 2004 MAY 03 16:08:02.279 TDB : 2004 MAY 03 18:46:27.306 TDB

E11: 2004 MAY 03 19:46:54.618 TDB : 2004 MAY 04 02:21:44.562 TDB
DSK: 2004 MAY 03 19:46:59.723 TDB : 2004 MAY 04 02:21:46.574 TDB

E11: 2004 MAY 04 03:21:56.347 TDB : 2004 MAY 04 05:32:50.765 TDB
DSK: 2004 MAY 04 03:22:00.850 TDB : 2004 MAY 04 05:32:50.765 TDB

```

(continues on next page)

(continued from previous page)

```

E11: 2004 MAY 04 16:06:51.259 TDB : 2004 MAY 04 17:32:25.809 TDB
DSK: 2004 MAY 04 16:06:51.259 TDB : 2004 MAY 04 17:32:27.118 TDB

E11: 2004 MAY 04 18:32:05.975 TDB : 2004 MAY 05 01:07:48.264 TDB
DSK: 2004 MAY 04 18:32:11.046 TDB : 2004 MAY 05 01:07:50.061 TDB

E11: 2004 MAY 05 02:07:11.601 TDB : 2004 MAY 05 05:31:43.600 TDB
DSK: 2004 MAY 05 02:07:16.241 TDB : 2004 MAY 05 05:31:43.600 TDB

E11: 2004 MAY 05 16:05:40.994 TDB : 2004 MAY 05 16:18:35.560 TDB
DSK: 2004 MAY 05 16:05:40.994 TDB : 2004 MAY 05 16:18:36.994 TDB

E11: 2004 MAY 05 17:17:27.717 TDB : 2004 MAY 05 23:54:04.672 TDB
DSK: 2004 MAY 05 17:17:32.375 TDB : 2004 MAY 05 23:54:06.221 TDB

```

6.3.15 Extra Credit

In this “extra credit” section you will be presented with more complex tasks, aimed at improving your understanding of the geometry event finding subsystem and particularly the `spiceypy.gfposc` and `spiceypy.gfdist` functions.

These “extra credit” tasks are provided as task statements, and unlike the regular tasks, no approach or solution source code is provided. In the next section, you will find the numeric solutions to the questions asked in these tasks.

6.3.16 Task statements

- Write a program that finds the times, within the time range

```

2004 MAY 2 TDB
2004 MAY 6 TDB

```

when the MEX spacecraft crosses Mars' equator. Display the results using TDB calendar dates **and** millisecond precision.
- Write a program that finds the times, within the time range

```

2004 MAY 2 TDB
2004 MAY 6 TDB

```

when the MEX spacecraft **is** at periapsis. Display the results using TDB calendar dates **and** millisecond precision.
- Write a program that finds the times, within the time range

```

2004 MAY 2 TDB
2004 MAY 6 TDB

```

when the MEX spacecraft **is** at apoapsis. Display the results using TDB calendar dates **and** millisecond precision.

6.3.17 Solutions

1. Solution **for** the equator crossing search, using `spiceypy.gfposc` **for** the MEX spacecraft latitude **in** the Mars body-fixed frame equal to 0 degrees:

Inputs **for** equator crossing search:

```

Target                = MEX
Observer              = MARS
Observer's reference frame = IAU_MARS
Latitude limit (degrees) = 0.000000
Aberration correction  = NONE
Step size (seconds)    = 300.000000
Start time            = 2004 MAY 02 00:00:00.000 (TDB)
Stop time             = 2004 MAY 06 00:00:00.000 (TDB)

```

MEX MARS equator crossing times:

```

Equator crossing or start time: 2004 MAY 02 05:00:08.334 (TDB)
Equator crossing time:          2004 MAY 02 06:15:13.074 (TDB)
Equator crossing time:          2004 MAY 02 12:35:14.856 (TDB)
Equator crossing time:          2004 MAY 02 13:50:09.161 (TDB)
Equator crossing time:          2004 MAY 02 20:10:24.439 (TDB)
Equator crossing time:          2004 MAY 02 21:25:10.344 (TDB)
Equator crossing time:          2004 MAY 03 03:45:26.758 (TDB)
Equator crossing time:          2004 MAY 03 05:00:04.086 (TDB)
Equator crossing time:          2004 MAY 03 11:20:32.419 (TDB)
Equator crossing time:          2004 MAY 03 12:34:57.968 (TDB)
Equator crossing time:          2004 MAY 03 18:55:34.883 (TDB)
Equator crossing time:          2004 MAY 03 20:09:53.063 (TDB)
Equator crossing time:          2004 MAY 04 02:30:35.509 (TDB)
Equator crossing time:          2004 MAY 04 03:44:42.753 (TDB)
Equator crossing time:          2004 MAY 04 10:05:41.638 (TDB)
Equator crossing time:          2004 MAY 04 11:19:38.397 (TDB)
Equator crossing time:          2004 MAY 04 17:40:41.405 (TDB)
Equator crossing time:          2004 MAY 04 18:54:31.413 (TDB)
Equator crossing time:          2004 MAY 05 01:15:45.967 (TDB)
Equator crossing time:          2004 MAY 05 02:29:25.294 (TDB)
Equator crossing time:          2004 MAY 05 08:50:53.931 (TDB)
Equator crossing time:          2004 MAY 05 10:04:26.915 (TDB)
Equator crossing time:          2004 MAY 05 16:25:58.350 (TDB)
Equator crossing or stop time: 2004 MAY 05 17:39:23.889 (TDB)

```

2. Solution **for** the periapsis search, using `spiceypy.gfdist` **for** the MEX spacecraft distance **from Mars** at a local minimum:

Inputs **for** periapsis search:

```

Target                = MEX
Observer              = MARS
Aberration correction  = NONE
Step size (seconds)    = 300.000000
Start time            = 2004 MAY 02 00:00:00.000 (TDB)
Stop time             = 2004 MAY 06 00:00:00.000 (TDB)

```

(continues on next page)

(continued from previous page)

MEX periapsis times:

```

Periapsis or start time:      2004 MAY 02 05:57:51.000 (TDB)
Periapsis time:              2004 MAY 02 13:32:43.325 (TDB)
Periapsis time:              2004 MAY 02 21:07:41.124 (TDB)
Periapsis time:              2004 MAY 03 04:42:30.648 (TDB)
Periapsis time:              2004 MAY 03 12:17:21.143 (TDB)
Periapsis time:              2004 MAY 03 19:52:12.267 (TDB)
Periapsis time:              2004 MAY 04 03:26:57.755 (TDB)
Periapsis time:              2004 MAY 04 11:01:49.826 (TDB)
Periapsis time:              2004 MAY 04 18:36:38.448 (TDB)
Periapsis time:              2004 MAY 05 02:11:28.558 (TDB)
Periapsis time:              2004 MAY 05 09:46:26.309 (TDB)
Periapsis or end time:        2004 MAY 05 17:21:18.875 (TDB)

```

3. Solution **for** the apoapsis search, using spiceypy.gfdist **for** the MEX spacecraft distance **from Mars** at a local maximum:

Inputs **for** apoapsis search:

```

Target          = MEX
Observer        = MARS
Aberration correction = NONE
Step size (seconds) = 300.000000
Start time      = 2004 MAY 02 00:00:00.000 (TDB)
Stop time       = 2004 MAY 06 00:00:00.000 (TDB)

```

MEX apoapsis times:

```

Apoapsis or start time:      2004 MAY 02 02:10:24.948 (TDB)
Apoapsis time:              2004 MAY 02 09:45:19.189 (TDB)
Apoapsis time:              2004 MAY 02 17:20:14.194 (TDB)
Apoapsis time:              2004 MAY 03 00:55:07.633 (TDB)
Apoapsis time:              2004 MAY 03 08:29:57.890 (TDB)
Apoapsis time:              2004 MAY 03 16:04:48.524 (TDB)
Apoapsis time:              2004 MAY 03 23:39:36.745 (TDB)
Apoapsis time:              2004 MAY 04 07:14:25.662 (TDB)
Apoapsis time:              2004 MAY 04 14:49:15.904 (TDB)
Apoapsis time:              2004 MAY 04 22:24:05.351 (TDB)
Apoapsis time:              2004 MAY 05 05:58:59.270 (TDB)
Apoapsis time:              2004 MAY 05 13:33:54.433 (TDB)
Apoapsis or stop time:       2004 MAY 05 21:08:50.211 (TDB)

```

6.4 In-situ Sensing Hands-On Lesson, using CASSINI (Python)

November 20, 2017

6.4.1 Overview

In this lesson you will develop a simple program illustrating how SPICE can be used to compute various kinds of geometry information applicable to the experiments carried out by an in-situ instrument flown on an interplanetary spacecraft.

6.4.2 References

This section lists SPICE documents referred to in this lesson.

In some cases the lesson explanations also refer to the information provided in the meta-data area of the kernels used in the lesson examples. It is especially true in case of the FK and IK files, which often contain comprehensive descriptions of the frames, instrument FOVs, etc. Since both FK and IK are text kernels, the information provided in them can be viewed using any text editor, while the meta information provided in binary kernels – SPKs and CKs – can be viewed using “commnt” or “spacit” utility programs located in “cspice/exe” of Toolkit installation tree.

The following SPICE tutorials serve as references for the discussions in this lesson:

Name	Lesson steps/functions it describes
-----	-----
Time	UTC to ET and SCLK to ET
Loading Kernels	Loading SPICE kernels
SCLK	SCLK to ET time conversion
SPK	Computing positions and velocities
Frames	Computing transformations between frames

These tutorials are available from the NAIF ftp server at JPL:

<http://naif.jpl.nasa.gov/naif/tutorials.html>

Required Readings

The Required Reading documents are provided with the Toolkit and are located under the “cspice/doc” directory in the CSPICE Toolkit installation tree.

Name	Lesson steps/functions that it describes
-----	-----
kernel.req	Loading SPICE kernels
naif_ids.req	Body and reference frame names
spk.req	Computing positions and velocities
sclk.req	SCLK to ET time conversion
time.req	UTC to ET time conversion

The Permuted Index

Another useful document distributed with the Toolkit is the permuted index. This is located under the “cspice/doc” directory in the C installation tree.

This text document provides a simple mechanism by which users can discover which SpiceyPy functions perform functions of interest, as well as the names of the source files that contain these functions.

SpiceyPy API Documentation

A SpiceyPy function's parameters specification is available using the built-in Python help system.

For example, the Python help function

```
>>> import spiceypy
>>> help(spiceypy.str2et)
```

describes of the str2et function's parameters, while the document

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/str2et_c.html

describes extensively the str2et functionality.

6.4.3 Kernels Used

The following kernels are used in examples provided in this lesson:

#	FILE NAME	TYPE	DESCRIPTION
1	naif0008.tls	LSK	Generic LSK
2	cas00084.tsc	SCLK	Cassini SCLK
3	020514_SE_SAT105.bsp	SPK	Saturnian Satellite Ephemeris SPK
4	030201AP_SK_SM546_T45.bsp	SPK	Cassini Spacecraft SPK
5	981005_PLTEPH-DE405S.bsp	SPK	Planetary Ephemeris SPK
6	sat128.bsp	SPK	Saturnian Satellite Ephemeris SPK
7	04135_04171pc_psiv2.bc	CK	Cassini Spacecraft CK
8	cas_v37.tf	FK	Cassini FK
9	cpck05Mar2004.tpc	PCK	Cassini project PCK

These SPICE kernels are included in the lesson package available from the NAIF server at JPL:

ftp://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Lessons/

6.4.4 SpiceyPy Modules Used

This section provides a complete list of the functions and kernels that are suggested for usage in each of the exercises in this lesson. (You may wish to not look at this list unless/until you “get stuck” while working on your own.)

CHAPTER	EXERCISE	FUNCTIONS	NON-VOID	KERNELS
1	convrt	spiceypy.furnsh spiceypy.unload	spiceypy.str2et	1
2	sclket	spiceypy.furnsh spiceypy.unload	spiceypy.str2et spiceypy.scs2e	1,2
3	getsta	spiceypy.furnsh spiceypy.unload	spiceypy.str2et spiceypy.scs2e spiceypy.spkezr	1-6
4	soldir	spiceypy.furnsh spiceypy.unload	spiceypy.str2et spiceypy.scs2e spiceypy.spkezr spiceypy.spkpos spiceypy.vhat	1-8

(continues on next page)

(continued from previous page)

5	sscpnt	spiceypy.furnsh	spiceypy.str2et	1-9
		spiceypy.unload	spiceypy.scs2e spiceypy.spkezr spiceypy.spkpos spiceypy.vhat spiceypy.subpnt spiceypy.reclat spiceypy.pxform spiceypy.mxv spiceypy.dpr	
6	scvel	spiceypy.furnsh	spiceypy.str2et	1-9
		spiceypy.unload	spiceypy.scs2e spiceypy.spkezr spiceypy.spkpos spiceypy.vhat spiceypy.subpnt spiceypy.reclat spiceypy.pxform spiceypy.mxv spiceypy.dpr	

Use the Python built-in help system on the various functions listed above for the API parameters’ description, and refer to the headers of their corresponding CSPICE versions for detailed interface specifications.

6.4.5 Step-1: “UTC to ET”

6.4.6 “UTC to ET” Task Statement

Write a program that computes and prints the Ephemeris Time (ET) corresponding to “2004-06-11T19:32:00” UTC, as the number of ephemeris seconds past J2000, .

6.4.7 “UTC to ET” Hints

Find out what SPICE kernel(s) is(are) needed to support this conversion. Reference the “time.req” and/or” Time” tutorial.

Find necessary kernel(s) on the NAIF’s FTP site.

Find out what routine should be called to load necessary kernel(s). Reference the “kernel.req” and/or” Loading Kernels” tutorial.

Find the “loader” routine calling sequence specification. Look at the” time.req”and that routine’s source code header. This routine may be an entry point, in which case there will be no source file with the same name. To find out in which source file this entry point is, search for its name in the “Permuted Index”.

Find the routine(s) used to convert time between UTC and ET. Look at the “time.req” and/or” Time” tutorial.

Find the “converter” routine(s) calling sequence specification. Look in the” time.req” and the routine’s source code header.

Put all calls together in a program, add variable declarations (the routine header’s “Declarations” and” Examples” sections are a good place to look for declaration specification and examples) and output print statements.

6.4.8 “UTC to ET” Solution Steps

Only one kernel file is needed to support this conversion – an LSK file “naif0008.tls”.

As any other SPICE kernel this file can be loaded by the `spiceypy.furnsh` function. For that, the name of the file can be provided as a sole argument of this routine:

```
...
lskfile = 'naif0008.tls'

spiceypy.furnsh(lskfile)
```

or it can be listed in a meta-kernel:

```
KPL/MK

The names and contents of the kernels referenced by this
meta-kernel are as follows:

File Name              Description
-----
naif0008.tls           Generic LSK.

\begindata
  KERNELS_TO_LOAD = (
                      'kernels/lsk/naif0008.tls'
                      )
\beginntext
```

the name of which, let’s call it “convrt.tm”, can be then provided as a sole argument of the `spiceypy.spiceypy.furnsh()` routine:

```
mkfile = 'convrt.tm'
spiceypy.furnsh(mkfile)
```

While the second option seems to involve a bit more work – it requires making an extra file – it is a much better way to go if you plan to load more kernels as you extend the program. With the meta-kernel approach simply adding more kernels to the list in `KERNEL_TO_LOAD` without changing the program code will accomplish that.

The highest level SpiceyPy time routine converting UTC to ET is `spiceypy.str2et` *spiceypy.spiceypy.str2et()*.

It has two arguments – input time string representing UTC in a variety of formats (see `spiceypy.str2et` header’s section “Particulars” for the complete description of input time formats) and output DP number of ET seconds past J2000. A call to `spiceypy.str2et` converting a given UTC to ET could look like this:

```
utc = '2004-06-11T19:32:00'
et = spiceypy.str2et(utc)
```

By combining `spiceypy.furnsh` and `spiceypy.str2et` calls and required declarations and by adding a simple print statement, one would get a complete program that prints ET for the given UTC epoch.

Use of SpiceyPy calls in a Python script requires the SpiceyPy package to be installed in your Python distribution, either using `pip` or `conda`, and imported within the script.

When you execute the script, “convrt”, it produces the following output:

```
> python convrt.py
UTC      = 2004-06-11T19:32:00
ET       = 140254384.184625
```

6.4.9 “UTC to ET” Code

Program “convrt.py”:

```
from __future__ import print_function
import spiceypy

def convrt():

    mkfile = 'convrt.tm'
    spiceypy.furnsh(mkfile)

    utc = '2004-06-11T19:32:00'
    et = spiceypy.str2et(utc)

    print('UTC      = {:s}'.format(utc))
    print('ET       = {:20.6f}'.format(et))

    spiceypy.unload(mkfile)

if __name__ == '__main__':
    convrt()
```

Meta-kernel file “convrt.tm”:

```
KPL/MK

The names and contents of the kernels referenced by this
meta-kernel are as follows:

File Name              Description
-----
naif0008.tls           Generic LSK.

\begindata
  KERNELS_TO_LOAD = (
    'kernels/lsk/naif0008.tls'
  )
\beginext
```

6.4.10 Step-2: “SCLK to ET”

6.4.11 “SCLK to ET” Task Statement

Extend the program from Step-1 to compute and print ET for the following CASSINI on-board clock epoch “1465674964.105”.

6.4.12 “SCLK to ET” Hints

Find out what additional (to those already loaded in Step-1) SPICE kernel(s) is(are) needed to support SCLK to ET conversion. Look at the “sclk.req” and/or “SCLK” tutorial.

Find necessary kernel(s) on the NAIF’s FTP site.

Modify the program or meta-kernel to load this(these) kernels.

Find the routine(s) needed to convert time between SCLK and ET. Look at the “sclk.req” and/or “Time” and “SCLK” tutorials.

Find the “converter” routine’s calling sequence specification. Look in the “sclk.req” and the routine’s source code header.

Look at “naif_ids.req” and the comments in the additional kernel(s) that you have loaded for information on proper values of input arguments of this routine.

Add calls to the “converter” routine(s), necessary variable declarations (the routine header’s “Declarations” and “Examples” sections are a good place to look for declaration specification and examples), and output print statements to the program.

6.4.13 “SCLK to ET” Solution Steps

A CASSINI SCLK file is needed additionally to the LSK file loaded in the Step-1 to support this conversion.

No code change is needed in the loading portion of the program if a meta-kernel approach was used in the Step-1. The program will load the file if it will be added to the list of kernels in the KERNELS_TO_LOAD variable:

```
KPL/MK

The names and contents of the kernels referenced by this
meta-kernel are as follows:

File Name                Description
-----
naif0008.tls             Generic LSK.
cas00084.tsc             Cassini SCLK.

\begindata
  KERNELS_TO_LOAD = (
    'kernels/lsk/naif0008.tls'
    'kernels/sclk/cas00084.tsc'
  )
\begintext
```

The highest level SpiceyPy routine converting SCLK to ET is `spiceypy.scs2e` *spiceypy.scs2e()*.

It has three arguments – NAIF ID for CASSINI s/c (-82 as described by “naif_ids.req” document), input time string representing CASSINI SCLK, and output DP number of ET seconds past J2000. A call to `spiceypy.str2et` converting given SCLK to ET could look like this:

```
scid = -82
sclk = '1465674964.105'
et = spiceypy.scs2e(scid, sclk)
```

By adding the `spiceypy.scs2e` call, required declarations and a simple print statement, one would get a complete program that prints ET for the given SCLK epoch.

When you execute the script, “`sclket`”, it produces the following output:

```
> python convrt.py
UTC      = 2004-06-11T19:32:00
ET       = 140254384.184625
SCLK     = 1465674964.105
ET       = 140254384.183426
```

6.4.14 “SCLK to ET” Code

Program “`sclket.py`”:

```
from __future__ import print_function
import spiceypy

def sclket():

    mkfile = 'sclket.tm'
    spiceypy.furnsh(mkfile)

    utc = '2004-06-11T19:32:00'
    et = spiceypy.str2et(utc)

    print('UTC      = {:s}'.format(utc))
    print('ET       = {:20.6f}'.format(et))

    scid = -82
    sclk = '1465674964.105'
    et = spiceypy.scs2e(scid, sclk)

    print('SCLK     = {:s}'.format(sclk))
    print('ET       = {:20.6f}'.format(et))

    spiceypy.unload(mkfile)

if __name__ == '__main__':
    sclket()
```

Meta-kernel file “`sclket.tm`”:

KPL/MK

The names and contents of the kernels referenced by this meta-kernel are as follows:

(continues on next page)

(continued from previous page)

File Name	Description
naif0008.tls	Generic LSK.
cas00084.tsc	Cassini SCLK.

```

\begindata
  KERNELS_TO_LOAD = (
    'kernels/lsk/naif0008.tls'
    'kernels/sclk/cas00084.tsc'
  )
\beginext

```

6.4.15 Step-3: “Spacecraft State”

6.4.16 “Spacecraft State” Task Statement

Extend the program from Step-2 to compute geometric state – position and velocity – of the CASSINI spacecraft with respect to the Sun in the Ecliptic frame at the epoch specified by SCLK time from Step-2.

6.4.17 “Spacecraft State” Hints

Find out what additional (to those already loaded in Steps-1&2) SPICE kernel(s) is(are) needed to support state computation. Look at the “spk.req” and/or” SPK” tutorial.

Find necessary kernel(s) on the NAIF’s FTP site.

Verify that the kernels contain enough data to compute the state of interest. Use “brief” utility program located under” toolkit/exe” directory for that.

Modify the meta-kernel to load this(these) kernels.

Determine the routine(s) needed to compute states. Look at the “spk.req” and/or” SPK” tutorial presentation.

Find the the routine(s) calling sequence specification. Look in the “spk.req” and the routine’s source code header.

Reference the “naif_ids.req” and” frames.req”and the routine(s) header “Inputs” and” Particulars” sections to determine proper values of the input arguments of this routine.

Add calls to the routine(s), necessary variable declarations and output print statements to the program.

6.4.18 “Spacecraft State” Solution Steps

A CASSINI spacecraft trajectory SPK and generic planetary ephemeris SPK files are needed to support computation of the state of interest.

The file names can be added to the meta-kernel to get them loaded into the program:

```

KPL/MK

The names and contents of the kernels referenced by this
meta-kernel are as follows:

```

(continues on next page)

(continued from previous page)

File Name	Description
naif0008.tls	Generic LSK.
cas00084.tsc	Cassini SCLK.
020514_SE_SAT105.bsp	Saturnian Satellite Ephemeris SPK.
030201AP_SK_SM546_T45.bsp	Cassini Spacecraft SPK.
981005_PLTEPH-DE405S.bsp	Planetary Ephemeris SPK.
sat128.bsp	Saturnian Satellite Ephemeris SPK.

```

\begindata
  KERNELS_TO_LOAD = (
    'kernels/lsk/naif0008.tls'
    'kernels/sclk/cas00084.tsc'
    'kernels/spk/020514_SE_SAT105.bsp'
    'kernels/spk/030201AP_SK_SM546_T45.bsp'
    'kernels/spk/981005_PLTEPH-DE405S.bsp'
    'kernels/spk/sat128.bsp'
  )
\beginext

```

The highest level SpiceyPy routine computing states is `spiceypy.spkezr` *spiceypy.spiceypy.spkezr()*.

We are interested in computing CASSINI position and velocity with respect to the Sun, therefore the target and observer names should be set to 'CASSINI' and 'Sun' (both names can be found in "naif_ids.req").

The state should be in ecliptic frame, therefore the name of the frame in which the state should be computed is 'ECLIPJ2000' (see "frames.req" document.)

Since we need only the geometric position, the 'abcorr' argument of the routine should be set to 'NONE' (see aberration correction discussion in the *spiceypy.spiceypy.spkezr()*).

Putting it all together, we get:

```

target = 'CASSINI'
frame  = 'ECLIPJ2000'
corrtn = 'NONE'
observ = 'SUN'

state, ltime = spiceypy.spkezr(target, et, frame,
                               corrtn, observ)

```

When you execute the script, "getsta", it produces the following output:

```

> python getsta.py
UTC      = 2004-06-11T19:32:00
ET       = 140254384.184625
SCLK     = 1465674964.105
ET       = 140254384.183426
X        = -376599061.916539
Y        = 1294487780.929154
Z        = -7064853.054698
VX       = -5.164226
VY       = 0.801719
VZ       = 0.040603

```

6.4.19 “Spacecraft State” Code

Program “getsta.py”:

```
from __future__ import print_function
import spiceypy

def getsta():

    mkfile = 'getsta.tm'
    spiceypy.furnsh(mkfile)

    utc = '2004-06-11T19:32:00'
    et = spiceypy.str2et(utc)

    print('UTC      = {:s}'.format(utc))
    print('ET       = {:20.6f}'.format(et))

    scid = -82
    sclk = '1465674964.105'
    et = spiceypy.scs2e(scid, sclk)

    print('SCLK      = {:s}'.format(sclk))
    print('ET       = {:20.6f}'.format(et))

    target = 'CASSINI'
    frame = 'ECLIPJ2000'
    corrtm = 'NONE'
    observ = 'SUN'

    state, ltime = spiceypy.spkezr(target, et, frame,
                                   corrtm, observ)

    print(' X      = {:20.6f}'.format(state[0]))
    print(' Y      = {:20.6f}'.format(state[1]))
    print(' Z      = {:20.6f}'.format(state[2]))
    print('VX      = {:20.6f}'.format(state[3]))
    print('VY      = {:20.6f}'.format(state[4]))
    print('VZ      = {:20.6f}'.format(state[5]))

    spiceypy.unload(mkfile)

if __name__ == '__main__':
    getsta()
```

Meta-kernel file “getsta.tm”:

KPL/MK

The names and contents of the kernels referenced by this meta-kernel are as follows:

File Name	Description
naif0008.tls	Generic LSK.
cas00084.tsc	Cassini SCLK.

(continues on next page)

(continued from previous page)

```

020514_SE_SAT105.bsp      Saturnian Satellite Ephemeris SPK.
030201AP_SK_SM546_T45.bsp Cassini Spacecraft SPK.
981005_PLTEPH-DE405S.bsp  Planetary Ephemeris SPK.
sat128.bsp                Saturnian Satellite Ephemeris SPK.

\begindata
  KERNELS_TO_LOAD = (
    'kernels/lsk/naif0008.tls'
    'kernels/sclk/cas00084.tsc'
    'kernels/spk/020514_SE_SAT105.bsp'
    'kernels/spk/030201AP_SK_SM546_T45.bsp'
    'kernels/spk/981005_PLTEPH-DE405S.bsp'
    'kernels/spk/sat128.bsp'
  )
\beginntext

```

6.4.20 Step-4: “Sun Direction”

6.4.21 “Sun Direction” Task Statement

Extend the program from Step-3 to compute apparent direction of the Sun in the INMS frame at the epoch specified by SCLK time from Step-2.

6.4.22 “Sun Direction” Hints

Determine the additional SPICE kernels needed to support the direction computation, knowing that they should provide the s/c and instrument frame orientation. Retrieve these kernels from the NAIF’s FTP site.

Verify that the orientation data in the kernels have adequate coverage to support computation of the direction of interest. Use “ckbrief” utility program located under “toolkit/exe” directory for that.

Modify the meta-kernel to load this(these) kernels.

Determine the proper input arguments for the `spiceypy.spkpos` call to calculate the direction (which is the position portion of the output state). Look through the Frames Kernel find the name of the frame to used.

Add calls to the routine(s), necessary variable declarations and output print statements to the program.

6.4.23 “Sun Direction” Solution Steps

A CASSINI spacecraft orientation CK file, providing s/c orientation with respect to an inertial frame, and CASSINI FK file, providing orientation of the INMS frame with respect to the s/c frame, are needed additionally to already loaded kernels to support computation of this direction.

The file names can be added to the meta-kernel to get them loaded into the program:

```

KPL/MK

The names and contents of the kernels referenced by this
meta-kernel are as follows:

File Name                Description

```

(continues on next page)

(continued from previous page)

```

-----
naif0008.tls          Generic LSK.
cas00084.tsc          Cassini SCLK.
020514_SE_SAT105.bsp  Saturnian Satellite Ephemeris SPK.
030201AP_SK_SM546_T45.bsp Cassini Spacecraft SPK.
981005_PLTEPH-DE405S.bsp Planetary Ephemeris SPK.
sat128.bsp            Saturnian Satellite Ephemeris SPK.
04135_04171pc_psiv2.bc Cassini Spacecraft CK.
cas_v37.tf            Cassini FK.

\begindata
  KERNELS_TO_LOAD = (
    'kernels/lsk/naif0008.tls'
    'kernels/sclk/cas00084.tsc'
    'kernels/spk/020514_SE_SAT105.bsp'
    'kernels/spk/030201AP_SK_SM546_T45.bsp'
    'kernels/spk/981005_PLTEPH-DE405S.bsp'
    'kernels/spk/sat128.bsp'
    'kernels/ck/04135_04171pc_psiv2.bc'
    'kernels/fk/cas_v37.tf'
  )
\beginntext

```

The same highest level SpiceyPy routine computing positions, `spiceypy.spkpos`, can be used to compute this direction.

Since this is the direction of the Sun as seen from the s/c, the target argument should be set to 'Sun' and the observer argument should be set to 'CASSINI'. The name of the INMS frame is 'CASSINI_INMS', the definition and description of this frame are provided in the CASSINI FK file, "cassini_v02.tf".

Since the apparent, or as seen", position is sought for, the 'abcorr' argument of the routine should be set to 'LT+S' (see aberration correction discussion in the (cspice/src/cspice/spkpos.c"))

If desired, the position can then be turned into a unit vector using `spiceypy.vhat` function (<https://spiceypy.readthedocs.io/en/master/documentation.html#spiceypy.vhat>) Putting it all together, we get:

```

target = 'SUN'
frame = 'CASSINI_INMS'
corrtn = 'LT+S'
observ = 'CASSINI'

sundir, ltime = spiceypy.spkpos(target, et, frame,
                                corrtn, observ)
sundir = spiceypy.vhat(sundir)

```

When you execute the script, "soldir", it produces the following output:

```

> python soldir.py
UTC      = 2004-06-11T19:32:00
ET       = 140254384.184625
SCLK     = 1465674964.105
ET       = 140254384.183426
X        = -376599061.916539
Y        = 1294487780.929154
Z        = -7064853.054698
VX       = -5.164226

```

(continues on next page)

(continued from previous page)

```

VY          =          0.801719
VZ          =          0.040603
SUNDIR(X)   =         -0.290204
SUNDIR(Y)   =          0.881631
SUNDIR(Z)   =          0.372167

```

6.4.24 “Sun Direction” Code

Program “soldir.py”:

```

from __future__ import print_function
import spiceypy

def soldir():

    mkfile = 'soldir.tm'
    spiceypy.furnsh(mkfile)

    utc = '2004-06-11T19:32:00'
    et = spiceypy.str2et(utc)

    print('UTC          = {:s}'.format(utc))
    print('ET           = {:20.6f}'.format(et))

    scid = -82
    sclk = '1465674964.105'
    et = spiceypy.scs2e(scid, sclk)

    print('SCLK          = {:s}'.format(sclk))
    print('ET           = {:20.6f}'.format(et))

    target = 'CASSINI'
    frame = 'ECLIPJ2000'
    corrtm = 'NONE'
    observ = 'SUN'

    state, ltime = spiceypy.spkezr(target, et, frame,
                                   corrtm, observ)

    print(' X          = {:20.6f}'.format(state[0]))
    print(' Y          = {:20.6f}'.format(state[1]))
    print(' Z          = {:20.6f}'.format(state[2]))
    print('VX          = {:20.6f}'.format(state[3]))
    print('VY          = {:20.6f}'.format(state[4]))
    print('VZ          = {:20.6f}'.format(state[5]))

    target = 'SUN'
    frame = 'CASSINI_INMS'
    corrtm = 'LT+S'
    observ = 'CASSINI'

    sundir, ltime = spiceypy.spkpos(target, et, frame,
                                    corrtm, observ)

    sundir = spiceypy.vhat(sundir)

```

(continues on next page)

(continued from previous page)

```

print('SUNDIR(X) = {:20.6f}'.format(sundir[0]))
print('SUNDIR(Y) = {:20.6f}'.format(sundir[1]))
print('SUNDIR(Z) = {:20.6f}'.format(sundir[2]))

spiceypy.unload(mkfile)

if __name__ == '__main__':
    soldir()
    
```

Meta-kernel file “soldir.tm”:

KPL/MK

The names and contents of the kernels referenced by this meta-kernel are as follows:

File Name	Description
-----	-----
naif0008.tls	Generic LSK.
cas00084.tsc	Cassini SCLK.
020514_SE_SAT105.bsp	Saturnian Satellite Ephemeris SPK.
030201AP_SK_SM546_T45.bsp	Cassini Spacecraft SPK.
981005_PLTEPH-DE405S.bsp	Planetary Ephemeris SPK.
sat128.bsp	Saturnian Satellite Ephemeris SPK.
04135_04171pc_psiv2.bc	Cassini Spacecraft CK.
cas_v37.tf	Cassini FK.

```

\begindata
  KERNELS_TO_LOAD = (
    'kernels/lsk/naif0008.tls'
    'kernels/sclk/cas00084.tsc'
    'kernels/spk/020514_SE_SAT105.bsp'
    'kernels/spk/030201AP_SK_SM546_T45.bsp'
    'kernels/spk/981005_PLTEPH-DE405S.bsp'
    'kernels/spk/sat128.bsp'
    'kernels/ck/04135_04171pc_psiv2.bc'
    'kernels/fk/cas_v37.tf'
  )
\beginext
    
```

6.4.25 Step-5: “Sub-Spacecraft Point”

6.4.26 “Sub-Spacecraft Point” Task Statement

Extend the program from Step-4 to compute planetocentric longitude and latitude of the sub-spacecraft point on Phoebe, and the direction from the spacecraft to that point in the INMS frame.

6.4.27 “Sub-Spacecraft Point” Hints

Find the SpiceyPy routine that computes sub-observer point coordinates. Use “Most Used SpiceyPy APIs” or “subpt” cookbook program for that.

Refer to the routine’s header to determine the additional kernels needed for this direction computation. Get these kernels from the NAIF’s FTP site. Modify the meta-kernel to load this(these) kernels.

Determine the proper input arguments for the routine. Refer to the routine’s header for that information.

Convert the surface point Cartesian vector returned by this routine to latitudinal coordinates. Use “Permuted Index” to find the routine that does this conversion. Refer to the routine’s header for input/output argument specifications.

Since the Cartesian vector from the spacecraft to the sub-spacecraft point is computed in the Phoebe body-fixed frame, it should be transformed into the instrument frame get the direction we are looking for. Refer to “frames.req” and/or “Frames” tutorial to determine the name of the routine computing transformations and use it to compute transformation from Phoebe body-fixed to the INMS frame.

Using “Permuted Index” find the routine that multiplies 3x3 matrix by 3d vector and use it to rotate the vector to the instrument frame.

Add calls to the routine(s), necessary variable declarations and output print statements to the program.

6.4.28 “Sub-Spacecraft Point” Solution Steps

The `spiceypy.subpnt` routine (“`cspice/src/cspice/subpnt_c.c`”) can be used to compute the sub-observer point and the vector from the observer to that point with a single call. To determine this point as the closest point on the Phoebe ellipsoid, the “method” argument has to be set to ‘NEAR POINT: ELLIPSOID’. For our case the ‘target’ is ‘PHOEBE’, the target body-fixed frame is ‘IAU_PHOEBE’, and the observer is ‘CASSINI’.

Since the s/c is close to Phoebe, light time does not need to be taken into account and, therefore, the ‘abcorr’ argument can be set to ‘NONE’.

In order for `spiceypy.subpnt` to compute the nearest point location, a PCK file containing Phoebe radii has to be loaded into the program (see “Files” section of the routine’s header.) All other files required for this computation are already being loaded by the program. With PCK file name added to it, the updated meta-kernel will look like this:

KPL/MK

The names and contents of the kernels referenced by this meta-kernel are as follows:

File Name	Description
-----	-----
naif0008.tls	Generic LSK.
cas00084.tsc	Cassini SCLK.
020514_SE_SAT105.bsp	Saturnian Satellite Ephemeris SPK.
030201AP_SK_SM546_T45.bsp	Cassini Spacecraft SPK.

(continues on next page)

(continued from previous page)

```

981005_PLTEPH-DE405S.bsp      Planetary Ephemeris SPK.
sat128.bsp                   Saturnian Satellite Ephemeris SPK.
04135_04171pc_psiv2.bc       Cassini Spacecraft CK.
cas_v37.tf                   Cassini FK.
cpck05Mar2004.tpc            Cassini project PCK.

\begindata
  KERNELS_TO_LOAD = (
    'kernels/lsk/naif0008.tls'
    'kernels/sclk/cas00084.tsc'
    'kernels/spk/020514_SE_SAT105.bsp'
    'kernels/spk/030201AP_SK_SM546_T45.bsp'
    'kernels/spk/981005_PLTEPH-DE405S.bsp'
    'kernels/spk/sat128.bsp'
    'kernels/ck/04135_04171pc_psiv2.bc'
    'kernels/fk/cas_v37.tf'
    'kernels/pck/cpck05Mar2004.tpc'
  )
\beginntext

```

The sub-spacecraft point Cartesian vector can be converted to planetocentric radius, longitude and latitude using the `spiceypy.reclat` routine `spiceypy.spiceypy.reclat()`.

The vector from the spacecraft to the sub-spacecraft point returned by `spiceypy.subpnt` has to be rotated from the body-fixed frame to the instrument frame. The name of the routine that computes 3x3 matrices rotating vectors from one frame to another is `spiceypy.pxform` `spiceypy.spiceypy.pxform()`.

In our case the “from” argument should be set to ‘IAU_PHOEBE’ and the “to” argument should be set to ‘CASSINI_INMS’

The vector should be then multiplied by this matrix to rotate it to the instrument frame. The `spiceypy.m xv` routine performs that function `spiceypy.spiceypy.m xv()`.

After applying the rotation, normalize the resultant vector using the `spiceypy.vhat` function.

For output the longitude and latitude angles returned by `spiceypy.reclat` in radians can be converted to degrees by multiplying by `spiceypy.dpr` function `spiceypy.spiceypy.dpr()`.

Putting it all together, we get:

```

method = 'NEAR POINT: ELLIPSOID'
target = 'PHOEBE'
frame = 'IAU_PHOEBE'
corrtn = 'NONE'
observ = 'CASSINI'

spoint, trgepc, srfvec = spiceypy.subpnt(method, target, et,
                                         frame, corrtn, observ)

srad, slon, slat = spiceypy.reclat(spoint)

fromfr = 'IAU_PHOEBE'
tofr = 'CASSINI_INMS'

m2imat = spiceypy.pxform(fromfr, tofr, et)

sbpdir = spiceypy.m xv(m2imat, srfvec)

```

(continues on next page)

(continued from previous page)

```

sbpdir = spiceypy.vhat(sbpdir)

print('LON      = {:.20.6f}'.format(slon * spiceypy.dpr()))
print('LAT      = {:.20.6f}'.format(slat * spiceypy.dpr()))

```

When you execute the script, “sscpnt”, it produces the following output:

```

> python sscpnt.py
UTC      = 2004-06-11T19:32:00
ET       = 140254384.184625
SCLK     = 1465674964.105
ET       = 140254384.183426
X        = -376599061.916539
Y        = 1294487780.929154
Z        = -7064853.054698
VX       = -5.164226
VY       = 0.801719
VZ       = 0.040603
SUNDIR(X) = -0.290204
SUNDIR(Y) = 0.881631
SUNDIR(Z) = 0.372167
LON      = 23.423158
LAT      = 3.709797
SBPDIR(X) = -0.000776
SBPDIR(Y) = -0.999873
SBPDIR(Z) = -0.015905

```

6.4.29 “Sub-Spacecraft Point” Code

Program

```

from __future__ import print_function
import spiceypy

def sscpnt():

    mkfile = 'sscpnt.tm'
    spiceypy.furnsh(mkfile)

    utc = '2004-06-11T19:32:00'
    et = spiceypy.str2et(utc)

    print('UTC      = {:s}'.format(utc))
    print('ET       = {:.20.6f}'.format(et))

    scid = -82
    sclk = '1465674964.105'
    et = spiceypy.scs2e(scid, sclk)

    print('SCLK     = {:s}'.format(sclk))
    print('ET       = {:.20.6f}'.format(et))

    target = 'CASSINI'
    frame = 'ECLIPJ2000'
    corrtm = 'NONE'

```

(continues on next page)

(continued from previous page)

```

observ = 'SUN'

state, ltime = spiceypy.spkezr(target, et, frame,
                               corrtm, observ)

print(' X      = {:.20.6f}'.format(state[0]))
print(' Y      = {:.20.6f}'.format(state[1]))
print(' Z      = {:.20.6f}'.format(state[2]))
print(' VX     = {:.20.6f}'.format(state[3]))
print(' VY     = {:.20.6f}'.format(state[4]))
print(' VZ     = {:.20.6f}'.format(state[5]))

target = 'SUN'
frame = 'CASSINI_INMS'
corrtm = 'LT+S'
observ = 'CASSINI'

sundir, ltime = spiceypy.spkpos(target, et, frame,
                               corrtm, observ)
sundir = spiceypy.vhat(sundir)

print('SUNDIR(X) = {:.20.6f}'.format(sundir[0]))
print('SUNDIR(Y) = {:.20.6f}'.format(sundir[1]))
print('SUNDIR(Z) = {:.20.6f}'.format(sundir[2]))

method = 'NEAR POINT: ELLIPSOID'
target = 'PHOEBE'
frame = 'IAU_PHOEBE'
corrtm = 'NONE'
observ = 'CASSINI'

spoint, trgepc, srfvec = spiceypy.subpnt(method, target, et,
                                          frame, corrtm, observ)

srad, slon, slat = spiceypy.reclat(spoint)

fromfr = 'IAU_PHOEBE'
tofr = 'CASSINI_INMS'

m2imat = spiceypy.pxform(fromfr, tofr, et)

sbpdir = spiceypy.mxv(m2imat, srfvec)
sbpdir = spiceypy.vhat(sbpdir)

print('LON      = {:.20.6f}'.format(slon * spiceypy.dpr()))
print('LAT      = {:.20.6f}'.format(slat * spiceypy.dpr()))
print('SBPDIR(X) = {:.20.6f}'.format(sbpdir[0]))
print('SBPDIR(Y) = {:.20.6f}'.format(sbpdir[1]))
print('SBPDIR(Z) = {:.20.6f}'.format(sbpdir[2]))

spiceypy.unload(mkfile)

if __name__ == '__main__':
    sscpnt()

```

Meta-kernel file “sscpnt.tm”:

KPL/MK

The names **and** contents of the kernels referenced by this meta-kernel are **as** follows:

File Name	Description
-----	-----
naif0008.tls	Generic LSK.
cas00084.tsc	Cassini SCLK.
020514_SE_SAT105.bsp	Saturnian Satellite Ephemeris SPK.
030201AP_SK_SM546_T45.bsp	Cassini Spacecraft SPK.
981005_PLTEPH-DE405S.bsp	Planetary Ephemeris SPK.
sat128.bsp	Saturnian Satellite Ephemeris SPK.
04135_04171pc_psiv2.bc	Cassini Spacecraft CK.
cas_v37.tf	Cassini FK.
cpck05Mar2004.tpc	Cassini project PCK.

```
\begindata
  KERNELS_TO_LOAD = (
    'kernels/lsk/naif0008.tls'
    'kernels/sclk/cas00084.tsc'
    'kernels/spk/020514_SE_SAT105.bsp'
    'kernels/spk/030201AP_SK_SM546_T45.bsp'
    'kernels/spk/981005_PLTEPH-DE405S.bsp'
    'kernels/spk/sat128.bsp'
    'kernels/ck/04135_04171pc_psiv2.bc'
    'kernels/fk/cas_v37.tf'
    'kernels/pck/cpck05Mar2004.tpc'
  )
\beginxt
```

6.4.30 Step-6: “Spacecraft Velocity”

6.4.31 “Spacecraft Velocity” Task Statement

Extend the program from Step-5 to compute the spacecraft velocity with respect to Phoebe in the INMS frame.

6.4.32 “Spacecraft Velocity” Hints

Compute velocity of the spacecraft with respect to Phoebe in some inertial frame, for example J2000. Recall that velocity is the last three components of the state vector returned by `spiceypy.spkezr`.

Since the velocity vector is computed in the inertial frame, it should be rotated to the instrument frame. Look at the previous step the routine that compute necessary rotation and rotate vectors.

Add calls to the routine(s), necessary variable declarations and output print statements to the program.

6.4.33 “Spacecraft Velocity” Solution Steps

All kernels required for computations in this step are already being loaded by the program, therefore, the meta-kernel does not need to be changed.

The spacecraft velocity vector is the last three components of the state returned by `spiceypy.spkezr`. To compute velocity of CASSINI with respect to Phoebe in the J2000 inertial frame the `spiceypy.spkezr` arguments should be set to ‘CASSINI’ (TARG), ‘PHOEBE’ (OBS), ‘J2000’ (REF) and ‘NONE’ (ABCORR).

The computed velocity vector has to be rotated from the J2000 frame to the instrument frame. The `spiceypy.pxform` routine used in the previous step can be used to compute the rotation matrix needed for that. In this case the frame name arguments should be set to ‘J2000’ (FROM) and ‘CASSINI_INMS’ (TO).

As in the previous step the difference vector should be then multiplied by this rotation matrix using the `spiceypy.m xv` routine. After applying the rotation, normalize the resultant vector using the `spiceypy.vhat` routine.

Putting it all together, we get:

```
target = 'CASSINI'
frame  = 'J2000'
corrtn = 'NONE'
observ = 'PHOEBE'

state, ltime = spiceypy.spkezr(target, et, frame,
                               corrtn, observ)

scvdir = state[3:6]

fromfr = 'J2000'
tofr    = 'CASSINI_INMS'
j2imat = spiceypy.pxform(fromfr, tofr, et)

scvdir = spiceypy.m xv(j2imat, scvdir)
scvdir = spiceypy.vhat(scvdir)
```

When you execute the script, “`scvel`”, it produces the following output:

```
> python scvel.py
UTC      = 2004-06-11T19:32:00
ET       = 140254384.184625
SCLK     = 1465674964.105
ET       = 140254384.183426
X        = -376599061.916539
Y        = 1294487780.929154
Z        = -7064853.054698
VX       = -5.164226
VY       = 0.801719
VZ       = 0.040603
SUNDIR(X) = -0.290204
SUNDIR(Y) = 0.881631
SUNDIR(Z) = 0.372167
LON      = 23.423158
LAT      = 3.709797
SBPDIR(X) = -0.000776
SBPDIR(Y) = -0.999873
SBPDIR(Z) = -0.015905
SCVDIR(X) = 0.395785
SCVDIR(Y) = -0.292808
SCVDIR(Z) = 0.870413
```

Note that computing the spacecraft velocity in the instrument frame by a single call to `spiceypy.spkezr` by specifying 'CASSINI_INMS' in the ``ref`` argument returns an incorrect result. Such computation will take into account the spacecraft angular velocity from the CK files, which should not be considered in this case.

6.4.34 “Spacecraft Velocity” Code Program” `scvel.py`:

```
from __future__ import print_function
import spiceypy

def scvel():

    mkfile = 'scvel.tm'
    spiceypy.furnsh(mkfile)

    utc = '2004-06-11T19:32:00'
    et = spiceypy.str2et(utc)

    print('UTC      = {:s}'.format(utc))
    print('ET       = {:20.6f}'.format(et))

    scid = -82
    sclk = '1465674964.105'
    et = spiceypy.scs2e(scid, sclk)

    print('SCLK      = {:s}'.format(sclk))
    print('ET       = {:20.6f}'.format(et))

    target = 'CASSINI'
    frame = 'ECLIPJ2000'
    corrtm = 'NONE'
    observ = 'SUN'

    state, ltime = spiceypy.spkezr(target, et, frame,
                                   corrtm, observ)

    print(' X      = {:20.6f}'.format(state[0]))
    print(' Y      = {:20.6f}'.format(state[1]))
    print(' Z      = {:20.6f}'.format(state[2]))
    print(' VX     = {:20.6f}'.format(state[3]))
    print(' VY     = {:20.6f}'.format(state[4]))
    print(' VZ     = {:20.6f}'.format(state[5]))

    target = 'SUN'
    frame = 'CASSINI_INMS'
    corrtm = 'LT+S'
    observ = 'CASSINI'

    sundir, ltime = spiceypy.spkpos(target, et, frame,
                                    corrtm, observ)
    sundir = spiceypy.vhat(sundir)

    print('SUNDIR(X) = {:20.6f}'.format(sundir[0]))
    print('SUNDIR(Y) = {:20.6f}'.format(sundir[1]))
    print('SUNDIR(Z) = {:20.6f}'.format(sundir[2]))

    method = 'NEAR POINT: ELLIPSOID'
```

(continues on next page)

(continued from previous page)

```

target = 'PHOEBE'
frame = 'IAU_PHOEBE'
corrtn = 'NONE'
observ = 'CASSINI'

spoint, trgepc, srfvec = spiceypy.subpnt(method, target, et,
                                         frame, corrtn, observ)

srad, slon, slat = spiceypy.reclat(spoint)

fromfr = 'IAU_PHOEBE'
tofr    = 'CASSINI_INMS'

m2imat = spiceypy.pxform(fromfr, tofr, et)

sbpdir = spiceypy.mxv(m2imat, srfvec)
sbpdir = spiceypy.vhat(sbpdir)

print('LON      = {:.2f}'.format(slon * spiceypy.dpr()))
print('LAT      = {:.2f}'.format(slat * spiceypy.dpr()))
print('SBPDIR(X) = {:.2f}'.format(sbpdir[0]))
print('SBPDIR(Y) = {:.2f}'.format(sbpdir[1]))
print('SBPDIR(Z) = {:.2f}'.format(sbpdir[2]))

target = 'CASSINI'
frame = 'J2000'
corrtn = 'NONE'
observ = 'PHOEBE'

state, ltime = spiceypy.spkezr(target, et, frame,
                               corrtn, observ)

scvdir = state[3:6]

fromfr = 'J2000'
tofr    = 'CASSINI_INMS'
j2imat = spiceypy.pxform(fromfr, tofr, et)

scvdir = spiceypy.mxv(j2imat, scvdir)
scvdir = spiceypy.vhat(scvdir)

print('SCVDIR(X) = {:.2f}'.format(scvdir[0]))
print('SCVDIR(Y) = {:.2f}'.format(scvdir[1]))
print('SCVDIR(Z) = {:.2f}'.format(scvdir[2]))

spiceypy.unload(mkfile)

if __name__ == '__main__':
    scvel()

```

Meta-kernel file “scvel.tm”:

KPL/MK

The names **and** contents of the kernels referenced by this meta-kernel are **as** follows:

(continues on next page)

(continued from previous page)

File Name	Description
naif0008.tls	Generic LSK.
cas00084.tsc	Cassini SCLK.
020514_SE_SAT105.bsp	Saturnian Satellite Ephemeris SPK.
030201AP_SK_SM546_T45.bsp	Cassini Spacecraft SPK.
981005_PLTEPH-DE405S.bsp	Planetary Ephemeris SPK.
sat128.bsp	Saturnian Satellite Ephemeris SPK.
04135_04171pc_psiv2.bc	Cassini Spacecraft CK.
cas_v37.tf	Cassini FK.
cpck05Mar2004.tpc	Cassini project PCK.

```

\begindata
  KERNELS_TO_LOAD = (
    'kernels/lsk/naif0008.tls'
    'kernels/sclk/cas00084.tsc'
    'kernels/spk/020514_SE_SAT105.bsp'
    'kernels/spk/030201AP_SK_SM546_T45.bsp'
    'kernels/spk/981005_PLTEPH-DE405S.bsp'
    'kernels/spk/sat128.bsp'
    'kernels/ck/04135_04171pc_psiv2.bc'
    'kernels/fk/cas_v37.tf'
    'kernels/pck/cpck05Mar2004.tpc'
  )
\beginext

```

6.5 Binary PCK Hands-On Lesson (Python)

November 20, 2017

6.5.1 Overview

In this lesson you will develop two programs that demonstrate geometric computations using “high-accuracy” Earth and Moon binary PCKs. The programs also demonstrate use of frame kernels and SPK files normally used together with these high-accuracy PCKs.

6.5.2 References

This section lists SPICE documents referred to in this lesson.

The following SPICE tutorials serve as references for the discussions in this lesson:

Name	Lesson steps/functions it describes
Frames	Moon rotation, Earth rotation
PCK	Moon rotation, Earth rotation
"High Accuracy Orientation and Body-Fixed frames for Moon	

(continues on next page)

(continued from previous page)

```
and Earth"
(backup)           Moon rotation, Earth rotation
```

These tutorials are available from the NAIF ftp server at JPL:

```
http://naif.jpl.nasa.gov/naif/tutorials.html
```

Required Readings

The Required Reading documents are provided with the Toolkit and are located under the “cspice/doc” directory in the CSPICE Toolkit installation tree.

Name	Lesson steps/functions that it describes
frames.req	Using reference frames
pck.req	Obtaining planetary constants data
spk.req	Obtaining ephemeris data
time.req	Time conversion

The Permuted Index

Another useful document distributed with the Toolkit is the permuted index. This is located under the “cspice/doc” directory in the C installation tree.

This text document provides a simple mechanism by which users can discover which SpiceyPy functions perform functions of interest, as well as the names of the source files that contain these functions.

SpiceyPy API Documentation

A SpiceyPy function’s parameters specification is available using the built-in Python help system.

For example, the Python help function

```
>>> import spiceypy
>>> help(spiceypy.str2et)
```

describes of the str2et function’s parameters, while the document

```
https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/str2et_c.html
```

describes extensively the str2et functionality.

6.5.3 Kernels Used

The following kernels are used in examples provided in this lesson:

#	FILE NAME	TYPE	DESCRIPTION
1	naif0008.tls	LSK	Generic LSK
2	de414_2000_2020.bsp	SPK	Solar System Ephemeris
3	moon_060721.tf	FK	Lunar FK
4	pck00008.tpc	PCK	NAIF text PCK
5	moon_pa_de403_1950-2198.bpc	PCK	Moon binary PCK
6	earthstns_itr93_050714.bsp	SPK	DSN station Ephemeris
7	earth_topo_050714.tf	FK	Earth topocentric FK
8	earth_000101_070725_070503.bpc	PCK	Earth binary PCK

These SPICE kernels are included in the lesson package available from the NAIF server at JPL:

```
ftp://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Lessons/
```

6.5.4 SpiceyPy Modules Used

This section provides a complete list of the functions and kernels that are suggested for usage in each of the exercises in this lesson. (You may wish to not look at this list unless/until you “get stuck” while working on your own.)

CHAPTER	EXERCISE	FUNCTIONS	NON-VOID	KERNELS
1	mrotat	spiceypy.furnsh spiceypy.unload	spiceypy.str2et spiceypy.spkpos spiceypy.reclat spiceypy.dpr spiceypy.vsep spiceypy.subpnt spiceypy.vdist	1-5
2	erotat	spiceypy.furnsh spiceypy.unload	spiceypy.str2et spiceypy.spkpos spiceypy.reclat spiceypy.dpr spiceypy.vsep spiceypy.spd spiceypy.timeout spiceypy.pxform spiceypy.twopi spiceypy.subslr spiceypy.vdist	1-2, 4, 6-8

Use the Python built-in help system on the various functions listed above for the API parameters’ description, and refer to the headers of their corresponding CSPICE versions for detailed interface specifications.

6.5.5 Moon rotation (mrotat)

6.5.6 Task Statement

Write a program that performs the following computations:

1. Convert the time string 2007 JAN 1 00:00:00 UTC to a double precision number representing seconds past J2000 TDB.

In the following instructions, we'll call the result of this computation ET.
2. Compute the apparent position of the Earth as seen from the Moon in the IAU_MOON reference frame at the epoch ET. Use light time and stellar aberration corrections. Use `spiceypy.reclat` to compute the planetocentric longitude and latitude of the Earth position vector; display these coordinates in degrees.
3. Repeat the computation of step 2 using the MOON_ME reference frame. Display the results as above.
4. Compute the angular separation of the position vectors found in

(continues on next page)

(continued from previous page)

- ```

steps 2 and 3. Display the result in degrees.

5. Repeat the computation of step 2 using the MOON_PA reference
 frame. Display the results as above.

6. Compute the angular separation of the position vectors found in
 steps 3 and 5 (these vectors are expressed in the MOON_ME and
 MOON_PA frames). Display the result in degrees.

7. Compute the apparent sub-Earth point on the Moon at ET,
 expressed in the MOON_ME reference frame and using light time
 and stellar aberration corrections. Convert the sub-Earth point
 to latitudinal coordinates using spiceypy.reclat. Display the
 longitude and latitude of the sub-Earth point in degrees.

8. Repeat step 7, now using the MOON_PA frame.

9. Compute the distance between the two sub-Earth points found
 above in steps 7 and 8. Display the result in kilometers.

```

### 6.5.7 Learning Goals

Familiarity with SPICE kernels required to obtain high-accuracy orientation of the Moon. Understanding the differences between results obtained using low and high-accuracy Moon orientation data. Understanding the difference between the MOON\_ME and MOON\_PA frames.

### 6.5.8 Approach

The following “tips” may simplify the solution process.

- ```

--  Examine the SPICE kernels provided with this lesson. Use BRIEF
    to find coverage periods of SPK kernels and binary PCKs. Use
    COMMNT to view the comment areas of binary PCKs. Examine text
    kernels, in particular text kernel comments, using a text
    editor or browser.

--  Decide which SPICE kernels are necessary. Prepare a meta-kernel
    listing the kernels and load it into the program.

--  Consult the above list titled "SpiceyPy Modules Used" to see
    which routines are needed.

--  The computational steps listed above should be followed in the
    order shown.

```

You may find it useful to consult the permuted index, the headers of various source modules, and the tutorials titled “PCK” and “High Accuracy Orientation and Body-Fixed frames for Moon and Earth.”

6.5.9 Solution

Solution Meta-Kernel

The meta-kernel we created for the solution to this exercise is named ‘mrotat.tm’. Its contents follow:

```
KPL/MK

Meta-kernel for the "Moon Rotation" task in the Binary PCK
Hands On Lesson.

The names and contents of the kernels referenced by this
meta-kernel are as follows:

File name                      Contents
-----
naif0008.tls                   Generic LSK
de414_2000_2020.bsp            Solar System Ephemeris
moon_060721.tf                 Lunar FK
pck00008.tpc                   NAIF text PCK
moon_pa_de403_1950-2198.bpc    Moon binary PCK

\begindata

    KERNELS_TO_LOAD = ( 'kernels/lsk/naif0008.tls'
                        'kernels/spk/de414_2000_2020.bsp'
                        'kernels/fk/moon_060721.tf'
                        'kernels/pck/pck00008.tpc'
                        'kernels/pck/moon_pa_de403_1950-2198.bpc' )

\begintext
```

Solution Source Code

A sample solution to the problem follows:

```
#
# Solution mrotat
#
from __future__ import print_function
#
# SpiceyPy package:
#
import spiceypy

def mrotat():
    #
    # Local parameters
    #
    METAKR = 'mrotat.tm'

    #
    # Load the kernels that this program requires.
    #
    spiceypy.furnsh( METAKR )

    #
    # Convert our UTC string to seconds past J2000 TDB.
    #
```

(continues on next page)

(continued from previous page)

```

timstr = '2007 JAN 1 00:00:00'
et      = spiceypy.str2et( timstr )

#
# Look up the apparent position of the Earth relative
# to the Moon's center in the IAU_MOON frame at ET.
#
[imoonv, ltime] = spiceypy.spkpos(
    'earth', et, 'iau_moon', 'lt+s', 'moon' )

#
# Express the Earth direction in terms of longitude
# and latitude in the IAU_MOON frame.
#
[r, lon, lat] = spiceypy.reclat( imoonv )

print( '\n'
    'Moon-Earth direction using low accuracy\n'
    'PCK and IAU_MOON frame:\n'
    'Earth lon (deg):      {0:15.6f}\n'
    'Earth lat (deg):      {1:15.6f}\n'.format(
        lon * spiceypy.dpr(),
        lat * spiceypy.dpr() ) )

#
# Look up the apparent position of the Earth relative
# to the Moon's center in the MOON_ME frame at ET.
#
[mmoonv, ltime] = spiceypy.spkpos( 'earth', et, 'moon_me',
    'lt+s', 'moon' )

#
# Express the Earth direction in terms of longitude
# and latitude in the MOON_ME frame.
#
[r, lon, lat] = spiceypy.reclat( mmoonv )

print( 'Moon-Earth direction using high accuracy\n'
    'PCK and MOON_ME frame:\n'
    'Earth lon (deg):      {0:15.6f}\n'
    'Earth lat (deg):      {1:15.6f}\n'.format(
        lon * spiceypy.dpr(),
        lat * spiceypy.dpr() ) )

#
# Find the angular separation of the Earth position
# vectors in degrees.
#
sep = spiceypy.dpr() * spiceypy.vsep( imoonv, mmoonv )

print( 'For IAU_MOON vs MOON_ME frames:' )
print( 'Moon-Earth vector separation angle (deg):      '
    '{:15.6f}\n'.format( sep ) )

#
# Look up the apparent position of the Earth relative
# to the Moon's center in the MOON_PA frame at ET.
#
[pmoonv, ltime] = spiceypy.spkpos( 'earth', et, 'moon_pa',
    'lt+s', 'moon' )

#

```

(continues on next page)

(continued from previous page)

```
# Express the Earth direction in terms of longitude
# and latitude in the MOON_PA frame.
#
[r, lon, lat] = spiceypy.reclat( pmoonv )

print( 'Moon-Earth direction using high accuracy\n'
      'PCK and MOON_PA frame:\n'
      'Earth lon (deg):      {0:15.6f}\n'
      'Earth lat (deg):      {1:15.6f}\n'.format(
          lon * spiceypy.dpr(),
          lat * spiceypy.dpr() ) )

#
# Find the angular separation of the Earth position
# vectors in degrees.
#
sep = spiceypy.dpr() * spiceypy.vsep( pmoonv, mmoonv )

print( 'For MOON_PA vs MOON_ME frames:' )
print( 'Moon-Earth vector separation angle (deg):      '
      '{:15.6f}\n'.format( sep ) )

#
# Find the apparent sub-Earth point on the Moon at ET
# using the MOON_ME frame.
#
[msub, trgepc, srfvec] = spiceypy.subpnt(
    'near point: ellipsoid', 'moon',
    et, 'moon_me', 'lt+s', 'earth' )

#
# Display the sub-point in latitudinal coordinates.
#
[r, lon, lat] = spiceypy.reclat( msub )

print( 'Sub-Earth point on Moon using high accuracy\n'
      'PCK and MOON_ME frame:\n'
      'Sub-Earth lon (deg):    {0:15.6f}\n'
      'Sub-Earth lat (deg):    {1:15.6f}\n'.format(
          lon * spiceypy.dpr(),
          lat * spiceypy.dpr() ) )

#
# Find the apparent sub-Earth point on the Moon at
# ET using the MOON_PA frame.
#
[psub, trgepc, srfvec] = spiceypy.subpnt(
    'near point: ellipsoid', 'moon',
    et, 'moon_pa', 'lt+s', 'earth' )

#
# Display the sub-point in latitudinal coordinates.
#
[r, lon, lat] = spiceypy.reclat( psub )

print( 'Sub-Earth point on Moon using high accuracy\n'
      'PCK and MOON_PA frame:\n'
      'Sub-Earth lon (deg):    {0:15.6f}\n'
      'Sub-Earth lat (deg):    {1:15.6f}\n'.format(
          lon * spiceypy.dpr(),
          lat * spiceypy.dpr() ) )

#
```

(continues on next page)

(continued from previous page)

```

# Find the distance between the sub-Earth points
# in km.
#
dist = spiceypy.vdist( msub, psub )

print( 'Distance between sub-Earth points (km): '
       '{:15.6f}\n'.format( dist ) )

spiceypy.unload( METAKR )

if __name__ == '__main__':
    mrotat()

```

Solution Sample Output

Execute the program:

```

Moon-Earth direction using low accuracy
PCK and IAU_MOON frame:
Earth lon (deg):           3.613102
Earth lat (deg):           -6.438342

Moon-Earth direction using high accuracy
PCK and MOON_ME frame:
Earth lon (deg):           3.611229
Earth lat (deg):           -6.439501

For IAU_MOON vs MOON_ME frames:
Moon-Earth vector separation angle (deg):           0.002194

Moon-Earth direction using high accuracy
PCK and MOON_PA frame:
Earth lon (deg):           3.593319
Earth lat (deg):           -6.417582

For MOON_PA vs MOON_ME frames:
Moon-Earth vector separation angle (deg):           0.028235

Sub-Earth point on Moon using high accuracy
PCK and MOON_ME frame:
Sub-Earth lon (deg):       3.611419
Sub-Earth lat (deg):       -6.439501

Sub-Earth point on Moon using high accuracy
PCK and MOON_PA frame:
Sub-Earth lon (deg):       3.593509
Sub-Earth lat (deg):       -6.417582

Distance between sub-Earth points (km):           0.856182

```

6.5.10 Earth rotation (erorat)

6.5.11 Task Statement

Write a program that performs the following computations:

1. Convert the time string 2007 JAN 1 00:00:00 UTC to a double precision number representing seconds past J2000 TDB.

In the following instructions, we'll call the result of this computation ET.
2. Compute the apparent position of the Moon as seen from the Earth in the IAU_EARTH reference frame at the epoch ET. Use light time and stellar aberration corrections. Display the planetocentric longitude and latitude of the Moon position vector in degrees.
3. Repeat the first computation using the ITRF93 reference frame. Display the results as above.
4. Compute the angular separation of the position vectors found the the previous two steps. Display the result in degrees.

The following computations (steps 5-10) examine the cause of the angular offset found above, which is attributable to the rotation between the ITRF93 and IAU_EARTH frames. Steps 11 and up don't rely on the results of steps 5-10, so steps 5-10 may be safely skipped if they're not of interest to you.

For each of the two epochs ET and ET + 100 days, examine the differences between the axes of the ITRF93 and IAU_EARTH frames using the following method:

5. Convert the epoch of interest to a string in the format style "2007-MAY-16 02:29:00.000 (UTC)." Display this string.
6. Look up the 3x3 position transformation matrix that converts vectors from the IAU_EARTH to the ITRF93 frame at the epoch of interest. We'll call the returned matrix RMAT.
7. Extract the first row of RMAT into a 3-vector, which we'll call ITRFX. This is the X-axis of the ITRF93 frame expressed relative to the IAU_EARTH frame.
8. Extract the third row of RMAT into a 3-vector, which we'll call ITRFZ. This is the Z-axis of the ITRF93 frame expressed relative to the IAU_EARTH frame.
9. Compute the angular separation between the vector ITRFX and the X-axis (1, 0, 0) of the IAU_EARTH frame. Display the result in degrees.
10. Compute the angular separation between the vector ITRFZ and the Z-axis (0, 0, 1) of the IAU_EARTH frame. Display the result in degrees.

This is the end of the computations to be performed for the epochs ET and ET + 100 days. The following steps are part of a new computation.

Find the azimuth and elevation of the apparent position of the Moon as seen from the DSN station DSS-13 by the

following steps:

11. Find the apparent position vector of the Moon relative to the DSN station DSS-13 in the topocentric reference frame DSS-13_TOPO at epoch ET. Use light time and stellar aberration corrections.

For this step, you'll need to have loaded a station SPK file providing geocentric station position vectors, as well as a frame kernel specifying topocentric reference frames centered at the respective DSN stations. (Other kernels will be needed as well; you must choose these.)
12. Convert the position vector to latitudinal coordinates. Use the routine `spiceypy.reclat` for this computation.
13. Compute the Moon's azimuth and elevation as follows: azimuth is the negative of topocentric longitude and lies within the range 0-360 degrees; elevation is equal to the topocentric latitude. Display the results in degrees.

The next computations demonstrate “high-accuracy” geometric computations using the Earth as the target body. These computations are *not* realistic; they are simply meant to demonstrate SPICE system features used for geometry computations involving the Earth as a target body. For example, the same basic techniques would be used to find the sub-solar point on the Earth as seen from an Earth-orbiting spacecraft.

14. Compute the apparent sub-solar point on the Earth at ET, expressed relative to the IAU_EARTH reference frame, using light time and stellar aberration corrections and using the Sun as the observer. Convert the sub-solar point to latitudinal coordinates using `spiceypy.reclat`. Display the longitude and latitude of the sub-solar point in degrees.
15. Repeat the sub-solar point computation described above, using the ITRF93 Earth body-fixed reference frame. Display the results as above.
16. Compute the distance between the two sub-solar points found above. Display the result in kilometers.

6.5.12 Learning Goals

Familiarity with SPICE kernels required to obtain high-accuracy orientation of the Earth. Understanding the differences between results obtained using low and high-accuracy Earth orientation data.

Understanding of topocentric frames and computation of target geometry relative to a surface location on the Earth. Knowledge of SPICE kernels required to support such computations.

6.5.13 Approach

The following “tips” may simplify the solution process.

- Examine the SPICE kernels provided with this lesson. Use BRIEF to find coverage periods of SPK kernels and binary PCKs. Use COMMNT to view the comment areas of binary PCKs. Examine text kernels, in particular text kernel comments, using a text editor or browser.
- Decide which SPICE kernels are necessary. Prepare a meta-kernel listing the kernels and load it into the program.
- Consult the above list titled "SpiceyPy Modules Used" to see which routines are needed. Note the functions used to provide the values "seconds per day," "degrees per radian," and "2 times Pi."
- Examine the header of the function spiceypy.reclat. Note that this function may be used for coordinate conversions in situations where the input rectangular coordinates refer to any reference frame, not only a body-centered, body-fixed frame whose X-Y plane coincides with the body's equator.
- The computational steps listed above should be followed in the order shown, but steps 5-10 may be omitted.

You may find it useful to consult the permuted index, the headers of various source modules, and the tutorials titled “PCK” and “High Accuracy Orientation and Body-Fixed frames for Moon and Earth.”

6.5.14 Solution

Solution Meta-Kernel

The meta-kernel we created for the solution to this exercise is named ‘erotat.tm’. Its contents follow:

```
KPL/MK

Meta-kernel for the "Earth Rotation" task
in the Binary PCK Hands On Lesson.

The names and contents of the kernels referenced by this
meta-kernel are as follows:

File name                               Contents
-----
naif0008.tls                            Generic LSK
de414_2000_2020.bsp                     Solar System Ephemeris
earthstns_itrf93_050714.bsp             DSN station Ephemeris
earth_topo_050714.tf                    Earth topocentric FK
pck00008.tpc                             NAIF text PCK
earth_000101_070725_070503.bpc         Earth binary PCK

\begindata

KERNELS_TO_LOAD = ( 'kernels/lsk/naif0008.tls'
```

(continues on next page)

(continued from previous page)

```

'kernels/spk/de414_2000_2020.bsp'
'kernels/spk/earthstns_itrf93_050714.bsp'
'kernels/fk/earth_topo_050714.tf'
'kernels/pck/pck00008.tpc'
'kernels/pck/earth_000101_070725_070503.bpc' )

\begin{text}

```

Solution Source Code

A sample solution to the problem follows:

```

#
# Solution mrotat
#
from __future__ import print_function
#
# SpiceyPy package:
#
import spiceypy

def erotat():
    #
    # Local parameters
    #
    METAKR = 'erotat.tm'

    x = [ 1.0, 0.0, 0.0 ]
    z = [ 0.0, 0.0, 1.0 ]

    #
    # Load the kernels that this program requires.
    #
    spiceypy.furnsh( METAKR )

    #
    # Convert our UTC string to seconds past J2000 TDB.
    #
    timstr = '2007 JAN 1 00:00:00'
    et      = spiceypy.str2et( timstr )

    #
    # Look up the apparent position of the Moon relative
    # to the Earth's center in the IAU_EARTH frame at ET.
    #
    [lmoonv, ltime] = spiceypy.spkpos( 'moon', et, 'iau_earth',
                                      'lt+s', 'earth' )

    #
    # Express the Moon direction in terms of longitude
    # and latitude in the IAU_EARTH frame.
    #
    [r, lon, lat] = spiceypy.reclat( lmoonv )

    print( 'Earth-Moon direction using low accuracy\n'
           'PCK and IAU_EARTH frame:\n'
           'Moon lon (deg):      {0:15.6f}\n'
           'Moon lat (deg):      {1:15.6f}\n'.format(

```

(continues on next page)

(continued from previous page)

```

        lon * spiceypy.dpr(),
        lat * spiceypy.dpr() ) )

#
# Look up the apparent position of the Moon relative
# to the Earth's center in the ITRF93 frame at ET.
#
[hmoonv, ltime] = spiceypy.spkpos( 'moon', et, 'ITRF93',
                                   'lt+s', 'earth' )

#
# Express the Moon direction in terms of longitude
# and latitude in the ITRF93 frame.
#
[r, lon, lat] = spiceypy.reclat( hmoonv )

print( 'Earth-Moon direction using high accuracy\n'
       'PCK and ITRF93 frame:\n'
       'Moon lon (deg):      {0:15.6f}\n'
       'Moon lat (deg):      {1:15.6f}\n'.format(
           lon * spiceypy.dpr(),
           lat * spiceypy.dpr() ) )

#
# Find the angular separation of the Moon position
# vectors in degrees.
#
sep = spiceypy.dpr() * spiceypy.vsep( lmoonv, hmoonv )

print( 'Earth-Moon vector separation angle (deg):      '
       '{:15.6f}\n'.format( sep ) )

#
# Next, express the +Z and +X axes of the ITRF93 frame in
# the IAU_EARTH frame. We'll do this for two times: et
# and et + 100 days.
#
for i in range(2):
    #
    # Set the time, expressing the time delta in
    # seconds.
    #
    t = et + i*spiceypy.spd()*100

    #
    # Convert the TDB time T to a string for output.
    #
    outstr = spiceypy.timeout(
        t, 'YYYY-MON-DD HR:MN:SC.### (UTC)' )

    print( 'Epoch: {:s}'.format( outstr ) )

    #
    # Find the rotation matrix for conversion of
    # position vectors from the IAU_EARTH to the
    # ITRF93 frame.
    #
    rmat = spiceypy.pxform( 'iau_earth', 'itr93', t )
    itr93x = rmat[0]
    itr93z = rmat[2]

```

(continues on next page)

(continued from previous page)

```

#
# Display the angular offsets of the ITRF93
# +X and +Z axes from their IAU_EARTH counterparts.
#
sep = spiceypy.vsep( itrfx, x )

print( 'ITRF93 - IAU_EARTH +X axis separation '
       'angle (deg): {:.13.6f}'.format(
           sep * spiceypy.dpr() ) )

sep = spiceypy.vsep( itr fz, z )

print( 'ITRF93 - IAU_EARTH +Z axis separation '
       'angle (deg): {:.13.6f}\n'.format(
           sep * spiceypy.dpr() ) )

#
# Find the azimuth and elevation of apparent
# position of the Moon in the local topocentric
# reference frame at the DSN station DSS-13.
# First look up the Moon's position relative to the
# station in that frame.
#
[topov, ltime] = spiceypy.spkpos( 'moon', et, 'DSS-13_TOPO',
                                'lt+s', 'DSS-13' )

#
# Express the station-moon direction in terms of longitude
# and latitude in the DSS-13_TOPO frame.
#
[r, lon, lat] = spiceypy.reclat( topov )

#
# Convert to azimuth-elevation.
#
az = -lon

if az < 0.0:
    az += spiceypy.twopi()

el = lat

print( 'DSS-13-Moon az/el using high accuracy '
       'PCK and DSS-13_TOPO frame:\n'
       'Moon Az (deg):      {0:15.6f}\n'
       'Moon El (deg):      {1:15.6f}\n'.format(
           az * spiceypy.dpr(),
           el * spiceypy.dpr() ) )

#
# Find the sub-solar point on the Earth at ET using the
# Earth body-fixed frame IAU_EARTH. Treat the Sun as
# the observer.
#
[lsun, trgepc, srfvec] = spiceypy.subslr(
    'near point: ellipsoid', 'earth', et,

```

(continues on next page)

(continued from previous page)

```

    'IAU_EARTH',          'lt+s',  'sun' );

#
# Display the sub-point in latitudinal coordinates.
#
[r, lon, lat] = spiceypy.reclat( lsub )

print( 'Sub-Solar point on Earth using low accuracy\n'
       'PCK and IAU_EARTH frame:\n'
       'Sub-Solar lon (deg):  {0:15.6f}\n'
       'Sub-Solar lat (deg):  {1:15.6f}\n'.format(
           lon * spiceypy.dpr(),
           lat * spiceypy.dpr() ) )

#
# Find the sub-solar point on the Earth at ET using the
# Earth body-fixed frame ITRF93. Treat the Sun as
# the observer.
#
[hsub, trgepc, srfvec] = spiceypy.subslr(
    'near point: ellipsoid', 'earth', et,
    'ITRF93',                'lt+s',  'sun' );

#
# Display the sub-point in latitudinal coordinates.
#
[r, lon, lat] = spiceypy.reclat( hsub )

print( 'Sub-Solar point on Earth using '
       'high accuracy \nPCK and ITRF93 frame:\n'
       'Sub-Solar lon (deg):  {0:15.6f}\n'
       'Sub-Solar lat (deg):  {1:15.6f}\n'.format(
           lon * spiceypy.dpr(),
           lat * spiceypy.dpr() ) )

#
# Find the distance between the sub-solar point
# vectors in km.
#
dist = spiceypy.vdist( lsub, hsub )

print( 'Distance between sub-solar points (km): '
       '{:15.6f}'.format( dist ) )

spiceypy.unload( METAKR )

if __name__ == '__main__':
    erotat()

```

Solution Sample Output

Execute the program:

```

Earth-Moon direction using low accuracy
PCK and IAU_EARTH frame:
Moon lon (deg):          -35.496272

```

(continues on next page)

(continued from previous page)

```

Moon lat (deg):                26.416959

Earth-Moon direction using high accuracy
PCK and ITRF93 frame:
Moon lon (deg):                -35.554286
Moon lat (deg):                26.419156

Earth-Moon vector separation angle (deg):          0.052002

Epoch: 2007-JAN-01 00:00:00.000 (UTC)
ITRF93 - IAU_EARTH +X axis separation angle (deg): 0.057677
ITRF93 - IAU_EARTH +Z axis separation angle (deg): 0.002326

Epoch: 2007-APR-10 23:59:59.998 (UTC)
ITRF93 - IAU_EARTH +X axis separation angle (deg): 0.057787
ITRF93 - IAU_EARTH +Z axis separation angle (deg): 0.002458

DSS-13-Moon az/el using high accuracy PCK and DSS-13_TOPO frame:
Moon Az (deg):                 72.169006
Moon El (deg):                 20.689488

Sub-Solar point on Earth using low accuracy
PCK and IAU_EARTH frame:
Sub-Solar lon (deg):           -177.100531
Sub-Solar lat (deg):           -22.910377

Sub-Solar point on Earth using high accuracy
PCK and ITRF93 frame:
Sub-Solar lon (deg):           -177.157874
Sub-Solar lat (deg):           -22.912593

Distance between sub-solar points (km):           5.881861

```

6.6 Other Stuff (Python)

November 20, 2017

The extensive scope of the SpiceyPy system’s functionality includes features the average user may not expect or appreciate, features NAIF refers to as “Other Stuff.” This workbook includes a set of lessons to introduce the beginning to moderate user to such features.

The lessons provide a brief description to several related sets of routines, associated reference documents, a programming task designed to teach the use of the routines, and an example solution to the programming problem.

6.6.1 Overview

This workbook contains lessons to demonstrate use of the less celebrated SpiceyPy routines.

1. Kernel Management **with** the Kernel Subsystem
2. The Kernel Pool
3. Coordinate Conversions
4. Advanced Time Manipulation Routines
5. Error Handling
6. Windows **and** Cells
7. Utility **and** Constants Routines

6.6.2 References

This section lists SPICE documents referred to in this lesson.

The following SPICE tutorials serve as references for the discussions in this lesson:

Name	Lesson steps/functions it describes
-----	-----
concepts	Concepts of space geometry and time
intro_to_kernels	Using kernels, meta-kernels
time	Time systems, conversions and formats
lsk_and_sclk	LSK and SCLK
derived_quant	"high-level" observation geometry computations
other_functions	Intro to some SPICE "low level" computations
exceptions	built-in mechanism for trapping/handling errors

These tutorials are available from the NAIF ftp server at JPL:

<http://naif.jpl.nasa.gov/naif/tutorials.html>

Required Readings

The Required Reading documents are provided with the Toolkit and are located under the "cspice/doc" directory in the CSPICE Toolkit installation tree.

Name	Lesson steps/functions that it describes
-----	-----
cells.req	The SPICE cell data type
error.req	The SPICE error handling system
kernel.req	Loading SPICE kernels
time.req	Time conversion
windows.req	The SPICE window data type

The Permuted Index

Another useful document distributed with the Toolkit is the permuted index. This is located under the "cspice/doc" directory in the C installation tree.

This text document provides a simple mechanism by which users can discover which SpiceyPy functions perform functions of interest, as well as the names of the source files that contain these functions.

SpiceyPy API Documentation

A SpiceyPy function’s parameters specification is available using the built-in Python help system.

For example, the Python help function

```
>>> import spiceypy
>>> help(spiceypy.str2et)
```

describes of the str2et function’s parameters, while the document

```
https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/str2et_c.html
```

describes extensively the str2et functionality.

6.6.3 Kernels Used

The following kernels are used in examples provided in this lesson:

#	FILE NAME	TYPE	DESCRIPTION
1	naif0008.tls	LSK	Generic LSK
2	de405s.bsp	SPK	Planet Ephemeris SPK
3	pck00008.tpc	PCK	Generic PCK

These SPICE kernels are included in the lesson package available from the NAIF server at JPL:

```
ftp://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Lessons/
```

6.6.4 SpiceyPy Modules Used

This section provides a complete list of the functions and kernels that are suggested for usage in each of the exercises in this lesson. (You may wish to not look at this list unless/until you “get stuck” while working on your own.)

CHAPTER	EXERCISE	FUNCTIONS	NON-VOID	KERNELS
1	kpool	spiceypy.furnsh spiceypy.unload spiceypy.kclear	spiceypy.ktotal spiceypy.kdata	1-3
2	kervar	spiceypy.furnsh spiceypy.kclear	spiceypy.gnpool spiceypy.dtpool spiceypy.gdpool spiceypy.gcpool	1-3
3	coord	spiceypy.furnsh spiceypy.kclear	spiceypy.dpr spiceypy.str2et spiceypy.bodvrd spiceypy.spkpos spiceypy.recrad spiceypy.reclat spiceypy.recsph spiceypy.recgeo	1-3
4	xtic	spiceypy.furnsh	spiceypy.str2et	1

(continues on next page)

(continued from previous page)

		spiceypy.tsetyr spiceypy.kclear	spiceypy.timeout spiceypy.tpicttr spiceypy.jyear	
5	aderr	spiceypy.furnsh spiceypy.kclear	spiceypy.spkezr	1-3
6	win	spiceypy.furnsh spiceypy.wninsd spiceypy.kclear	spiceypy.str2et spiceypy.wnvald spiceypy.wnintd spiceypy.card spiceypy.wnfetd spiceypy.et2utc spiceypy.wnsumd	1-3
7	units		spiceypy.tkvrns spiceypy.convrt	
	xconst		spiceypy.spd spiceypy.dpr spiceypy.rpd spiceypy.clight spiceypy.j2100 spiceypy.j2000 spiceypy.tyear spiceypy.halfpi	

Use the Python built-in help system on the various functions listed above for the API parameters' description, and refer to the headers of their corresponding CSPICE versions for detailed interface specifications.

6.6.5 NAIF Documentation

The technical complexity of the various SPICE subsystems mandates an extensive, user-friendly documentation set. The set differs somewhat depending on your choice of development language but provides the same information with regards to SPICE operation. The sources for a user needing information concerning SPICE are:

```
-- Required Readings and Users Guides
-- Library Source Code Documentation
-- API Documentation
-- Tutorials
```

Required Reading and Users Guides

NAIF Required Reading (*.req) documents introduce the functionality of particular SpiceyPy subsystems:

```
abcorr.req
cells.req
ck.req
daf.req
das.req
dla.req
dsk.req
```

(continues on next page)

(continued from previous page)

```
ek.req
ellipses.req
error.req
frames.req
gf.req
kernel.req
naif_ids.req
pck.req
planes.req
problems.req
rotation.req
scanning.req
sclk.req
sets.req
spc.req
spk.req
symbols.req
time.req
windows.req
```

NAIF Users Guides (*.ug) describe the proper use of particular SpiceyPy tools:

```
brief.ug
chronos.ug
ckbrief.ug
commnt.ug
convert.ug
dskbrief.ug
dskexp.ug
frmdiff.ug
inspekt.ug
mkdsk.ug
mkspk.ug
msopck.ug
simple.ug
spacit.ug
spkdiff.ug
spkmerge.ug
states.ug
subpt.ug
tictoc.ug
tobin.ug
toxfr.ug
version.ug
```

These text documents exist in the ‘doc’ directory of the main CSPICE Toolkit directory:

```
../cspice/doc/
```

HTML format documentation

The SpiceyPy distributions include HTML versions of Required Readings and Users Guides, accessible from the HTML documentation directory:

```
../cspice/doc/html/index.html
```

Library Source Code Documentation

All SPICELIB and CSPICE source files include usage and design information incorporated in a comment block known as the “header.” (Every toolkit includes either the SPICELIB or CSPICE library.)

A header consists of several marked sections:

```
-- Procedure: Routine name and one line expansion of the routine's
name.

-- Abstract: A tersely worded explanation describing the routine.

-- Copyright: An identification of the copyright holder for the
routine.

-- Required_Reading: A list of SpiceyPy required reading documents
relating to the routine.

-- Brief_I/O: A table of arguments, identifying each as either
input, output, or both, with a very brief description of the
variable.

-- Detailed_Input & Detailed_Output: An elaboration of the
Brief_I/O section providing comprehensive information on
argument use.

-- Parameters: Description and declaration of any parameters
(constants) specific to the routine.

-- Exceptions: A list of error conditions the routine detects and
signals plus a discussion of any other exceptional conditions
the routine may encounter.

-- Files: A list of other files needed for the routine to operate.

-- Particulars: A discussion of the routine's function (if
needed). This section may also include information relating to
"how" and "why" the routine performs an operation and to
explain functionality of routines that operate by side effects.

-- Examples: Descriptions and code snippets concerning usage of
the routine.

-- Restrictions: Restrictions or warnings concerning use.

-- Literature_References: A list of sources required to understand
the algorithms or data used in the routine.

-- Author_and_Institution: The names and affiliations for authors
of the routine.

-- Version: A list of edits and the authors of those edits made to
the routine since initial delivery to the SpiceyPy system.
```

The source code for SpiceyPy products is stored in ‘src’ sub-directory of the main SpiceyPy directory:

API Documentation

The SpiceyPy package is documented in “readthedocs” website:

```
https://spiceypy.readthedocs.io/en/master/index.html
```

Each API documentation page is in large part copied from the “Abstract” and “Brief_I/O” sections of the corresponding CSPICE function documentation. Each API page includes a link to the API documentation for the CSPICE routine called by the SpiceyPy interface.

This SpiceyPy API documentation (the same information as in the website but without hyperlinks) is also available from the Python built-in help system:

```
>>> help ( spiceypy.str2et )
Help on function str2et in module spiceypy.spiceypy:

str2et(*args, **kwargs)
    Convert a string representing an epoch to a double precision
    value representing the number of TDB seconds past the J2000
    epoch corresponding to the input epoch.

    ...

:param time: A string representing an epoch.
:type time: str
:return: The equivalent value in seconds past J2000, TDB.
:rtype: float
```

In order to have offline access to the documentation it is recommended to have the CSPICE Toolkit installed locally. The CSPICE package includes the CSPICE Reference Guide, an index of all CSPICE wrapper APIs with hyperlinks to API specific documentation. Each API documentation page includes cross-links to any other wrapper API mentioned in the document and links to the wrapper source code.

```
...cspice/doc/html/cspice/index.html
```

6.6.6 Text kernels

Several workbooks use SPICE text kernels. SPICE identifies a text kernel as an ASCII text file containing the mark-up tags the kernel subsystem requires to identify data assignments in that file, and “name=value” data assignments.

The subsystem uses two tags:

```
\begintext
```

and

```
\begindata
```

to mark information blocks within the text kernel. The `\begintext` tag specifies all text following the tag as comment information to be ignored by the subsystem.

Things to know:

1. The `\begindata` tag marks the start of a data definition block. The subsystem processes **all** text following this marker **as** SPICE kernel data assignments until finding a `\begintext` marker.
2. The kernel subsystem defaults to the `\begintext` mode until the parser encounters a `\begindata` tag. Once **in** `\begindata` mode the subsystem processes **all** text **as** variable assignments until the **next** `\begintext` tag.

(continues on next page)

(continued from previous page)

3. Enter the tags **as** the only text on a line, i.e.:

```
\begintext

... commentary information on the data assignments ...

\begindata

... data assignments ...
```

4. CSPICE delivery N0059 added to the CSPICE **and** Icy text kernel parsers the functionality to read non native text kernels, i.e. a Unix compiled library can read a MS Windows native text kernel, a MS Windows compiled library can read a Unix native text kernel. Mice acquires this capability **from CSPICE**.
5. With regards to the FORTRAN distribution, **as** of delivery N0057 the spiceypy.furnsh call includes a line terminator check, signaling an error on **any** attempt to read non-native text kernels.

Text kernel format

Scalar assignments.

```
VAR_NAME_DP  = 1.234
VAR_NAME_INT = 1234
VAR_NAME_STR = 'FORBIN'
```

Please note the use of a single quote in string assignments.

Vector assignments. Vectors must contain the same type data.

```
VEC_NAME_DP  = ( 1.234 , 45.678 , 901234.5 )
VEC_NAME_INT = ( 1234 , 456 , 789 )
VEC_NAME_STR = ( 'FORBIN', 'FALKEN', 'ROBUR' )
```

also

```
VEC_NAME_DP  = ( 1.234,
                  45.678,
                  901234.5 )

VEC_NAME_STR = ( 'FORBIN',
                  'FALKEN',
                  'ROBUR' )
```

Time assignments.

```
TIME_VAL = @31-JAN-2003-12:34:56.798
TIME_VEC = ( @01-DEC-2004, @15-MAR-2004 )
```

The at-sign character '@' indicates a time string. The pool subsystem converts the strings to double precision TDB (a numeric value). Please note, the time strings must not contain embedded blanks. WARNING - a TDB string is not the same as a UTC string.

The above examples depict direct assignments via the '=' operator. The kernel pool also permits incremental assignments via the '+=' operator.

Please refer to the kernels required reading, `kernel.req`, for additional information.

6.6.7 Lesson 1: Kernel Management with the Kernel Subsystem

6.6.8 Task Statement

Write a program to load a meta kernel, interrogate the SpiceyPy system for the names and types of all loaded kernels, then demonstrate the unload functionality and the resulting effects.

6.6.9 Learning Goals

This lesson demonstrates use of the kernel subsystem to load, unload, and list loaded kernels.

This lesson requires creation of a SPICE meta kernel.

6.6.10 Code Solution

First, create a meta text kernel:

You can use two versions of a meta kernel with code examples (`kpool.tm`) in this lesson. Either a kernel with explicit path information:

```
KPL/MK

\begindata

    KERNELS_TO_LOAD = ( 'kernels/spk/de405s.bsp',
                        'kernels/pck/pck00008.tpc',
                        'kernels/lsk/naif0008.tls' )

\begintext
```

... or a more generic meta kernel using the `PATH_VALUES`/`PATH_SYMBOLS` functionality to declare path names as variables:

```
KPL/MK

Define the paths to the kernel directory. Use the PATH_SYMBOLS
as aliases to the paths.

The names and contents of the kernels referenced by this
meta-kernel are as follows:

    File Name      Description
    -----
    naif0008.tls   Generic LSK.
    de405s.bsp     Planet Ephemeris SPK.
    pck00008.tpc   Generic PCK.

\begindata
```

(continues on next page)

(continued from previous page)

```
PATH_VALUES      = ( 'kernels/lsk',
                     'kernels/spk',
                     'kernels/pck' )

PATH_SYMBOLS     = ( 'LSK', 'SPK', 'PCK' )

KERNELS_TO_LOAD = ( '$LSK/naif0008.tls',
                     '$SPK/de405s.bsp',
                     '$PCK/pck00008.tpc' )

\begin{text}
```

Now the solution source code:

```
from __future__ import print_function

#
# Import the CSPICE-Python interface.
#
import spiceypy

def kpool():

    #
    # Assign the path name of the meta kernel to META.
    #
    META = 'kpool.tm'

    #
    # Load the meta kernel then use KTOTAL to interrogate the SPICE
    # kernel subsystem.
    #
    spiceypy.furnsh( META )

    count = spiceypy.ktotal( 'ALL' );
    print( 'Kernel count after load:          {0}\n'.format(count))

    #
    # Loop over the number of files; interrogate the SPICE system
    # with spiceypy.kdata for the kernel names and the type.
    # 'found' returns a boolean indicating whether any kernel files
    # of the specified type were loaded by the kernel subsystem.
    # This example ignores checking 'found' as kernels are known
    # to be loaded.
    #
    for i in range(0, count):
        [ file, type, source, handle] = spiceypy.kdata(i, 'ALL');
        print( 'File      {0}'.format(file) )
        print( 'Type      {0}'.format(type) )
        print( 'Source {0}\n'.format(source) )

    #
```

(continues on next page)

(continued from previous page)

```

# Unload one kernel then check the count.
#
spiceypy.unload( 'kernels/spk/de405s.bsp')
count = spiceypy.ktotal( 'ALL' );

#
# The subsystem should report one less kernel.
#
print( 'Kernel count after one unload:  {0}'.format(count))

#
# Now unload the meta kernel. This action unloads all
# files listed in the meta kernel.
#
spiceypy.unload( META )

#
# Check the count; spiceypy should return a count of zero.
#
count = spiceypy.ktotal( 'ALL');
print( 'Kernel count after meta unload: {0}'.format(count))

#
# Done. Unload the kernels.
#
spiceypy.kclear

if __name__ == '__main__':
    kpool()

```

Run the code example

First we see the number of all loaded kernels returned from the `spiceypy.ktotal` call.

Then the `spiceypy.kdata` loop returns the name of each loaded kernel, the type of kernel (SPK, CK, TEXT, etc.) and the source of the kernel - the mechanism that loaded the kernel. The source either identifies a meta kernel, or contains an empty string. An empty source string indicates a direct load of the kernel with a `spiceypy.furnsh` call.

```

Kernel count after load:      4

File   kpool.tm
Type   META
Source

File   kernels/lsk/naif0008.tls
Type   TEXT
Source kpool.tm

File   kernels/spk/de405s.bsp
Type   SPK
Source kpool.tm

File   kernels/pck/pck00008.tpc
Type   TEXT
Source kpool.tm

```

(continues on next page)

(continued from previous page)

```
Kernel count after one unload: 3
Kernel count after meta unload: 0
```

6.6.11 Lesson 2: The Kernel Pool

6.6.12 Task Statement

Write a program to retrieve particular string and numeric text kernel variables, both scalars and arrays. Interrogate the kernel pool for assigned variable names.

6.6.13 Learning Goals

The lesson demonstrates the SpiceyPy system's facility to retrieve different types of data (string, numeric, scalar, array) from the kernel pool.

For the code examples, use this generic text kernel (kervar.tm) containing PCK-type data, kernels to load, and example time strings:

```
KPL/MK

Name the kernels to load. Use path symbols.

The names and contents of the kernels referenced by this
meta-kernel are as follows:

      File Name      Description
      -
naif0008.tls        Generic LSK.
de405s.bsp          Planet Ephemeris SPK.
pck00008.tpc        Generic PCK.

\begindata

  PATH_VALUES      = ('kernels/spk',
                     'kernels/pck',
                     'kernels/lsk')

  PATH_SYMBOLS     = ('SPK' , 'PCK' , 'LSK' )

  KERNELS_TO_LOAD = ( '$SPK/de405s.bsp',
                       '$PCK/pck00008.tpc',
                       '$LSK/naif0008.tls')

\beginntext

Ring model data.

\begindata

  BODY699_RING1_NAME = 'A Ring'
  BODY699_RING1      = (122170.0 136780.0 0.1 0.1 0.5)
```

(continues on next page)

(continued from previous page)

```

BODY699_RING1_1_NAME    = 'Encke Gap'
BODY699_RING1_1         = (133405.0 133730.0 0.0 0.0 0.0)

BODY699_RING2_NAME      = 'Cassini Division'
BODY699_RING2           = (117580.0 122170.0 0.0 0.0 0.0)

\begintext

The kernel pool recognizes values preceded by '@' as time
values. When read, the kernel subsystem converts these
representations into double precision ephemeris time.

Caution: The kernel subsystem interprets the time strings
identified by '@' as TDB. The same string passed as input
to @STR2ET is processed as UTC.

The three expressions stored in the EXAMPLE_TIMES array represent
the same epoch.

\begindata

    EXAMPLE_TIMES        = ( @APRIL-1-2004-12:34:56.789,
                             @4/1/2004-12:34:56.789,
                             @JD2453097.0242684
                             )

\begintext

```

The main references for pool routines are found in the help command, the CSPICE source files or the API documentation for the particular routines.

6.6.14 Code Solution

```

from __future__ import print_function

#
# Import the CSPICE-Python interface.
#
import spiceypy
from spiceypy.utils.support_types import SpiceyError

def kervar():
    #
    # Define the max number of kernel variables
    # of concern for this examples.
    #
    N_ITEMS = 20

    #
    # Load the example kernel containing the kernel variables.
    # The kernels defined in KERNELS_TO_LOAD load into the
    # kernel pool with this call.
    #

```

(continues on next page)

(continued from previous page)

```

spiceypy.furnsh( 'kernvar.tm' )

#
# Initialize the start value. This value indicates
# index of the first element to return if a kernel
# variable is an array. START = 0 indicates return everything.
# START = 1 indicates return everything but the first element.
#
START = 0

#
# Set the template for the variable names to find. Let's
# look for all variables containing the string RING.
# Define this with the wildcard template '*RING*'. Note:
# the template '*RING' would match any variable name
# ending with the RING string.
#
tmplate = '*RING*'

#
# We're ready to interrogate the kernel pool for the
# variables matching the template. spiceypy.gnpool tells us:
#
# 1. Does the kernel pool contain any variables that
#    match the template (value of found).
# 2. If so, how many variables?
# 3. The variable names. (cvals, an array of strings)
#

try:
    cvals = spiceypy.gnpool( tmplate, START, N_ITEMS )
    print( 'Number variables matching template: {0}'.\
format( len(cvals)) )
except SpiceyError:
    print( 'No kernel variables matched template.' )
    return

#
# Okay, now we know something about the kernel pool
# variables of interest to us. Let's find out more...
#
for cval in cvals:

    #
    # Use spiceypy.dtpool to return the dimension and type,
    # C (character) or N (numeric), of each pool
    # variable name in the cvals array. We know the
    # kernel data exists.
    #
    [dim, type] = spiceypy.dtpool( cval )

    print( '\n' + cval )
    print( ' Number items: {0}   Of type: {1}\n'.\
format(dim, type) )

#

```

(continues on next page)

(continued from previous page)

```

# Test character equality, 'N' or 'C'.
#
if type == 'N':

    #
    # If 'type' equals 'N', we found a numeric array.
    # In this case any numeric array will be an array
    # of double precision numbers ('doubles').
    # spiceypy.gdpool retrieves doubles from the
    # kernel pool.
    #
    dvars = spiceypy.gdpool( cval, START, N_ITEMS )
    for dvar in dvars:
        print('  Numeric value: {0:20.6f}'.format(dvar))

elif type == 'C':

    #
    # If 'type' equals 'C', we found a string array.
    # spiceypy.gcpool retrieves string values from the
    # kernel pool.
    #
    cvars = spiceypy.gcpool( cval, START, N_ITEMS )

    for cvar in cvars:
        print('  String value: {0}\n'.format(cvar))

else:

    #
    # This block should never execute.
    #
    print('Unknown type. Code error.')

#
# Now look at the time variable EXAMPLE_TIMES. Extract this
# value as an array of doubles.
#
dvars = spiceypy.gdpool( 'EXAMPLE_TIMES', START, N_ITEMS )

print( 'EXAMPLE_TIMES' )

for dvar in dvars:
    print('  Time value:      {0:20.6f}'.format(dvar))

#
# Done. Unload the kernels.
#
spiceypy.kclear

if __name__ == '__main__':
    kervar()

```

Run the code example

The program runs and first reports the number of kernel pool variables matching the template, 6.

The program then loops over the `spiceypy.dtpool` 6 times, reporting the name of each pool variable, the number of data items assigned to that variable, and the variable type. Within the `spiceypy.dtpool` loop, a second loop outputs the contents of the data variable using `spiceypy.gcpool` or `spiceypy.gdpool`.

```
Number variables matching template: 6

BODY699_RING1_1
  Number items: 5    Of type: N

    Numeric value:      133405.000000
    Numeric value:      133730.000000
    Numeric value:           0.000000
    Numeric value:           0.000000
    Numeric value:           0.000000

BODY699_RING1
  Number items: 5    Of type: N

    Numeric value:      122170.000000
    Numeric value:      136780.000000
    Numeric value:           0.100000
    Numeric value:           0.100000
    Numeric value:           0.500000

BODY699_RING2
  Number items: 5    Of type: N

    Numeric value:      117580.000000
    Numeric value:      122170.000000
    Numeric value:           0.000000
    Numeric value:           0.000000
    Numeric value:           0.000000

BODY699_RING1_1_NAME
  Number items: 1    Of type: C

    String value: Encke Gap

BODY699_RING2_NAME
  Number items: 1    Of type: C

    String value: Cassini Division

BODY699_RING1_NAME
  Number items: 1    Of type: C

    String value: A Ring

EXAMPLE_TIMES
  Time value:      134094896.789000
  Time value:      134094896.789000
  Time value:      134094896.789753
```

Note the final time value differs from the previous values in the final three decimal places despite the intention that all three strings represent the same time. This results from round-off when converting a decimal Julian day representation to the seconds past J2000 ET representation.

6.6.15 Related Routines

```
--  spiceypy.gipool retrieves integer values from the kernel
    subsystem.
```

6.6.16 Lesson 3: Coordinate Conversions

6.6.17 Task Statement

Write a program to convert a Cartesian 3-vector representing some location to the other coordinate representations. Use the position of the Moon with respect to Earth in an inertial and non-inertial reference frame as the example vector.

6.6.18 Learning Goals

The SpiceyPy system provides functions to convert coordinate tuples between Cartesian and various non Cartesian coordinate systems including conversion between geodetic and rectangular coordinates.

This lesson presents these coordinate transform routines for rectangular, cylindrical, and spherical systems.

6.6.19 Code Solution

```
from __future__ import print_function
from builtins import input
import sys

#
# Import the CSPICE-Python interface.
#
import spiceypy

def coord():

    #
    # Define the inertial and non inertial frame names.
    #
    # Initialize variables or set type. All variables
    # used in a PROMPT construct must be initialized
    # as strings.
    #
    INRFRM = 'J2000'
    NONFRM = 'IAU_EARTH'
    r2d = spiceypy.dpr()

    #
    # Load the needed kernels using a spiceypy.furnsh call on the
    # meta kernel.
    #
    spiceypy.furnsh( 'coord.tm' )

    #
    # Prompt the user for a time string. Convert the
    # time string to ephemeris time J2000 (ET).
```

(continues on next page)

(continued from previous page)

```

#
timstr = input( 'Time of interest: ' )
et      = spiceypy.str2et( timstr )

#
# Access the kernel pool data for the triaxial radii of the
# Earth, rad[1][0] holds the equatorial radius, rad[1][2]
# the polar radius.
#
rad = spiceypy.bodvrd( 'EARTH', 'RADII', 3 )

#
# Calculate the flattening factor for the Earth.
#
#      equatorial_radius - polar_radius
# flat =  -----
#      equatorial_radius
#
flat = (rad[1][0] - rad[1][2])/rad[1][0]

#
# Make the spiceypy.spkpos call to determine the apparent
# position of the Moon w.r.t. to the Earth at 'et' in the
# inertial frame.
#
[pos, ltime] = spiceypy.spkpos('MOON', et, INRFRM,
                              'LT+S', 'EARTH' )

#
# Show the current frame and time.
#
print( ' Time : {0}'.format(timstr) )
print( ' Inertial Frame: {0}\n'.format(INRFRM) )

#
# First convert the position vector
# X = pos(1), Y = pos(2), Z = pos(3), to RA/DEC.
#
[ range, ra, dec ] = spiceypy.recrad( pos )

print( '   Range/Ra/Dec' )
print( '   Range: {0:20.6f}'.format(range) )
print( '   RA   : {0:20.6f}'.format(ra * r2d) )
print( '   DEC  : {0:20.6f}'.format(dec* r2d) )

#
# ...latitudinal coordinates...
#
[ range, lon, lat ] = spiceypy.reclat( pos )
print( '   Latitudinal ' )
print( '   Rad  : {0:20.6f}'.format(range) )
print( '   Lon  : {0:20.6f}'.format(lon * r2d) )
print( '   Lat  : {0:20.6f}'.format(lat * r2d) )

#
# ...spherical coordinates use the colatitude,

```

(continues on next page)

(continued from previous page)

```

# the angle from the Z axis.
#
[ range, colat, lon ] = spiceypy.recsph( pos )
print( '   Spherical' )
print('   Rad  : {0:20.6f}'.format(range) )
print('   Lon  : {0:20.6f}'.format(lon * r2d) )
print('   Colat: {0:20.6f}'.format(colat * r2d) )

#
# Make the spiceypy.spkpos call to determine the apparent
# position of the Moon w.r.t. to the Earth at 'et' in the
# non-inertial, body fixed, frame.
#
[pos, ltime] = spiceypy.spkpos('MOON', et, NONFRM,
                              'LT+S', 'EARTH')

print()
print( '   Non-inertial Frame: {0}'.format(NONFRM) )

#
# ...latitudinal coordinates...
#
[ range, lon, lat ] = spiceypy.reclat( pos )
print('   Latitudinal ' )
print('   Rad  : {0:20.6f}'.format(range) )
print('   Lon  : {0:20.6f}'.format(lon * r2d) )
print('   Lat  : {0:20.6f}'.format(lat * r2d) )

#
# ...spherical coordinates use the colatitude,
# the angle from the Z axis.
#
[ range, colat, lon ] = spiceypy.recsph( pos )
print( '   Spherical' )
print('   Rad  : {0:20.6f}'.format(range) )
print('   Lon  : {0:20.6f}'.format(lon * r2d) )
print('   Colat: {0:20.6f}'.format(colat * r2d) )

#
# ...finally, convert the position to geodetic coordinates.
#
[ lon, lat, range ] = spiceypy.recgeo( pos, rad[1][0], flat )
print( '   Geodetic' )
print('   Rad  : {0:20.6f}'.format(range) )
print('   Lon  : {0:20.6f}'.format(lon * r2d) )
print('   Lat  : {0:20.6f}'.format(lat * r2d) )
print()

#
# Done. Unload the kernels.
#
spiceypy.kclear

if __name__ == '__main__':
    coord()

```

Run the code example

Input “Feb 3 2002 TDB” to calculate the Moon’s position. (the ‘TDB’ tag indicates a Barycentric Dynamical Time value).

```
Time of interest: Feb 3 2002 TDB
```

Examine the Moon position in the J2000 inertial frame, display the time and frame:

```
Time : Feb 3 2002 TDB
Inertial Frame: J2000
```

Convert the Moon Cartesian coordinates to right ascension declination.

```
Range/Ra/Dec
Range:      369340.815193
RA   :      203.643686
DEC  :      -4.979010
```

Latitudinal. Note the difference in the expressions for longitude and right ascension though they represent a measure of the same quantity. The RA/DEC system measures RA in the interval $[0, 2\pi]$. Latitudinal coordinates measures longitude in the interval $(-\pi, \pi]$.

```
Latitudinal
Rad  :      369340.815193
Lon  :      -156.356314
Lat  :      -4.979010
```

Spherical. Note the difference between the expression of latitude in the Latitudinal system and the corresponding Spherical colatitude. The spherical coordinate system uses the colatitude, the angle measure away from the positive Z axis. Latitude is the angle between the position vector and the x-y (equatorial) plane with positive angle defined as toward the positive Z direction

```
Spherical
Rad  :      369340.815193
Lon  :      -156.356314
Colat:      94.979010
```

The same position look-up in a body fixed (non-inertial) frame, IAU_EARTH.

```
Non-inertial Frame: IAU_EARTH
```

Latitudinal coordinates return the geocentric latitude.

```
Latitudinal
Rad  :      369340.815193
Lon  :      70.986950
Lat  :      -4.989675
```

Spherical.

```
Spherical
Rad  :      369340.815193
Lon  :      70.986950
Colat:      94.989675
```

Geodetic. The cartographic lat/lon.

```
Geodetic
Rad  :      362962.836755
Lon  :      70.986950
Lat  :      -4.990249
```

6.6.20 Related Routines

```
--  spiceypy.latrec, latitudinal to rectangular
--  spiceypy.latcyl, latitudinal to cylindrical
--  spiceypy.latsph, latitudinal to spherical
--  spiceypy.reccyl, rectangular to cylindrical
--  spiceypy.sphrec, spherical to rectangular
--  spiceypy.sphcyl, spherical to cylindrical
--  spiceypy.sphlat, spherical to latitudinal
--  spiceypy.cyllat, cylindrical to latitudinal
--  spiceypy.cylsph, cylindrical to spherical
--  spiceypy.cylrec, cylindrical to rectangular
--  spiceypy.georec, geodetic to rectangular
```

6.6.21 Lesson 4: Advanced Time Manipulation Routines

6.6.22 Task Statement

Demonstrate the advanced functions of the time utilities with regard to formatting of time strings for output. Formatting options include altering calendar representations of the time strings. Convert time-date strings between different SpiceyPy-supported formats.

6.6.23 Learning Goals

Introduce the routines used for advanced manipulation of time strings. Understand the concept of ephemeris time (ET) as used in SpiceyPy.

6.6.24 Code Solution

Caution: Be sure to assign sufficient string lengths for time formats/pictures.

```
from __future__ import print_function

#
# Import the CSPICE-Python interface.
#
import spiceypy

def xtic():

    #
    # Assign the META variable to the name of the meta-kernel
    # that contains the LSK kernel and create an arbitrary
    # time string.
    #
    CALSTR      = 'Mar 15, 2003 12:34:56.789 AM PST'
    META         = 'xtic.tm'
    AMBIGSTR     = 'Mar 15, 79 12:34:56'
    T_FORMAT1    = 'Wkd Mon DD HR:MN:SC PDT YYYY ::UTC-7'
    T_FORMAT2    = 'Wkd Mon DD HR:MN ::UTC-7 YR (JULIAND.#### JDUTC) '

    #
    # Load the meta-kernel.
    #
    spiceypy.furnsh( META )
    print( 'Original time string      : {0}'.format( CALSTR ) )

    #
    # Convert the time string to the number of ephemeris
    # seconds past the J2000 epoch. This is the most common
    # internal time representation used by the CSPICE
    # system; CSPICE refers to this as ephemeris time (ET).
    #
    et = spiceypy.str2et( CALSTR )
    print( 'Corresponding ET          : {0:20.6f}\n'.format(et) )

    #
    # Make a picture of an output format. Describe a Unix-like
    # time string then send the picture and the 'et' value through
    # spiceypy.timeout to format and convert the ET representation
    # of the time string into the form described in
    # spiceypy.timeout. The '::UTC-7' token indicates the time
    # zone for the 'timstr' output - PDT. 'PDT' is part of the
    # output, but not a time system token.
    #
    timstr = spiceypy.timeout( et, T_FORMAT1 )
    print( 'Time in string format 1   : {0}'.format(timstr) )
```

(continues on next page)

(continued from previous page)

```

timstr = spiceypy.timeout( et, T_FORMAT2)
print( 'Time in string format 2 : {0}'.format(timstr) )

#
# Why create a picture by hand when spiceypy can do it for
# you? Input a string to spiceypy.tpictr with the format of
# interest. `ok' returns a boolean indicating whether an
# error occurred while parsing the picture string, if so,
# an error diagnostic message returns in 'xerror'. In this
# example the picture string is known as correct.
#
pic = '12:34:56.789 P.M. PDT January 1, 2006'
[ pictr, ok, xerror] = spiceypy.tpictr(pic)

if not bool(ok):
    print( xerror )
    exit

timstr = spiceypy.timeout( et, pictr)
print( 'Time in string format 3 : {0}'.format( timstr ) )

#
# Two digit year representations often cause problems due to
# the ambiguity of the century. The routine spiceypy.tsetyr
# gives the user the ability to set a default range for 2
# digit year representation. SPICE uses 1969AD as the default
# start year so the numbers inclusive of 69 to 99 represent
# years 1969AD to 1999AD, the numbers inclusive of 00 to 68
# represent years 2000AD to 2068AD.
#
# The defined time string 'AMBIGSTR' contains a two-digit
# year. Since the SPICE base year is 1969, the time subsystem
# interprets the string as 1979.
#
et1 = spiceypy.str2et( AMBIGSTR )

#
# Set 1980 as the base year causes SPICE to interpret the
# time string's "79" as 2079.
#
spiceypy.tsetyr( 1980 )
et2 = spiceypy.str2et( AMBIGSTR )

#
# Calculate the number of years between the two ET
# representations, ~100.
#
print( 'Years between evaluations: {0:20.6f}'.\
format( (et2 - et1)/spiceypy.jyear()) )

#
# Reset the default year to 1969.
#
spiceypy.tsetyr( 1969 )

#

```

(continues on next page)

(continued from previous page)

```

# Done. Unload the kernels.
#
spiceypy.kclear

if __name__ == '__main__':
    xtic()

```

Run the code example

```

Original time string      : Mar 15, 2003 12:34:56.789 AM PST
Corresponding ET         : 100989360.974561

Time in string format 1  : Sat Mar 15 01:34:56 PDT 2003
Time in string format 2  : Sat Mar 15 01:34 03 (2452713.85760 JDUTC)
Time in string format 3  : 01:34:56.789 A.M. PDT March 15, 2003
Years between evaluations: 100.000000

```

6.6.25 Lesson 5: Error Handling

6.6.26 Task Statement

Write an interactive program to return a state vector based on a user’s input. Code the program with the capability to recover from user input mistakes, inform the user of the mistake, then continue to run.

6.6.27 Learning Goals

Learn how to write a program that has the capability to recover from expected SPICE errors.

The SpiceyPy error subsystem differs from CSPICE and SPICELIB packages in that the user cannot alter the state of the error subsystem, rather the user can respond to an error signal using try-except blocks. This block natively receives and processes any SpiceyError exception signaled from SpiceyPy. The user can therefore “catch” an error signal so as to respond in an appropriate manner.

6.6.28 Code Solution

```

from __future__ import print_function
from builtins import input

#
# Import the CSPICE-Python interface.
#
import spiceypy
from spiceypy.utils.support_types import SpiceyError

def aderr():

    #
    # Set initial parameters.
    #
    SPICETRUE = True

```

(continues on next page)

(continued from previous page)

```

SPICEFALSE=  False
doloop      =  SPICETRUE

#
# Load the data we need for state evaluation.
#
spiceypy.furnsh( 'aderr.tm' )

#
# Start our input query loop to the user.
#
while (doloop):

    #
    # For simplicity, we request only one input.
    # The program calculates the state vector from
    # Earth to the user specified target 'targ' in the
    # J2000 frame, at ephemeris time zero, using
    # aberration correction LT+S (light time plus
    # stellar aberration).
    #
    targ = input( 'Target: ' )

    if targ == 'NONE':
        #
        # An exit condition. If the user inputs NONE
        # for a target name, set the loop to stop...
        #
        doloop = SPICEFALSE

    else:

        #
        # ...otherwise evaluate the state between the Earth
        # and the target. Initialize an error handler.
        #
        try:

            #
            # Perform the state lookup.
            #
            [state, ltime] = spiceypy.spkezr(targ, 0., 'J2000',
                                           'LT+S', 'EARTH')

            #
            # No error, output the state.
            #
            print( 'R : {0:20.6f} {1:20.6f} '
                  '{2:20.5f}'.format(*state[0:3]))
            print( 'V : {0:20.6f} {1:20.6f} '
                  '{2:20.6f}'.format(*state[3:6]) )
            print( 'LT: {0:20.6f}\n'.format(float(ltime)))

        except SpiceyError as err:

            #

```

(continues on next page)

(continued from previous page)

```

        # What if spiceypy.spkezr signaled an error?
        # Then spiceypy signals an error to python.
        #
        # Examine the value of 'e' to retrieve the
        # error message.
        #
        print( err )
        print( )

        #
        # Done. Unload the kernels.
        #
        spiceypy.kclear

if __name__ == '__main__':
    aderr()

```

Run the code example

Now run the code with various inputs to observe behavior. Begin the run using known astronomical bodies, e.g. “Moon”, “Mars”, “Pluto barycenter” and “Puck”. Recall the SpiceyPy default units are kilometers, kilometers per second, kilograms, and seconds. The ‘R’ marker identifies the (X,Y,Z) position of the body in kilometers, the ‘V’ marker identifies the velocity of the body in kilometers per second, and the ‘LT’ marker identifies the one-way light time between the bodies at the requested evaluation time.

```

Target: Moon
R :      -291584.616595      -266693.402359      -76095.64756
V :           0.643439          -0.666066          -0.301310
LT:           1.342311

Target: Mars
R :      234536077.419136      -132584383.595569      -63102685.70619
V :           30.961373           28.932996           13.113031
LT:           923.001080

Target: Pluto barycenter
R :      -1451304742.838526      -4318174144.406321      -918251433.58736
V :           35.079843           3.053138           -0.036762
LT:           15501.258293

Target: Puck

=====
=====

Toolkit version: N0066

SPICE(SPKINSUFFDATA) --

Insufficient ephemeris data has been loaded to compute the state of 7
15 (PUCK) relative to 0 (SOLAR SYSTEM BARYCENTER) at the ephemeris ep
och 2000 JAN 01 12:00:00.000.

spkezr_c --> SPKEZR --> SPKEZ --> SPKACS --> SPKAPS --> SPKLTC --> SP
KGEO

```

(continues on next page)

(continued from previous page)

```
=====
=====
Target:
```

Perplexing. What happened?

The kernel files named in meta.tm did not include ephemeris data for Puck. When the SPK subsystem tried to evaluate Puck’s position, the evaluation failed due to lack of data, so an error signaled.

The above error signifies an absence of state information at ephemeris time 2000 JAN 01 12:00:00.000 (the requested time, ephemeris time zero).

Try another look-up, this time for “Casper”

```
Target: Casper

=====
=====

Toolkit version: N0066

SPICE(IDCODENOTFOUND) --

The target, 'Casper', is not a recognized name for an ephemeris object. The cause of this problem may be that you need an updated version of the SPICE Toolkit. Alternatively you may call SPKEZ directly if you know the SPICE ID codes for both 'Casper' and 'EARTH'

spkezs_c --> SPKEZR

=====
=====

Target:
```

An easy to understand error. The SPICE system does not contain information on a body named ‘Casper.’

Another look-up, this time, “Venus”.

```
Target: Venus
R :      -80970027.540532      -139655772.573898      -53860125.95820
V :           31.166910           -27.001056           -12.316514
LT:           567.655074

Target:
```

The look-up succeeded despite two errors in our run. The SpiceyPy system can respond to error conditions (not system errors) in much the same fashion as languages with catch/throw instructions.

6.6.29 Lesson 6: Windows, and Cells

6.6.30 Programming task

Given the times of line-of-sight for a vehicle from a ground station and the times for an acceptable Sun-station-vehicle phase angle, write a program to determine the time intervals common to both configurations.

6.6.31 Learning Goals

SpiceyPy implementation of SPICE cells consists of a class that provides an interface to the underlying CSPICE cell structure.

A user should create cells by use of the appropriate SpiceyPy calls. NAIF recommends against manual creation of cells.

A ‘window’ is a type of cell containing ordered, double precision values describing a collection of zero or more intervals.

We define an interval, ‘i’, as all double precision values bounded by and including an ordered pair of numbers,

```
[ a , b ]
  i   i
```

where

```
a    <    b
  i    -    i
```

The intervals within a window are both ordered and disjoint. That is, the beginning of each interval is greater than the end of the previous interval:

```
b    <    a
  i        i+1
```

A common use of the windows facility is to calculate the intersection set of a number of time intervals.

6.6.32 Code Solution

```
from __future__ import print_function

#
# Import the CSPICE-Python interface.
#
import spiceypy

def win():

    MAXSIZ = 8

    #
    # Define a set of time intervals. For the purposes of this
    # tutorial program, define time intervals representing
    # an unobscured line of sight between a ground station
    # and some body.
    #
```

(continues on next page)

(continued from previous page)

```

los = [ 'Jan 1, 2003 22:15:02', 'Jan 2, 2003  4:43:29',
        'Jan 4, 2003  9:55:30', 'Jan 4, 2003 11:26:52',
        'Jan 5, 2003 11:09:17', 'Jan 5, 2003 13:00:41',
        'Jan 6, 2003 00:08:13', 'Jan 6, 2003  2:18:01' ]

#
# A second set of intervals representing the times for which
# an acceptable phase angle exists between the ground station,
# the body and the Sun.
#
phase = [ 'Jan 2, 2003 00:03:30', 'Jan 2, 2003 19:00:00',
          'Jan 3, 2003  8:00:00', 'Jan 3, 2003  9:50:00',
          'Jan 5, 2003 12:00:00', 'Jan 5, 2003 12:45:00',
          'Jan 6, 2003 00:30:00', 'Jan 6, 2003 23:00:00' ]

#
# Load our meta kernel for the leapseconds data.
#
spiceypy.furnsh( 'win.tm' )

#
# SPICE windows consist of double precision values; convert
# the string time tags defined in the 'los' and 'phase'
# arrays to double precision ET. Store the double values
# in the 'loswin' and 'phswin' windows.
#
los_et = spiceypy.str2et( los )
phs_et = spiceypy.str2et( phase )

loswin = spiceypy.stypes.SPICEDOUBLE_CELL( MAXSIZ )
phswin = spiceypy.stypes.SPICEDOUBLE_CELL( MAXSIZ )

for i in range(0, int( MAXSIZ/2 ) ):
    spiceypy.wninsd( los_et[2*i], los_et[2*i+1], loswin )
    spiceypy.wninsd( phs_et[2*i], phs_et[2*i+1], phswin )

spiceypy.wnvald( MAXSIZ, MAXSIZ, loswin )
spiceypy.wnvald( MAXSIZ, MAXSIZ, phswin )

#
# The issue for consideration, at what times do line of
# sight events coincide with acceptable phase angles?
# Perform the set operation AND on loswin, phswin,
# (the intersection of the time intervals)
# place the results in the window 'sched'.
#
sched = spiceypy.wnintd( loswin, phswin )

print( 'Number data values in sched : '
        '{0:2d}'.format( spiceypy.card(sched) ) )

#
# Output the results. The number of intervals in 'sched'
# is half the number of data points (the cardinality).
#
print( ' ' )
print( 'Time intervals meeting defined criterion.' )

```

(continues on next page)

(continued from previous page)

```

for i in range( spiceypy.card(sched)//2):

    #
    # Extract from the derived 'sched' the values defining the
    # time intervals.
    #
    [left, right ] = spiceypy.wnfetd( sched, i )

    #
    # Convert the ET values to UTC for human comprehension.
    #
    utcstr_l = spiceypy.et2utc( left , 'C', 3 )
    utcstr_r = spiceypy.et2utc( right, 'C', 3 )

    #
    # Output the UTC string and the corresponding index
    # for the interval.
    #
    print( '{0:2d}    {1}    {2}'.format(i, utcstr_l, utcstr_r))

#
# Summarize the 'sched' window.
#
[meas, avg, stddev, small, large] = spiceypy.wnsumd( sched )

print( '\nSummary of sched window\n' )

print( 'o Total measure of sched      : {0:16.6f}'.format(meas))
print( 'o Average measure of sched    : {0:16.6f}'.format(avg))
print( 'o Standard deviation of ' )
print( '   the measures in sched        : '
      '{0:16.6f}'.format(stddev))

#
# The values for small and large refer to the indexes of the
# values in the window ('sched'). The shortest interval is
#
#      [ sched.base[ sched.data + small]
#        sched.base[ sched.data + small +1]  ];
#
# the longest is
#
#      [ sched.base[ sched.data + large]
#        sched.base[ sched.data + large +1]  ];
#
# Output the interval indexes for the shortest and longest
# intervals. As Python bases an array index on 0, the interval
# index is half the array index.
#
print( 'o Index of shortest interval: '
      '{0:2d}'.format(int(small/2)) )
print( 'o Index of longest interval : '
      '{0:2d}'.format(int(large/2)) )

#

```

(continues on next page)

(continued from previous page)

```
# Done. Unload the kernels.
#
spiceypy.kclear

if __name__ == '__main__':
    win()
```

Run the code example

The output window has the name `sched` (schedule).

Output the amount of data held in `sched` compared to the maximum possible amount.

```
Number data values in sched : 6
```

List the time intervals for which a line of sight exists during the time of a proper phase angle.

```
Time intervals meeting defined criterion.
0  2003 JAN 02 00:03:30.000  2003 JAN 02 04:43:29.000
1  2003 JAN 05 12:00:00.000  2003 JAN 05 12:45:00.000
2  2003 JAN 06 00:30:00.000  2003 JAN 06 02:18:01.000
```

Finally, an analysis of the `sched` data. The measure of an interval [a,b] ($a \leq b$) equals $b-a$. Real values output in units of seconds.

```
Summary of sched window

o Total measure of sched      : 25980.000009
o Average measure of sched   : 8660.000003
o Standard deviation of
  the measures in sched      : 5958.550217
o Index of shortest interval: 1
o Index of longest interval : 0
```

6.6.33 Related Routines

```
-- spiceypy.wncomd determines the compliment of a window with
   respect to a defined interval.

-- spiceypy.wncond contracts a window's intervals.

-- spiceypy.wndifd : Calculate the difference between two windows;
   i.e. every point existing in the first but not the second.

-- spiceypy.wnelmd returns TRUE or FALSE if a value exists in a
   window.

-- spiceypy.wnexpd expands the size of the intervals in a window.

-- spiceypy.wnextd extracts a window's endpoints .

-- spiceypy.wnfiltd fills gaps between intervals in a window.

-- spiceypy.wnfltd filter/removes small intervals from a window.
```

(continues on next page)

(continued from previous page)

```
--  spiceypy.wnincd determines if an interval exists within a
    window.

--  spiceypy.wninsd inserts an interval into a window.

--  spiceypy.wnreld compares two windows. Comparison operations
    available, equality '=', inequality '<>', subset '<=' and '>=',
    proper subset '<' and '>'.

--  spiceypy.wnunid calculates the union of two windows.
```

6.6.34 Lesson 7: Utility and Constants Routines

6.6.35 Task Statement

Write an interactive program to convert values between various units. Demonstrate the flexibility of the unit conversion routine, the string equality function, and show the version ID function.

6.6.36 Learning Goals

SpiceyPy provides several routines to perform commonly needed tasks. Among these:

SpiceyPy also includes a set of functions that return constant values often used in astrodynamics, time calculations, and geometry.

6.6.37 Code Solution

```
from __future__ import print_function
from builtins import input

#
# Import the CSPICE-Python interface.
#
import spiceypy

def tostan(alias):

    value = alias

    #
    # As a convenience, let's alias a few common terms
    # to their appropriate counterpart.
    #
    if alias == 'meter':

        #
        # First, a 'meter' by any other name is a
        # 'METER' and smells as sweet ...
        #
        value = 'METERS'
```

(continues on next page)

(continued from previous page)

```

elif (alias == 'klicks') \
    or (alias == 'kilometers') \
    or (alias == 'kilometer'):

    #
    # ... 'klicks' and 'KILOMETERS' and 'KILOMETER'
    # identifies 'KM'....
    #
    value = 'KM'

elif alias == 'secs':

    #
    # ... 'secs' to 'SECONDS'.
    #
    value = 'SECONDS'

elif alias == 'miles':

    #
    # ... and finally 'miles' to 'STATUTE_MILES'.
    # Normal people think in statute miles.
    # Only sailors think in nautical miles - one
    # minute of arc at the equator.
    #
    value = 'STATUTE_MILES'

else:
    pass

#
# Much better. Now return. If the input matched
# none of the aliases, this function did nothing.
#
return value

def units():

    #
    # Display the Toolkit version string with a spiceypy.tkvrns
    # call.
    #
    vers = spiceypy.tkvrns( 'TOOLKIT' )
    print('\nConvert demo program compiled against CSPICE '
          'Toolkit ' + vers)

    #
    # The user first inputs the name of a unit of measure.
    # Send the name through tostan for de-aliasing.
    #
    funits = input( 'From Units : ' )
    funits = tostan( funits )

    #
    # Input a double precision value to express in a new
    # unit format.

```

(continues on next page)

(continued from previous page)

```
#
fvalue = float(input( 'From Value : ' ))

#
# Now the user inputs the name of the output units.
# Again we send the units name through tostan for
# de-aliasing.
#
tunits = input( 'To Units : ' )
tunits = tostan( tunits )

tvalue = spiceypy.convrt( fvalue, funits, tunits)
print( '{0:12.5f} {1}'.format(tvalue, tunits) )

if __name__ == '__main__':
    units()
```

Run the code example

Run a few conversions through the application to ensure it works. The intro banner gives us the Toolkit version against which the application was linked:

```
Convert demo program compiled against CSPICE Toolkit CSPICE_N0066
From Units : klicks
From Value : 3
To Units : miles
    1.86411 STATUTE_MILES
```

Now we know. Three kilometers equals 1.864 miles.

Legend states Pheidippides ran from the Marathon Plain to Athens. The modern marathon race (inspired by this event) spans 26.2 miles. How far in kilometers?

```
Convert demo program compiled against CSPICE Toolkit CSPICE_N0066
From Units : miles
From Value : 26.2
To Units : km
    42.16481 km
```

6.6.38 Task Statement

Write a program to output SpiceyPy constants and use those constants to calculate some rudimentary values.

6.6.39 Code Solution

```
from __future__ import print_function

#
# Import the CSPICE-Python interface.
#
import spiceypy

def xconst():
```

(continues on next page)

(continued from previous page)

```

#
# All the function have the same calling sequence:
#
#     VALUE = function_name()
#
#     some_procedure( function_name() )
#
# First a simple example using the seconds per day
# constant...
#
print( 'Number of (S)econds (P)er (D)ay          : '
      '{0:19.12f}'.format(spiceypy.spd() ) )

#
# ...then show the value of degrees per radian, 180/Pi...
#
print( 'Number of (D)egrees (P)er (R)adian       : '
      '{0:19.16f}'.format(spiceypy.dpr() ) )

#
# ...and the inverse, radians per degree, Pi/180.
# It is obvious spiceypy.dpr() equals 1.d/spiceypy.rpd(), or
# more simply spiceypy.dpr() * spiceypy.rpd() equals 1
#
print( 'Number of (R)adians (P)er (D)egree       : '
      '{0:19.16f}'.format(spiceypy.rpd() ) )

#
# What's the value for the astrophysicist's favorite
# physical constant (in a vacuum)?
#
print( 'Speed of light in KM per second         : '
      '{0:19.12f}'.format(spiceypy.clight() ) )

#
# How long (in Julian days) from the J2000 epoch to the
# J2100 epoch?
#
print( 'Number of days between epochs J2000')
print( '   and J2100                          : '
      '{0:19.12f}'.format( spiceypy.j2100()
                          - spiceypy.j2000() ) )

#
# Redo the calculation returning seconds...
#
print( 'Number of seconds between epochs' )
print( '   J2000 and J2100                    : '
      '{0:19.5f}'.format(spiceypy.spd() *
                        (spiceypy.j2100() - spiceypy.j2000() ) ) )

#
# ...then tropical years.
#
val =(spiceypy.spd()/spiceypy.tyear()          ) *
      (spiceypy.j2100()- spiceypy.j2000() ) )

```

(continues on next page)

(continued from previous page)

```

print( 'Number of tropical years between' )
print( '   epochs J2000 and J2100           : '
      '{0:19.12f}'.format(val))

#
# Finally, how can I convert a radian value to degrees.
#
print( 'Number of degrees in Pi/2 radians of arc: '
      '{0:19.16f}'.format( spiceypy.halfpi()
                          * spiceypy.dpr() ))

#
# and degrees to radians.
#
print( 'Number of radians in 250 degrees of arc : '
      '{0:19.16f}'.format(250. * spiceypy.rpd() ))

if __name__ == '__main__':
    xconst()

```

Run the code example

```

Number of (S)econds (P)er (D)ay           : 86400.000000000000
Number of (D)egrees (P)er (R)adian        : 57.2957795130823229
Number of (R)adians (P)er (D)egree        : 0.0174532925199433
Speed of light in KM per second           : 299792.457999999984
Number of days between epochs J2000
and J2100                                 : 36525.000000000000
Number of seconds between epochs
J2000 and J2100                           : 3155760000.00000
Number of tropical years between
epochs J2000 and J2100                    : 100.002135902909
Number of degrees in Pi/2 radians of arc: 90.0000000000000000
Number of radians in 250 degrees of arc : 4.3633231299858242

```

6.6.40 Related Routines

```

--  spiceypy.b1900 : Julian Date of the epoch Besselian Date 1900.0
--
--  spiceypy.b1950 : Julian date of the epoch Besselian Date 1950.0
--
--  spiceypy.j1900 : Julian date of 1900 JAN 0.5 this corresponds
to calendar date 1899 DEC 31 12:00:00
--
--  spiceypy.j1950 : Julian date of 1950 JAN 1.0 this corresponds
to calendar date 1950 JAN 01 00:00:00
--
--  spiceypy.twopi : double precision value of 2 * Pi
--
--  spiceypy.pi : double precision value of Pi
--
--  spiceypy.jyear : seconds per Julian year (365.25 Julian days)

```

SPICEYPY PACKAGE

7.1 spiceypy module

The MIT License (MIT)

Copyright (c) [2015-2019] [Andrew Annex]

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

`spiceypy.spiceypy.appendc(item, cell)`

Append an item to a character cell.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/appndc_c.html

Parameters

- **item** (Union[str, Iterable[str], ndarray, str_]) – The item to append.
- **cell** (Union[Cell_Char, SpiceCell]) – The cell to append to.

Return type None

`spiceypy.spiceypy.appendd(item, cell)`

Append an item to a double precision cell.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/appndd_c.html

Parameters

- **item** (Union[int, float, Iterable[float]]) – The item to append.
- **cell** (Union[SpiceCell, Cell_Double]) – The cell to append to.

Return type None

`spiceypy.spiceypy.appendi(item, cell)`

Append an item to an integer cell.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/appndi_c.html

Parameters

- **item** (Union[Iterable[int], int]) – The item to append.
- **cell** (Union[*SpiceCell*, *Cell_Int*]) – The cell to append to.

Return type None

`spiceypy.spiceypy.axisar(axis, angle)`

Construct a rotation matrix that rotates vectors by a specified angle about a specified axis.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/axisar_c.html

Parameters

- **axis** (Union[ndarray, Iterable[float]]) – Rotation axis.
- **angle** (float) – Rotation angle, in radians.

Return type ndarray

Returns Rotation matrix corresponding to axis and angle.

`spiceypy.spiceypy.b1900()`

Return the Julian Date corresponding to Besselian Date 1900.0.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/b1900_c.html

Return type float

Returns The Julian Date corresponding to Besselian Date 1900.0.

`spiceypy.spiceypy.b1950()`

Return the Julian Date corresponding to Besselian Date 1950.0.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/b1950_c.html

Return type float

Returns The Julian Date corresponding to Besselian Date 1950.0.

`spiceypy.spiceypy.badkpvcaller, name, comp, insize, divby, intype)`

Determine if a kernel pool variable is present and if so that it has the correct size and type.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/badkpvc_c.html

Parameters

- **caller** (str) – Name of the routine calling this routine.
- **name** (str) – Name of a kernel pool variable.
- **comp** (str) – Comparison operator.
- **insize** (int) – Expected size of the kernel pool variable.
- **divby** (int) – A divisor of the size of the kernel pool variable.
- **intype** (str) – Expected type of the kernel pool variable

Return type bool

Returns returns false if the kernel pool variable is OK.

`spiceypy.spiceypy.bltfm(frmcls, out_cell=None)`

Return a SPICE set containing the frame IDs of all built-in frames of a specified class.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bltfm_c.html

Parameters

- **frmcls** (int) – Frame class.
- **out_cell** (Optional[*SpiceCell*]) – Optional SpiceInt Cell that is returned

Return type *SpiceCell*

Returns Set of ID codes of frames of the specified class.

`spiceypy.spiceypy.bodc2n(code, lenout=256)`

Translate the SPICE integer code of a body into a common name for that body.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bodc2n_c.html

Parameters

- **code** (int) – Integer ID code to be translated into a name.
- **lenout** (int) – Maximum length of output name.

Return type Tuple[str, bool]

Returns A common name for the body identified by code.

`spiceypy.spiceypy.bodc2s(code, lenout=256)`

Translate a body ID code to either the corresponding name or if no name to ID code mapping exists, the string representation of the body ID value.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bodc2s_c.html

Parameters

- **code** (int) – Integer ID code to translate to a string.
- **lenout** (int) – Maximum length of output name.

Return type str

Returns String corresponding to ‘code’.

`spiceypy.spiceypy.boddef(name, code)`

Define a body name/ID code pair for later translation via *bodn2c()* or *bodc2n()*.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/boddef_c.html

Parameters

- **name** (str) – Common name of some body.
- **code** (int) – Integer code for that body.

Return type None

`spiceypy.spiceypy.bodfnd(body, item)`

Determine whether values exist for some item for any body in the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bodfnd_c.html

Parameters

- **body** (int) – ID code of body.
- **item** (str) – Item to find (“RADII”, “NUT_AMP_RA”, etc.).

Return type `bool`

Returns True if the item is in the kernel pool, and is False if it is not.

`spiceypy.spiceypy.bodn2c(name)`

Translate the name of a body or object to the corresponding SPICE integer ID code.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bodn2c_c.html

Parameters `name` (`str`) – Body name to be translated into a SPICE ID code.

Return type `Tuple[int, bool]`

Returns SPICE integer ID code for the named body.

`spiceypy.spiceypy.bods2c(name)`

Translate a string containing a body name or ID code to an integer code.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bods2c_c.html

Parameters `name` (`str`) – String to be translated to an ID code.

Return type `Tuple[int, bool]`

Returns Integer ID code corresponding to name.

`spiceypy.spiceypy.bodvar(body, item, dim)`

Deprecated: This routine has been superseded by `bodvcd()` and `bodvrd()`. This routine is supported for purposes of backward compatibility only.

Return the values of some item for any body in the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bodvar_c.html

Parameters

- `body` (`int`) – ID code of body.
- `item` (`str`) – Item for which values are desired, (“RADII”, “NUT_PREC_ANGLES”, etc.)
- `dim` (`int`) – Number of values returned.

Return type `ndarray`

Returns values

`spiceypy.spiceypy.bodvcd(bodyid, item, maxn)`

Fetch from the kernel pool the double precision values of an item associated with a body, where the body is specified by an integer ID code.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bodvcd_c.html

Parameters

- `bodyid` (`int`) – Body ID code.
- `item` (`str`) – Item for which values are desired, (“RADII”, “NUT_PREC_ANGLES”, etc.)
- `maxn` (`int`) – Maximum number of values that may be returned.

Return type `Tuple[int, ndarray]`

Returns `dim`, values

`spiceypy.spiceypy.bodvrd(bodynm, item, maxn)`

Fetch from the kernel pool the double precision values of an item associated with a body.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bodvrd_c.html

Parameters

- **bodynm** (str) – Body name.
- **item** (str) – Item for which values are desired, (“RADII”, “NUT_PREC_ANGLES”, etc.)
- **maxn** (int) – Maximum number of values that may be returned.

Return type Tuple[int, ndarray]

Returns tuple of (dim, values)

`spiceypy.spiceypy.brcktd(number, end1, end2)`

Bracket a number. That is, given a number and an acceptable interval, make sure that the number is contained in the interval. (If the number is already in the interval, leave it alone. If not, set it to the nearest endpoint of the interval.)

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/brcktd_c.html

Parameters

- **number** (float) – Number to be bracketed.
- **end1** (float) – One of the bracketing endpoints for number.
- **end2** (float) – The other bracketing endpoint for number.

Return type float

Returns value within an interval

`spiceypy.spiceypy.brckti(number, end1, end2)`

Bracket a number. That is, given a number and an acceptable interval, make sure that the number is contained in the interval. (If the number is already in the interval, leave it alone. If not, set it to the nearest endpoint of the interval.)

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/brckti_c.html

Parameters

- **number** (int) – Number to be bracketed.
- **end1** (int) – One of the bracketing endpoints for number.
- **end2** (int) – The other bracketing endpoint for number.

Return type int

Returns value within an interval

`spiceypy.spiceypy.bschoch(value, ndim, lenvals, array, order)`

Do a binary search for a given value within a character string array, accompanied by an order vector. Return the index of the matching array entry, or -1 if the key value is not found.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bschoch_c.html

Parameters

- **value** (Union[str_, str]) – Key value to be found in array.
- **ndim** (int) – Dimension of array.

- **lenvals** (*int*) – String length.
- **array** (*Union[ndarray, Iterable[str]]*) – Character string array to search.
- **order** (*Iterable[int]*) – Order vector.

Return type *int*

Returns *index*

`spiceypy.spiceypy.bschoi (value, ndim, array, order)`

Do a binary search for a given value within an integer array, accompanied by an order vector. Return the index of the matching array entry, or -1 if the key value is not found.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bschoi_c.html

Parameters

- **value** (*int*) – Key value to be found in array.
- **ndim** (*int*) – Dimension of array.
- **array** (*Iterable[int]*) – Integer array to search.
- **order** (*Iterable[int]*) – Order vector.

Return type *int*

Returns *index*

`spiceypy.spiceypy.bsrchc (value, ndim, lenvals, array)`

Do a binary search for a given value within a character string array. Return the index of the first matching array entry, or -1 if the key value was not found.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bsrchc_c.html

Parameters

- **value** (*str*) – Key value to be found in array.
- **ndim** (*int*) – Dimension of array.
- **lenvals** (*int*) – String length.
- **array** (*Iterable[str]*) – Character string array to search.

Return type *int*

Returns *index*

`spiceypy.spiceypy.bsrchd (value, ndim, array)`

Do a binary search for a key value within a double precision array, assumed to be in increasing order. Return the index of the matching array entry, or -1 if the key value is not found.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bsrchd_c.html

Parameters

- **value** (*float*) – Value to find in array.
- **ndim** (*int*) – Dimension of array.
- **array** (*ndarray*) – Array to be searched.

Return type *int*

Returns *index*

`spiceypy.spiceypy.bsrch`(*value*, *ndim*, *array*)

Do a binary search for a key value within an integer array, assumed to be in increasing order. Return the index of the matching array entry, or -1 if the key value is not found.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/bsrch_c.html

Parameters

- **value** (*int*) – Value to find in array.
- **ndim** (*int*) – Dimension of array.
- **array** (*ndarray*) – Array to be searched.

Return type *int*

Returns *index*

`spiceypy.spiceypy.card`(*cell*)

Return the cardinality (current number of elements) in a cell of any data type.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/card_c.html

Parameters **cell** (*SpiceCell*) – Input cell.

Return type *int*

Returns the number of elements in a cell of any data type.

`spiceypy.spiceypy.ccifrm`(*frclass*, *classid*, *lenout*=256)

Return the frame name, frame ID, and center associated with a given frame class and class ID.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ccifrm_c.html

Parameters

- **frclass** (*int*) – Class of frame.
- **classid** (*int*) – Class ID of frame.
- **lenout** (*int*) – Maximum length of output string.

Return type *Tuple[int, str, int, bool]*

Returns the frame name, frame ID, center.

`spiceypy.spiceypy.cell_bool`(*cell_size*)

Return type *SpiceCell*

`spiceypy.spiceypy.cell_char`(*cell_size*, *length*)

Return type *SpiceCell*

`spiceypy.spiceypy.cell_double`(*cell_size*)

Return type *SpiceCell*

`spiceypy.spiceypy.cell_int`(*cell_size*)

Return type *SpiceCell*

`spiceypy.spiceypy.cell_time`(*cell_size*)

`spiceypy.spiceypy.cg2el`(*center*, *vec1*, *vec2*)

Form a SPICE ellipse from a center vector and two generating vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cg2el_c.html

Parameters

- **center** (Iterable[float]) – Center Vector
- **vec1** (Union[ndarray, Iterable[float]]) – Vector 1
- **vec2** (Union[ndarray, Iterable[float]]) – Vector 2

Return type *Ellipse*

Returns Ellipse

`spiceypy.spiceypy.chbder(cp, degp, x2s, x, nderiv)`

Given the coefficients for the Chebyshev expansion of a polynomial, this returns the value of the polynomial and its first nderiv derivatives evaluated at the input X.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/chbder_c.html

Parameters

- **cp** (Iterable[float]) – degp+1 Chebyshev polynomial coefficients.
- **degp** (int) – Degree of polynomial.
- **x2s** (Iterable[float]) – Transformation parameters of polynomial.
- **x** (float) – Value for which the polynomial is to be evaluated
- **nderiv** (int) – The number of derivatives to compute

Return type ndarray

Returns Array of the derivatives of the polynomial

`spiceypy.spiceypy.check_for_spice_error(f)`

Internal function to check :type f: Optional[Callable] :param f: :raise stypes.SpiceError:

Return type None

`spiceypy.spiceypy.chkkin(module)`

Inform the SPICE error handling mechanism of entry into a routine.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/chkin_c.html

Parameters **module** (str) – The name of the calling routine.

Return type None

`spiceypy.spiceypy.chkout(module)`

Inform the SPICE error handling mechanism of exit from a routine.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/chkout_c.html

Parameters **module** (str) – The name of the calling routine.

Return type None

`spiceypy.spiceypy.cidfrm(cent, lenout=256)`

Retrieve frame ID code and name to associate with a frame center.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cidfrm_c.html

Parameters

- **cent** (int) – An object to associate a frame with.
- **lenout** (int) – Available space in output string fname.

Return type Tuple[int, str, bool]

Returns frame ID code, name to associate with a frame center.

`spiceypy.spiceypy.ckcls(handle)`

Close an open CK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckcls_c.html

Parameters `handle` (`int`) – Handle of the CK file to be closed.

Return type `None`

`spiceypy.spiceypy.ckcov(ck, idcode, needav, level, tol, timsys, cover=None)`

Find the coverage window for a specified object in a specified CK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckcov_c.html

Parameters

- **ck** (`str`) – Name of CK file.
- **idcode** (`int`) – ID code of object.
- **needav** (`bool`) – Flag indicating whether angular velocity is needed.
- **level** (`str`) – Coverage level: (SEGMENT OR INTERVAL)
- **tol** (`float`) – Tolerance in ticks.
- **timsys** (`str`) – Time system used to represent coverage.
- **cover** (Optional[*SpiceCell*]) – Window giving coverage for idcode.

Return type *SpiceCell*

Returns coverage window for a specified object in a specified CK file

`spiceypy.spiceypy.ckfrot(inst, et)`

Find the rotation from a C-kernel Id to the native frame at the time requested.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/spicelib/ckfrot.html

Parameters

- **inst** (`int`) – NAIF instrument ID
- **et** (`float`) – Epoch measured in seconds past J2000

Return type `Tuple[ndarray, int, bool]`

Returns Rotation matrix from the input frame to the returned reference frame, id for the reference frame

`spiceypy.spiceypy.ckgpp(inst, sclkdp, tol, ref)`

Get pointing (attitude) for a specified spacecraft clock time.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckgpp_c.html

Parameters

- **inst** (`int`) – NAIF ID of instrument, spacecraft, or structure.
- **sclkdp** (`Union[float, int]`) – Encoded spacecraft clock time.
- **tol** (`int`) – Time tolerance.
- **ref** (`str`) – Reference frame.

Return type `Tuple[ndarray, float, bool]`

Returns C-matrix pointing data, Output encoded spacecraft clock time

`spiceypy.spiceypy.ckgpav (inst, sclkdp, tol, ref)`

Get pointing (attitude) and angular velocity for a specified spacecraft clock time.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckgpav_c.html

Parameters

- **inst** (`int`) – NAIF ID of instrument, spacecraft, or structure.
- **sclkdp** (`float`) – Encoded spacecraft clock time.
- **tol** (`Union[float, int]`) – Time tolerance.
- **ref** (`str`) – Reference frame.

Return type `Tuple[ndarray, ndarray, float, bool]`

Returns C-matrix pointing data, Angular velocity vector, Output encoded spacecraft clock time.

`spiceypy.spiceypy.cklpf (filename)`

Load a CK pointing file for use by the CK readers. Return that file's handle, to be used by other CK routines to refer to the file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cklpf_c.html

Parameters **filename** (`str`) – Name of the CK file to be loaded.

Return type `int`

Returns Loaded file's handle.

`spiceypy.spiceypy.ckobj (ck, out_cell=None)`

Find the set of ID codes of all objects in a specified CK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckobj_c.html

Parameters

- **ck** (`str`) – Name of CK file.
- **out_cell** (`Optional[SpiceCell]`) – Optional user provided Spice Int cell.

Return type `SpiceCell`

Returns Set of ID codes of objects in CK file.

`spiceypy.spiceypy.ckopn (filename, ifname, ncomch)`

Open a new CK file, returning the handle of the opened file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckopn_c.html

Parameters

- **filename** (`str`) – The name of the CK file to be opened.
- **ifname** (`str`) – The internal filename for the CK.
- **ncomch** (`int`) – The number of characters to reserve for comments.

Return type `int`

Returns The handle of the opened CK file.

`spiceypy.spiceypy.ckupf (handle)`

Unload a CK pointing file so that it will no longer be searched by the readers.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckupf_c.html

Parameters **handle** (`int`) – Handle of CK file to be unloaded

Return type None

`spiceypy.spiceypy.ckw01` (*handle, begtim, endtim, inst, ref, avflag, segid, nrec, sclkdp, quats, avvs*)
Add a type 1 segment to a C-kernel.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckw01_c.html

Parameters

- **handle** (int) – Handle of an open CK file.
- **begtim** (float) – The beginning encoded SCLK of the segment.
- **endtim** (float) – The ending encoded SCLK of the segment.
- **inst** (int) – The NAIF instrument ID code.
- **ref** (str) – The reference frame of the segment.
- **avflag** (bool) – True if the segment will contain angular velocity.
- **segid** (str) – Segment identifier.
- **nrec** (int) – Number of pointing records.
- **sclkdp** (Union[ndarray, Iterable[float]]) – Encoded SCLK times.
- **quats** (Union[ndarray, Iterable[Iterable[float]]]) – Quaternions representing instrument pointing.
- **avvs** (Union[ndarray, Iterable[Iterable[float]]]) – Angular velocity vectors.

Return type None

`spiceypy.spiceypy.ckw02` (*handle, begtim, endtim, inst, ref, segid, nrec, start, stop, quats, avvs, rates*)
Write a type 2 segment to a C-kernel.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckw02_c.html

Parameters

- **handle** (int) – Handle of an open CK file.
- **begtim** (float) – The beginning encoded SCLK of the segment.
- **endtim** (float) – The ending encoded SCLK of the segment.
- **inst** (int) – The NAIF instrument ID code.
- **ref** (str) – The reference frame of the segment.
- **segid** (str) – Segment identifier.
- **nrec** (int) – Number of pointing records.
- **start** (ndarray) – Encoded SCLK interval start times.
- **stop** (ndarray) – Encoded SCLK interval stop times.
- **quats** (ndarray) – Quaternions representing instrument pointing.
- **avvs** (ndarray) – Angular velocity vectors.
- **rates** (Iterable[float]) – Number of seconds per tick for each interval.

Return type None

`spiceypy.spiceypy.ckw03` (*handle, begtim, endtim, inst, ref, avflag, segid, nrec, sclkdp, quats, avvs, nints, starts*)

Add a type 3 segment to a C-kernel.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckw03_c.html

Parameters

- **handle** (`int`) – Handle of an open CK file.
- **begtim** (`float`) – The beginning encoded SCLK of the segment.
- **endtim** (`float`) – The ending encoded SCLK of the segment.
- **inst** (`int`) – The NAIF instrument ID code.
- **ref** (`str`) – The reference frame of the segment.
- **avflag** (`bool`) – True if the segment will contain angular velocity.
- **segid** (`str`) – Segment identifier.
- **nrec** (`int`) – Number of pointing records.
- **sclkdp** (`ndarray`) – Encoded SCLK times.
- **quats** (`ndarray`) – Quaternions representing instrument pointing.
- **avvs** (`ndarray`) – Angular velocity vectors.
- **nints** (`int`) – Number of intervals.
- **starts** (`Iterable[float]`) – Encoded SCLK interval start times.

Return type `None`

`spiceypy.spiceypy.ckw05` (*handle, subtype, degree, begtim, endtim, inst, ref, avflag, segid, sclkdp, packets, rate, nints, starts*)

Write a type 5 segment to a CK file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ckw05_c.html

Parameters

- **handle** (`int`) – Handle of an open CK file.
- **subtype** (`int`) – CK type 5 subtype code. Can be: 0, 1, 2, 3 see naif docs via link above.
- **degree** (`int`) – Degree of interpolating polynomials.
- **begtim** (`float`) – The beginning encoded SCLK of the segment.
- **endtim** (`float`) – The ending encoded SCLK of the segment.
- **inst** (`int`) – The NAIF instrument ID code.
- **ref** (`str`) – The reference frame of the segment.
- **avflag** (`bool`) – True if the segment will contain angular velocity.
- **segid** (`str`) – Segment identifier.
- **sclkdp** (`ndarray`) – Encoded SCLK times.
- **packts** (`Sequence[Iterable[float]]`) – Array of packets.
- **rate** (`float`) – Nominal SCLK rate in seconds per tick.
- **nints** (`int`) – Number of intervals.
- **starts** (`ndarray`) – Encoded SCLK interval start times.

Return type None

`spiceypy.spiceypy.cleared()`

`spiceypy.spiceypy.clight()`

Return the speed of light in a vacuum (IAU official value, in km/sec).

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/clight_c.html

Return type float

Returns The function returns the speed of light in vacuum (km/sec).

`spiceypy.spiceypy.clpool()`

Remove all variables from the kernel pool. Watches on kernel variables are retained.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/clpool_c.html

Return type None

`spiceypy.spiceypy.cltext(fname)`

Internal undocumented command for closing a text file opened by RDTEXT.

No URL available; relevant lines from SPICE source:

FORTRAN SPICE, rdtext.f:

```
C$Procedure  CLTEXT ( Close a text file opened by RDTEXT)
      ENTRY  CLTEXT ( FILE )
      CHARACTER*(*)      FILE
C      VARIABLE  I/O  DESCRIPTION
C      -----  ---  -----
C      FILE      I   Text file to be closed.
```

CSPICE, rdtext.c:

```
/* $Procedure  CLTEXT ( Close a text file opened by RDTEXT) */
/* Subroutine */ int cltext_(char *file, ftnlen file_len)
```

Parameters **fname** (str) – Text file to be closed.

Return type None

`spiceypy.spiceypy.cmprss(delim, n, instr, lenout=256)`

Compress a character string by removing occurrences of more than N consecutive occurrences of a specified character.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cmprss_c.html

Parameters

- **delim** (str) – Delimiter to be compressed.
- **n** (int) – Maximum consecutive occurrences of delim.
- **instr** (str) – Input string.
- **lenout** (int) – Optional available space in output string.

Return type str

Returns Compressed string.

`spiceypy.spiceypy.cnmfrm(cname, lenout=256)`

Retrieve frame ID code and name to associate with an object.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cnmfrm_c.html

Parameters

- **cname** (`str`) – Name of the object to find a frame for.
- **lenout** (`int`) – Maximum length available for frame name.

Return type `Tuple[int, str, bool]`

Returns The ID code of the frame associated with `cname`, The name of the frame with ID `frcode`.

`spiceypy.spiceypy.conics(elts, et)`

Determine the state (position, velocity) of an orbiting body from a set of elliptic, hyperbolic, or parabolic orbital elements.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/conics_c.html

Parameters

- **elts** (`ndarray`) – Conic elements.
- **et** (`float`) – Input time.

Return type `ndarray`

Returns State of orbiting body at `et`.

`spiceypy.spiceypy.convrt(x, inunit, outunit)`

Take a measurement `X`, the units associated with `X`, and units to which `X` should be converted; return `Y` the value of the measurement in the output units.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/convrt_c.html

Parameters

- **x** (`Union[float, Iterable[float]]`) – Number representing a measurement in some units.
- **inunit** (`str`) – The units in which `x` is measured.
- **outunit** (`str`) – Desired units for the measurement.

Return type `Union[ndarray, float]`

Returns The measurement in the desired units.

`spiceypy.spiceypy.copy(cell)`

Copy the contents of a `SpiceCell` of any data type to another cell of the same type.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/copy_c.html

Parameters **cell** (`SpiceCell`) – Cell to be copied.

Return type `SpiceCell`

Returns New cell

`spiceypy.spiceypy.cpos(string, chars, start)`

Find the first occurrence in a string of a character belonging to a collection of characters, starting at a specified location, searching forward.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cpos_c.html

Parameters

- **string** (*str*) – Any character string.
- **chars** (*str*) – A collection of characters.
- **start** (*int*) – Position to begin looking for one of chars.

Return type `int`

Returns The index of the first character of *str* at or following index *start* that is in the collection *chars*.

`spiceypy.spiceypy.cposr` (*string, chars, start*)

Find the first occurrence in a string of a character belonging to a collection of characters, starting at a specified location, searching in reverse.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cposr_c.html

Parameters

- **string** (*str*) – Any character string.
- **chars** (*str*) – A collection of characters.
- **start** (*int*) – Position to begin looking for one of chars.

Return type `int`

Returns The index of the last character of *str* at or before index *start* that is in the collection *chars*.

`spiceypy.spiceypy.cvpool` (*agent*)

Indicate whether or not any watched kernel variables that have a specified agent on their notification list have been updated.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cvpool_c.html

Parameters **agent** (*str*) – Name of the agent to check for notices.

Return type `bool`

Returns True if variables for “agent” have been updated.

`spiceypy.spiceypy.cyllat` (*r, lonc, z*)

Convert from cylindrical to latitudinal coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cyllat_c.html

Parameters

- **r** (*float*) – Distance of point from z axis.
- **lonc** (*float*) – Cylindrical angle of point from XZ plane(radians).
- **z** (*float*) – Height of point above XY plane.

Return type `Tuple[float, float, float]`

Returns Distance, Longitude (radians), and Latitude of point (radians).

`spiceypy.spiceypy.cylrec` (*r, lon, z*)

Convert from cylindrical to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cylrec_c.html

Parameters

- **r** (*float*) – Distance of a point from z axis.
- **lon** (*float*) – Angle (radians) of a point from xZ plane.

- **z** (float) – Height of a point above xY plane.

Return type ndarray

Returns Rectangular coordinates of the point.

`spiceypy.spiceypy.cylsph(r, lonc, z)`

Convert from cylindrical to spherical coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/cylsph_c.html

Parameters

- **r** (float) – Rectangular coordinates of the point.
- **lonc** (float) – Angle (radians) of point from XZ plane.
- **z** (float) – Height of point above XY plane.

Return type Tuple[float, float, float]

Returns Distance of point from origin, Polar angle (co-latitude in radians) of point, Azimuthal angle (longitude) of point (radians).

`spiceypy.spiceypy.dafac(handle, buffer)`

Add comments from a buffer of character strings to the comment area of a binary DAF file, appending them to any comments which are already present in the file's comment area.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafac_c.html

Parameters

- **handle** (int) – handle of a DAF opened with write access.
- **buffer** (Sequence[str]) – Buffer of comments to put into the comment area.

Return type None

`spiceypy.spiceypy.dafbbs(handle)`

Begin a backward search for arrays in a DAF.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafbbs_c.html

Parameters **handle** (int) – Handle of DAF to be searched.

Return type None

`spiceypy.spiceypy.dafbfs(handle)`

Begin a forward search for arrays in a DAF.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafbfs_c.html

Parameters **handle** (int) – Handle of file to be searched.

Return type None

`spiceypy.spiceypy.dafcls(handle)`

Close the DAF associated with a given handle.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafcls_c.html

Parameters **handle** (int) – Handle of DAF to be closed.

Return type None

`spiceypy.spiceypy.dafcs(handle)`

Select a DAF that already has a search in progress as the one to continue searching.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafcs_c.html

Parameters `handle` (`int`) – Handle of DAF to continue searching.

Return type `None`

`spiceypy.spiceypy.dafdc` (`handle`)

Delete the entire comment area of a specified DAF file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafdc_c.html

Parameters `handle` (`int`) – The handle of a binary DAF opened for writing.

Return type `None`

`spiceypy.spiceypy.dafec` (`handle`, `bufsiz`, `lenout=256`)

Extract comments from the comment area of a binary DAF.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafec_c.html

Parameters

- **handle** (`int`) – Handle of binary DAF opened with read access.
- **bufsiz** (`int`) – Maximum size, in lines, of buffer.
- **lenout** (`int`) – Length of strings in output buffer.

Return type `Tuple[int, Iterable[str], bool]`

Returns Number of extracted comment lines, buffer where extracted comment lines are placed, Indicates whether all comments have been extracted.

`spiceypy.spiceypy.daffna` ()

Find the next (forward) array in the current DAF.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/daffna_c.html

Return type `bool`

Returns True if an array was found.

`spiceypy.spiceypy.daffpa` ()

Find the previous (backward) array in the current DAF.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/daffpa_c.html

Return type `bool`

Returns True if an array was found.

`spiceypy.spiceypy.dafgda` (`handle`, `begin`, `end`)

Read the double precision data bounded by two addresses within a DAF.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafgda_c.html

Parameters

- **handle** (`int`) – Handle of a DAF.
- **begin** (`int`) – Initial address within file.
- **end** (`int`) – Final address within file.

Return type `ndarray`

Returns Data contained between begin and end.

`spiceypy.spiceypy.dafgh()`

Return (get) the handle of the DAF currently being searched.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafgh_c.html

Return type `int`

Returns Handle for current DAF.

`spiceypy.spiceypy.dafgn(lenout=256)`

Return (get) the name for the current array in the current DAF.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafgn_c.html

Parameters `lenout` (`int`) – Length of array name string.

Return type `str`

Returns Name of current array.

`spiceypy.spiceypy.dafgs(n=125)`

Return (get) the summary for the current array in the current DAF.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafgs_c.html

Parameters `n` (`int`) – Optional length N for result Array.

Return type `ndarray`

Returns Summary for current array.

`spiceypy.spiceypy.dafgsr(handle, recno, begin, end)`

Read a portion of the contents of (words in) a summary record in a DAF file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafgsr_c.html

Parameters

- **handle** (`int`) – Handle of DAF.
- **recno** (`int`) – Record number; word indices are 1-based, 1 to 128 inclusive.
- **begin** (`int`) – Index of first word to read from record, will be clamped > 0 .
- **end** (`int`) – Index of last word to read, will be clamped < 129

Return type `Tuple[ndarray, bool]`

Returns Contents of request sub-record

`spiceypy.spiceypy.dafopr(fname)`

Open a DAF for subsequent read requests.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafopr_c.html

Parameters `fname` (`str`) – Name of DAF to be opened.

Return type `int`

Returns Handle assigned to DAF.

`spiceypy.spiceypy.dafopw(fname)`

Open a DAF for subsequent write requests.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafopw_c.html

Parameters `fname` (`str`) – Name of DAF to be opened.

Return type `int`

Returns Handle assigned to DAF.

`spiceypy.spiceypy.dafps(nd, ni, dc, ic)`

Pack (assemble) an array summary from its double precision and integer components.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafps_c.html

Parameters

- **nd** (int) – Number of double precision components.
- **ni** (int) – Number of integer components.
- **dc** (ndarray) – Double precision components.
- **ic** (ndarray) – Integer components.

Return type ndarray

Returns Array summary.

`spiceypy.spiceypy.dafrda(handle, begin, end)`

Read the double precision data bounded by two addresses within a DAF.

Deprecated: This routine has been superseded by `dafgda()` and `dafgsr()`. This routine is supported for purposes of backward compatibility only.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafrda_c.html

Parameters

- **handle** (int) – Handle of a DAF.
- **begin** (int) – Initial address within file.
- **end** (int) – Final address within file.

Return type ndarray

Returns Data contained between begin and end.

`spiceypy.spiceypy.dafrrf(handle, lenout=256)`

Read the contents of the file record of a DAF.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafrrf_c.html

Parameters

- **handle** (int) – Handle of an open DAF file.
- **lenout** (int) – Available room in the output string

Return type Tuple[int, int, str, int, int, int]

Returns Number of double precision components in summaries, Number of integer components in summaries, Internal file name, Forward list pointer, Backward list pointer, Free address pointer.

`spiceypy.spiceypy.dafrs(insum)`

Change the summary for the current array in the current DAF.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafrs_c.html

Parameters **insum** (ndarray) – New summary for current array.

Return type None

`spiceypy.spiceypy.dafus (insum, nd, ni)`

Unpack an array summary into its double precision and integer components.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dafus_c.html

Parameters

- **insum** (ndarray) – Array summary.
- **nd** (int) – Number of double precision components.
- **ni** (int) – Number of integer components.

Return type Tuple[ndarray, ndarray]

Returns Double precision components, Integer components.

`spiceypy.spiceypy.dasac (handle, buffer)`

Add comments from a buffer of character strings to the comment area of a binary DAS file, appending them to any comments which are already present in the file’s comment area.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dasac_c.html

Parameters

- **handle** (int) – DAS handle of a file opened with write access.
- **buffer** (Sequence[str]) – Buffer of lines to be put into the comment area.

Return type None

`spiceypy.spiceypy.dascls (handle)`

Close a DAS file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dascls_c.html

Parameters **handle** (int) – Handle of an open DAS file.

Return type None

`spiceypy.spiceypy.dasdc (handle)`

Delete the entire comment area of a previously opened binary DAS file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dasdc_c.html

Parameters **handle** (int) – The handle of a binary DAS file opened for writing.

Return type None

`spiceypy.spiceypy.dasec (handle, bufsiz=256, buflen=256)`

Extract comments from the comment area of a binary DAS file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dasec_c.html

Parameters

- **handle** (int) – Handle of binary DAS file open with read access.
- **bufsiz** (int) – Maximum size, in lines, of buffer.
- **buflen** (int) – Line length associated with buffer.

Return type Tuple[int, Iterable[str], int]

Returns Number of comments extracted from the DAS file, Buffer in which extracted comments are placed, Indicates whether all comments have been extracted.

`spiceypy.spiceypy.dashfn(handle, lenout=256)`

Return the name of the DAS file associated with a handle.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dashfn_c.html

Parameters

- **handle** (`int`) – Handle of a DAS file.
- **lenout** (`int`) – Length of output file name string.

Return type `str`

Returns Corresponding file name.

`spiceypy.spiceypy.dasonw(fname, ftype, ifname, ncomch)`

Internal undocumented command for creating a new DAS file

Parameters

- **fname** (`str`) – filename
- **ftype** (`str`) – type
- **ifname** (`str`) – internal file name
- **ncomch** (`int`) – amount of comment area

Return type `int`

Returns Handle to new DAS file

`spiceypy.spiceypy.dasopr(fname)`

Open a DAS file for reading.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dasopr_c.html

Parameters **fname** (`str`) – Name of a DAS file to be opened.

Return type `int`

Returns Handle assigned to the opened DAS file.

`spiceypy.spiceypy.dasopw(fname)`

Open a DAS file for writing.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dasopw_c.html :type fname: `str` :param fname: Name of a DAS file to be opened. :rtype: `int` :return: Handle assigned to the opened DAS file.

`spiceypy.spiceypy.dasrfr(handle, lenout=256)`

Return the contents of the file record of a specified DAS file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dasrfr_c.html

Parameters

- **handle** (`int`) – DAS file handle.
- **lenout** (`int`) – length of output str

Return type `Tuple[str, str, int, int, int, int]`

Returns ID word, DAS internal file name, Number of reserved records in file, Number of characters in use in reserved rec. area, Number of comment records in file, Number of characters in use in comment area.

`spiceypy.spiceypy.datetime2et(dt)`

Converts a standard Python datetime to a double precision value representing the number of TDB seconds past the J2000 epoch corresponding to the input epoch.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/time.html#The%20J2000%20Epoch

Parameters `dt` (Union[Iterable[datetime], datetime]) – A standard Python datetime

Return type Union[ndarray, float]

Returns The equivalent value in seconds past J2000, TDB.

`spiceypy.spiceypy.dcyldr(x, y, z)`

This routine computes the Jacobian of the transformation from rectangular to cylindrical coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dcyldr_c.html

Parameters

- `x` (float) – X-coordinate of point.
- `y` (float) – Y-coordinate of point.
- `z` (float) – Z-coordinate of point.

Return type ndarray

Returns Matrix of partial derivatives.

`spiceypy.spiceypy.deltet(epoch, eptype)`

Return the value of Delta ET (ET-UTC) for an input epoch.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/deltet_c.html

Parameters

- `epoch` (float) – Input epoch (seconds past J2000).
- `eptype` (str) – Type of input epoch (“UTC” or “ET”).

Return type float

Returns Delta ET (ET-UTC) at input epoch.

`spiceypy.spiceypy.det(m1)`

Compute the determinant of a double precision 3x3 matrix.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/det_c.html

Parameters `m1` (ndarray) – Matrix whose determinant is to be found.

Return type float

Returns The determinant of the matrix.

`spiceypy.spiceypy.dgeodr(x, y, z, re, f)`

This routine computes the Jacobian of the transformation from rectangular to geodetic coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dgeodr_c.html

Parameters

- `x` (float) – X-coordinate of point.
- `y` (float) – Y-coordinate of point.
- `z` (float) – Z-coord
- `re` (float) – Equatorial radius of the reference spheroid.

- **f** (float) – Flattening coefficient.

Return type ndarray

Returns Matrix of partial derivatives.

`spiceypy.spiceypy.diams2(symmat)`

Diagonalize a symmetric 2x2 matrix.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/diams2_c.html

Parameters **symmat** (Iterable[Iterable[float]]) – A symmetric 2x2 matrix.

Return type Tuple[ndarray, ndarray]

Returns A diagonal matrix similar to symmat, A rotation used as the similarity transformation.

`spiceypy.spiceypy.diff(a, b)`

Take the difference of two sets of any data type to form a third set. http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/diff_c.html

Parameters

- **a** (*SpiceCell*) – First input set.
- **b** (*SpiceCell*) – Second input set.

Return type *SpiceCell*

Returns Difference of a and b.

`spiceypy.spiceypy.dlabbs(handle)`

Begin a backward segment search in a DLA file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dlabbs_c.html

Parameters **handle** (int) – Handle of open DLA file.

Return type Tuple[*SpiceDLADescr*, bool]

Returns Descriptor of last segment in DLA file

`spiceypy.spiceypy.dlabfs(handle)`

Begin a forward segment search in a DLA file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dlabfs_c.html

Parameters **handle** (int) – Handle of open DLA file.

Return type Tuple[*SpiceDLADescr*, bool]

Returns Descriptor of next segment in DLA file

`spiceypy.spiceypy.dlafns(handle, descr)`

Find the segment following a specified segment in a DLA file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dlafns_c.html

Parameters

- **handle** (int) – Handle of open DLA file.
- **descr** (*SpiceDLADescr*) – Descriptor of a DLA segment.

Return type Tuple[*SpiceDLADescr*, bool]

Returns Descriptor of next segment in DLA file

`spiceypy.spiceypy.dlafps` (*handle*, *descr*)

Find the segment preceding a specified segment in a DLA file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dlafps_c.html

Parameters

- **handle** (`int`) – Handle of open DLA file.
- **descr** (*SpiceDLADescr*) – Descriptor of a segment in DLA file.

Return type `Tuple[SpiceDLADescr, bool]`

Returns Descriptor of previous segment in DLA file

`spiceypy.spiceypy.dlatdr` (*x*, *y*, *z*)

This routine computes the Jacobian of the transformation from rectangular to latitudinal coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dlatdr_c.html

Parameters

- **x** (`float`) – X-coordinate of point.
- **y** (`float`) – Y-coordinate of point.
- **z** (`float`) – Z-coord

Return type `ndarray`

Returns Matrix of partial derivatives.

`spiceypy.spiceypy.dp2hx` (*number*, *lenout*=256)

Convert a double precision number to an equivalent character string using base 16 “scientific notation.”

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dp2hx_c.html

Parameters

- **number** (`float`) – D.p. number to be converted.
- **lenout** (`int`) – Available space for output string.

Return type `str`

Returns Equivalent character string, left justified.

`spiceypy.spiceypy.dpgrdr` (*body*, *x*, *y*, *z*, *re*, *f*)

This routine computes the Jacobian matrix of the transformation from rectangular to planetographic coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dpgrdr_c.html

Parameters

- **body** (`str`) – Body with which coordinate system is associated.
- **x** (`float`) – X-coordinate of point.
- **y** (`float`) – Y-coordinate of point.
- **z** (`int`) – Z-coordinate of point.
- **re** (`float`) – Equatorial radius of the reference spheroid.
- **f** (`float`) – Flattening coefficient.

Return type `ndarray`

Returns Matrix of partial derivatives.

`spiceypy.spiceypy.dpmax()`

Return the value of the largest (positive) number representable in a double precision variable.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dpmax_c.html

Return type float

Returns The largest (positive) number representable in a double precision variable.

`spiceypy.spiceypy.dpmin()`

Return the value of the smallest (negative) number representable in a double precision variable.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dpmin_c.html

Return type float

Returns The smallest (negative) number that can be represented in a double precision variable.

`spiceypy.spiceypy.dpr()`

Return the number of degrees per radian.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dpr_c.html

Return type float

Returns The number of degrees per radian.

`spiceypy.spiceypy.drdcyl(r, lon, z)`

This routine computes the Jacobian of the transformation from cylindrical to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/drdcyl_c.html

Parameters

- **r** (float) – Distance of a point from the origin.
- **lon** (float) – Angle of the point from the xz plane in radians.
- **z** (float) – Height of the point above the xy plane.

Return type ndarray

Returns Matrix of partial derivatives.

`spiceypy.spiceypy.drldgeo(lon, lat, alt, re, f)`

This routine computes the Jacobian of the transformation from geodetic to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/drldgeo_c.html

Parameters

- **lon** (float) – Geodetic longitude of point (radians).
- **lat** (float) – Geodetic latitude of point (radians).
- **alt** (float) – Altitude of point above the reference spheroid.
- **re** (float) – Equatorial radius of the reference spheroid.
- **f** (float) – Flattening coefficient.

Return type ndarray

Returns Matrix of partial derivatives.

`spiceypy.spiceypy.drldlat(r, lon, lat)`

Compute the Jacobian of the transformation from latitudinal to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/drldlat_c.html

Parameters

- **r** (float) – Distance of a point from the origin.
- **lon** (float) – Angle of the point from the XZ plane in radians.
- **lat** (float) – Angle of the point from the XY plane in radians.

Return type ndarray

Returns Matrix of partial derivatives.

`spiceypy.spiceypy.drdpgr(body, lon, lat, alt, re, f)`

This routine computes the Jacobian matrix of the transformation from planetographic to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/drdpgr_c.html

Parameters

- **body** (str) – Body with which coordinate system is associated.
- **lon** (float) – Planetographic longitude of a point (radians).
- **lat** (float) – Planetographic latitude of a point (radians).
- **alt** (int) – Altitude of a point above reference spheroid.
- **re** (float) – Equatorial radius of the reference spheroid.
- **f** (float) – Flattening coefficient.

Return type ndarray

Returns Matrix of partial derivatives.

`spiceypy.spiceypy.drdsph(r, colat, lon)`

This routine computes the Jacobian of the transformation from spherical to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/drdsph_c.html

Parameters

- **r** (float) – Distance of a point from the origin.
- **colat** (float) – Angle of the point from the positive z-axis.
- **lon** (float) – Angle of the point from the xy plane.

Return type ndarray

Returns Matrix of partial derivatives.

`spiceypy.spiceypy.dskb02(handle, dlasc)`

Return bookkeeping data from a DSK type 2 segment.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskb02_c.html

Parameters

- **handle** (int) – DSK file handle
- **dlasc** (*SpiceDLADescr*) – DLA descriptor

Return type Tuple[int, int, int, ndarray, float, ndarray, ndarray, int, int, int, int]

Returns bookkeeping data from a DSK type 2 segment

`spiceypy.spiceypy.dskcls` (*handle*, *optimiz=False*)
Close a DSK file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskcls_c.html

Parameters

- **handle** (*int*) – Handle assigned to the opened DSK file.
- **optimiz** (*bool*) – Flag indicating whether to segregate the DSK.

Return type `None`

Returns

`spiceypy.spiceypy.dskd02` (*handle*, *dladsc*, *item*, *start*, *room*)
Fetch double precision data from a type 2 DSK segment.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskd02_c.html

Parameters

- **handle** (*int*) – DSK file handle
- **dladsc** (*SpiceDLADescr*) – DLA descriptor
- **item** (*int*) – Keyword identifying item to fetch
- **start** (*int*) – Start index
- **room** (*int*) – Amount of room in output array

Return type `ndarray`

Returns Array containing requested item

`spiceypy.spiceypy.dskgd` (*handle*, *dladsc*)
Return the DSK descriptor from a DSK segment identified by a DAS handle and DLA descriptor.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskgd_c.html

Parameters

- **handle** (*int*) – Handle assigned to the opened DSK file.
- **dladsc** (*SpiceDLADescr*) – DLA segment descriptor.

Return type *SpiceDSKDescr*

Returns DSK segment descriptor.

`spiceypy.spiceypy.dskgt1` (*keywr*)
Retrieve the value of a specified DSK tolerance or margin parameter.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskgt1_c.html

Parameters **keywr** (*int*) – Code specifying parameter to retrieve.

Return type `float`

Returns Value of parameter.

`spiceypy.spiceypy.dski02` (*handle*, *dladsc*, *item*, *start*, *room*)
Fetch integer data from a type 2 DSK segment.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dski02_c.html

Parameters

- **handle** (*int*) – DSK file handle.

- **dladsc** (*SpiceDLADescr*) – DLA descriptor.
- **item** (*int*) – Keyword identifying item to fetch.
- **start** (*int*) – Start index.
- **room** (*int*) – Amount of room in output array.

Return type `ndarray`

Returns Array containing requested item.

`spiceypy.spiceypy.dskmi2` (*vrtces, plates, finscl, corscl, worksz, voxpsz, voxlsz, makvtl, spxisz*)

Make spatial index for a DSK type 2 segment. The index is returned as a pair of arrays, one of type `int` and one of type `float`. These arrays are suitable for use with the DSK type 2 writer `dskw02`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskmi2_c.html

Parameters

- **vrtces** (`ndarray`) – Vertices
- **plates** (`ndarray`) – Plates
- **finscl** (`float`) – Fine voxel scale
- **corscl** (`int`) – Coarse voxel scale
- **worksiz** (`int`) – Workspace size
- **voxpsz** (`int`) – Voxel plate pointer array size
- **voxlsz** (`int`) – Voxel plate list array size
- **makvtl** (`bool`) – Vertex plate list flag
- **spxisz** (`int`) – Spatial index integer component size

Return type `Tuple[ndarray, ndarray]`

Returns double precision and integer components of the spatial index of the segment.

`spiceypy.spiceypy.dskn02` (*handle, dladsc, plid*)

Compute the unit normal vector for a specified plate from a type 2 DSK segment.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskn02_c.html

Parameters

- **handle** (`int`) – DSK file handle.
- **dladsc** (*SpiceDLADescr*) – DLA descriptor.
- **plid** (`int`) – Plate ID.

Return type `ndarray`

Returns plate's unit normal vector.

`spiceypy.spiceypy.dskobj` (*dsk*)

Find the set of body ID codes of all objects for which topographic data are provided in a specified DSK file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskobj_c.html

Parameters **dsk** (`str`) – Name of DSK file.

Return type *SpiceCell*

Returns Set of ID codes of objects in DSK file.

`spiceypy.spiceypy.dskopn (fname, ifname, ncomch)`

Open a new DSK file for subsequent write operations.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskopn_c.html

Parameters

- **fname** (`str`) – Name of a DSK file to be opened.
- **ifname** (`str`) – Internal file name.
- **ncomch** (`int`) – Number of comment characters to allocate.

Return type `int`

Returns Handle assigned to the opened DSK file.

`spiceypy.spiceypy.dskp02 (handle, dladsc, start, room)`

Fetch triangular plates from a type 2 DSK segment.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskp02_c.html

Parameters

- **handle** (`int`) – DSK file handle.
- **dladsc** (*SpiceDLADescr*) – DLA descriptor.
- **start** (`int`) – Start index.
- **room** (`int`) – Amount of room in output array.

Return type `ndarray`

Returns Array containing plates.

`spiceypy.spiceypy.dskrb2 (vrtces, plates, corsys, corpar)`

Determine range bounds for a set of triangular plates to be stored in a type 2 DSK segment.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskrb2_c.html

Parameters

- **vrtces** (`ndarray`) – Vertices
- **plates** (`ndarray`) – Plates
- **corsys** (`int`) – DSK coordinate system code
- **corpar** (`ndarray`) – DSK coordinate system parameters

Return type `Tuple[float, float]`

Returns Lower and Upper bound on range of third coordinate

`spiceypy.spiceypy.dsksrfr (dsk, bodyid)`

Find the set of surface ID codes for all surfaces associated with a given body in a specified DSK file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dsksrfr_c.html

Parameters

- **dsk** (`str`) – Name of DSK file.
- **bodyid** (`int`) – Integer body ID code.

Return type *SpiceCell*

Returns Set of ID codes of surfaces in DSK file.

`spiceypy.spiceypy.dskstl(keywrld, dpval)`

Set the value of a specified DSK tolerance or margin parameter.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskstl_c.html

Parameters

- **keywrld** (int) – Code specifying parameter to set.
- **dpval** (float) – Value of parameter.

Return type None

Returns

`spiceypy.spiceypy.dskv02(handle, dladsc, start, room)`

Fetch vertices from a type 2 DSK segment.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskv02_c.html

Parameters

- **handle** (int) – DSK file handle.
- **dladsc** (*SpiceDLADescr*) – DLA descriptor.
- **start** (int) – Start index.
- **room** (int) – Amount of room in output array.

Return type ndarray

Returns Array containing vertices.

`spiceypy.spiceypy.dskw02(handle, center, surfid, dclass, fname, corsys, corpar, mncor1, mxcor1, mncor2, mxcor2, mncor3, mxcor3, first, last, vrtces, plates, spaixd, spaixi)`

Write a type 2 segment to a DSK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskw02_c.html

Parameters

- **handle** (int) – Handle assigned to the opened DSK file
- **center** (int) – Central body ID code
- **surfid** (int) – Surface ID code
- **dclass** (int) – Data class
- **fname** (str) – Reference frame
- **corsys** (int) – Coordinate system code
- **corpar** (ndarray) – Coordinate system parameters
- **mncor1** (float) – Minimum value of first coordinate
- **mxcor1** (float) – Maximum value of first coordinate
- **mncor2** (float) – Minimum value of second coordinate
- **mxcor2** (float) – Maximum value of second coordinate
- **mncor3** (float) – Minimum value of third coordinate
- **mxcor3** (float) – Maximum value of third coordinate
- **first** (float) – Coverage start time
- **last** (float) – Coverage stop time

- **vtces** (ndarray) – Vertices
- **plates** (ndarray) – Plates
- **spaixd** (ndarray) – Double precision component of spatial index
- **spaixi** (ndarray) – Integer component of spatial index

Return type None

`spiceypy.spiceypy.dskx02(handle, dladsc, vertex, raydir)`

Determine the plate ID and body-fixed coordinates of the intersection of a specified ray with the surface defined by a type 2 DSK plate model.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskx02_c.html

Parameters

- **handle** (int) – Handle of DSK kernel containing plate model.
- **dladsc** (*SpiceDLADescr*) – DLA descriptor of plate model segment.
- **vertex** (ndarray) – Ray’s vertex in the body fixed frame.
- **raydir** (ndarray) – Ray direction in the body fixed frame.

Return type Tuple[int, ndarray, bool]

Returns ID code of the plate intersected by the ray, Intercept, and Flag indicating whether intercept exists.

`spiceypy.spiceypy.dskxsi(pri, target, srflst, et, fixref, vertex, raydir)`

Compute a ray-surface intercept using data provided by multiple loaded DSK segments. Return information about the source of the data defining the surface on which the intercept was found: DSK handle, DLA and DSK descriptors, and DSK data type-dependent parameters.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskxsi_c.html

Parameters

- **pri** (bool) – Data prioritization flag.
- **target** (str) – Target body name.
- **srflst** (Sequence[int]) – Surface ID list.
- **et** (float) – Epoch, expressed as seconds past J2000 TDB.
- **fixref** (str) – Name of target body-fixed reference frame.
- **vertex** (ndarray) – Vertex of ray.
- **raydir** (ndarray) – Direction vector of ray.

Return type Tuple[ndarray, int, *SpiceDLADescr*, *SpiceDSKDescr*, ndarray, ndarray, bool]

Returns Intercept point, Handle of segment contributing surface data, DLADSC, DSKDSC, Double precision component of source info, Integer component of source info

`spiceypy.spiceypy.dskxv(pri, target, srflst, et, fixref, vtxarr, dirarr)`

Compute ray-surface intercepts for a set of rays, using data provided by multiple loaded DSK segments.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskxv_c.html

Parameters

- **pri** (bool) – Data prioritization flag.

- **target** (str) – Target body name.
- **srflst** (Sequence[int]) – Surface ID list.
- **et** (float) – Epoch, expressed as seconds past J2000 TDB.
- **fixref** (str) – Name of target body-fixed reference frame.
- **vtxarr** (Sequence[ndarray]) – Array of vertices of rays.
- **dirarr** (Sequence[ndarray]) – Array of direction vectors of rays.

Return type Tuple[ndarray, ndarray]

Returns Intercept point array and Found flag array.

`spiceypy.spiceypy.dskz02(handle, dladsc)`

Return plate model size parameters—plate count and vertex count—for a type 2 DSK segment.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dskz02_c.html

Parameters

- **handle** (int) – DSK file handle.
- **dladsc** (*SpiceDLADescr*) – DLA descriptor.

Return type Tuple[int, int]

Returns Number of vertices, Number of plates.

`spiceypy.spiceypy.dsphdr(x, y, z)`

This routine computes the Jacobian of the transformation from rectangular to spherical coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dsphdr_c.html

Parameters

- **x** (float) – X-coordinate of point.
- **y** (float) – Y-coordinate of point.
- **z** (float) – Z-coordinate of point.

Return type ndarray

Returns Matrix of partial derivatives.

`spiceypy.spiceypy.dtpool(name)`

Return the data about a kernel pool variable.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dtpool_c.html

Parameters **name** (str) – Name of the variable whose value is to be returned.

Return type Tuple[int, str, bool]

Returns Number of values returned for name, Type of the variable “C”, “N”, or “X”.

`spiceypy.spiceypy.ducrss(s1, s2)`

Compute the unit vector parallel to the cross product of two 3-dimensional vectors and the derivative of this unit vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ducrss_c.html

Parameters

- **s1** (ndarray) – Left hand state for cross product and derivative.
- **s2** (ndarray) – Right hand state for cross product and derivative.

Return type ndarray

Returns Unit vector and derivative of the cross product.

`spiceypy.spiceypy.dvcrss(s1, s2)`

Compute the cross product of two 3-dimensional vectors and the derivative of this cross product.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dvcrss_c.html

Parameters

- **s1** (ndarray) – Left hand state for cross product and derivative.
- **s2** (ndarray) – Right hand state for cross product and derivative.

Return type ndarray

Returns State associated with cross product of positions.

`spiceypy.spiceypy.dvdot(s1, s2)`

Compute the derivative of the dot product of two double precision position vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dvdot_c.html

Parameters

- **s1** (Sequence[float]) – First state vector in the dot product.
- **s2** (Sequence[float]) – Second state vector in the dot product.

Return type float

Returns The derivative of the dot product.

`spiceypy.spiceypy.dvhat(s1)`

Find the unit vector corresponding to a state vector and the derivative of the unit vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dvhat_c.html

Parameters **s1** (ndarray) – State to be normalized.

Return type ndarray

Returns Unit vector $s1 / \text{abs}(s1)$, and its time derivative.

`spiceypy.spiceypy.dvnorm(state)`

Function to calculate the derivative of the norm of a 3-vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dvnorm_c.html

Parameters **state** (ndarray) – A 6-vector composed of three coordinates and their derivatives.

Return type float

Returns The derivative of the norm of a 3-vector.

`spiceypy.spiceypy.dvpool(name)`

Delete a variable from the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dvpool_c.html

Parameters **name** (str) – Name of the kernel variable to be deleted.

Return type None

`spiceypy.spiceypy.dvsep(s1, s2)`

Calculate the time derivative of the separation angle between two input states, S1 and S2.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/dvsep_c.html

Parameters

- **s1** (ndarray) – State vector of the first body.
- **s2** (ndarray) – State vector of the second body.

Return type float

Returns The time derivative of the angular separation between S1 and S2.

`spiceypy.spiceypy.edlimb(a, b, c, viewpt)`

Find the limb of a triaxial ellipsoid, viewed from a specified point.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/edlimb_c.html

Parameters

- **a** (float) – Length of ellipsoid semi-axis lying on the x-axis.
- **b** (float) – Length of ellipsoid semi-axis lying on the y-axis.
- **c** (float) – Length of ellipsoid semi-axis lying on the z-axis.
- **viewpt** (Iterable[float]) – Location of viewing point.

Return type *Ellipse*

Returns Limb of ellipsoid as seen from viewing point.

`spiceypy.spiceypy.edterm(trmtyp, source, target, et, fixref, abcorr, obsrvr, npts)`

Compute a set of points on the umbral or penumbral terminator of a specified target body, where the target shape is modeled as an ellipsoid.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/edterm_c.html

Parameters

- **trmtyp** (str) – Terminator type.
- **source** (str) – Light source.
- **target** (str) – Target body.
- **et** (float) – Observation epoch.
- **fixref** (str) – Body-fixed frame associated with target.
- **abcorr** (str) – Aberration correction.
- **obsrvr** (str) – Observer.
- **npts** (int) – Number of points in terminator set.

Return type Tuple[float, ndarray, ndarray]

Returns Epoch associated with target center, Position of observer in body-fixed frame, Terminator point set.

`spiceypy.spiceypy.ekacec(handle, segno, recno, column, nvals, cvals, isnull)`

Add data to a character column in a specified EK record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekacec_c.html

Parameters

- **handle** (int) – EK file handle.
- **segno** (int) – Index of segment containing record.
- **recno** (int) – Record to which data is to be added.

- **column** (*str*) – Column name.
- **nvals** (*int*) – Number of values to add to column.
- **cvals** (*Iterable[str]*) – Character values to add to column.
- **isnull** (*bool*) – Flag indicating whether column entry is null.

Return type *None*

`spiceypy.spiceypy.ekaced` (*handle, segno, recno, column, nvals, dvals, isnull*)
Add data to an double precision column in a specified EK record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekaced_c.html

Parameters

- **handle** (*int*) – EK file handle.
- **segno** (*int*) – Index of segment containing record.
- **recno** (*int*) – Record to which data is to be added.
- **column** (*str*) – Column name.
- **nvals** (*int*) – Number of values to add to column.
- **dvals** (*Iterable[float]*) – Double precision values to add to column.
- **isnull** (*bool*) – Flag indicating whether column entry is null.

Return type *None*

`spiceypy.spiceypy.ekacei` (*handle, segno, recno, column, nvals, ival, isnull*)
Add data to an integer column in a specified EK record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekacei_c.html

Parameters

- **handle** (*int*) – EK file handle.
- **segno** (*int*) – Index of segment containing record.
- **recno** (*int*) – Record to which data is to be added.
- **column** (*str*) – Column name.
- **nvals** (*int*) – Number of values to add to column.
- **ival** (*Iterable[int]*) – Integer values to add to column.
- **isnull** (*bool*) – Flag indicating whether column entry is null.

Return type *None*

`spiceypy.spiceypy.ekac1c` (*handle, segno, column, vallen, cvals, entszs, n1flgs, rcptrs, wkindx*)
Add an entire character column to an EK segment.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekac1c_c.html

Parameters

- **handle** (*int*) – EK file handle.
- **segno** (*int*) – Number of segment to add column to.
- **column** (*str*) – Column name.
- **vallen** (*int*) – Length of character values.

- **cvals** (Iterable[str]) – Character values to add to column.
- **entszs** (Iterable[int]) – Array of sizes of column entries.
- **nlflgs** (Iterable[bool]) – Array of null flags for column entries.
- **rcptrs** (ndarray) – Record pointers for segment.
- **wkindx** (Iterable[int]) – Work space for column index.

Return type ndarray

Returns Work space for column index.

`spiceypy.spiceypy.ekacld(handle, segno, column, dvals, entszs, nlflgs, rcptrs, wkindx)`
 Add an entire double precision column to an EK segment.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekacld_c.html

Parameters

- **handle** (int) – EK file handle.
- **segno** (int) – Number of segment to add column to.
- **column** (str) – Column name.
- **dvals** (Iterable[float]) – Double precision values to add to column.
- **entszs** (Iterable[int]) – Array of sizes of column entries.
- **nlflgs** (Iterable[bool]) – Array of null flags for column entries.
- **rcptrs** (ndarray) – Record pointers for segment.
- **wkindx** (Iterable[int]) – Work space for column index.

Return type ndarray

Returns Work space for column index.

`spiceypy.spiceypy.ekacli(handle, segno, column, ival, entszs, nlflgs, rcptrs, wkindx)`
 Add an entire integer column to an EK segment.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekacli_c.html

Parameters

- **handle** (int) – EK file handle.
- **segno** (int) – Number of segment to add column to.
- **column** (str) – Column name.
- **ival** (Iterable[int]) – Integer values to add to column.
- **entszs** (Iterable[int]) – Array of sizes of column entries.
- **nlflgs** (Iterable[bool]) – Array of null flags for column entries.
- **rcptrs** (ndarray) – Record pointers for segment.
- **wkindx** (Iterable[int]) – Work space for column index.

Return type ndarray

Returns Work space for column index.

`spiceypy.spiceypy.ekappr` (*handle*, *segno*)

Append a new, empty record at the end of a specified E-kernel segment.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekappr_c.html

Parameters

- **handle** (*int*) – File handle.
- **segno** (*int*) – Segment number.

Return type *int*

Returns Number of appended record.

`spiceypy.spiceypy.ekbseg` (*handle*, *tabnam*, *cnames*, *decls*)

Start a new segment in an E-kernel.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekbseg_c.html

Parameters

- **handle** (*int*) – File handle.
- **tabnam** (*str*) – Table name.
- **cnames** (*Sequence[str]*) – Names of columns.
- **decls** (*Sequence[str]*) – Declarations of columns.

Return type *int*

Returns Segment number.

`spiceypy.spiceypy.ekccnt` (*table*)

Return the number of distinct columns in a specified, currently loaded table.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekccnt_c.html

Parameters **table** (*str*) – Name of table.

Return type *int*

Returns Count of distinct, currently loaded columns.

`spiceypy.spiceypy.ekcii` (*table*, *cindex*, *lenout=256*)

Return attribute information about a column belonging to a loaded EK table, specifying the column by table and index.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekcii_c.html

Parameters

- **table** (*str*) – Name of table containing column.
- **cindex** (*int*) – Index of column whose attributes are to be found.
- **lenout** (*int*) – Maximum allowed length of column name.

Return type *Tuple[str, SpiceEKAttDsc]*

Returns Name of column, Column attribute descriptor.

`spiceypy.spiceypy.ekcls` (*handle*)

Close an E-kernel.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekcls_c.html

Parameters **handle** (*int*) – EK file handle.

Return type None

`spiceypy.spiceypy.ekdelr(handle, segno, recno)`

Delete a specified record from a specified E-kernel segment.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekdelr_c.html

Parameters

- **handle** (int) – File handle.
- **segno** (int) – Segment number.
- **recno** (int) – Record number.

Return type None

`spiceypy.spiceypy.ekffld(handle, segno, rcptrs)`

Complete a fast write operation on a new E-kernel segment.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekffld_c.html

Parameters

- **handle** (int) – File handle.
- **segno** (int) – Segment number.
- **rcptrs** (ndarray) – Record pointers.

Return type None

`spiceypy.spiceypy.ekfind(query, lenout=256)`

Find E-kernel data that satisfy a set of constraints.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekfind_c.html

Parameters

- **query** (str) – Query specifying data to be found.
- **lenout** (int) – Declared length of output error message string.

Return type Tuple[int, int, str]

Returns Number of matching rows, Flag indicating whether query parsed correctly, Parse error description.

`spiceypy.spiceypy.ekgc(selidx, row, element, lenout=256)`

Return an element of an entry in a column of character type in a specified row.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekgc_c.html

Parameters

- **selidx** (int) – Index of parent column in SELECT clause.
- **row** (int) – Row to fetch from.
- **element** (int) – Index of element, within column entry, to fetch.
- **lenout** (int) – Maximum length of column element.

Return type Tuple[str, int, bool]

Returns Character string element of column entry, Flag indicating whether column entry was null.

`spiceypy.spiceypy.ekgd(selidx, row, element)`

Return an element of an entry in a column of double precision type in a specified row.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekgd_c.html

Parameters

- **selidx** (*int*) – Index of parent column in SELECT clause.
- **row** (*int*) – Row to fetch from.
- **element** (*int*) – Index of element, within column entry, to fetch.

Return type `Tuple[float, int, bool]`

Returns Double precision element of column entry, Flag indicating whether column entry was null.

`spiceypy.spiceypy.ekgi(selidx, row, element)`

Return an element of an entry in a column of integer type in a specified row.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekgi_c.html

Parameters

- **selidx** (*int*) – Index of parent column in SELECT clause.
- **row** (*int*) – Row to fetch from.
- **element** (*int*) – Index of element, within column entry, to fetch.

Return type `Tuple[int, int, bool]`

Returns Integer element of column entry, Flag indicating whether column entry was null.

`spiceypy.spiceypy.ekifld(handle, tabnam, ncols, nrows, cnmlen, cnames, declen, decls)`

Initialize a new E-kernel segment to allow fast writing.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekifld_c.html

Parameters

- **handle** (*int*) – File handle.
- **tabnam** (*str*) – Table name.
- **ncols** (*int*) – Number of columns in the segment.
- **nrows** (*int*) – Number of rows in the segment.
- **cnmlen** (*int*) – Length of names in in column name array.
- **cnames** (*Iterable[str]*) – Names of columns.
- **declen** (*int*) – Length of declaration strings in declaration array.
- **decls** (*Iterable[str]*) – Declarations of columns.

Return type `Tuple[int, ndarray]`

Returns Segment number, Array of record pointers.

`spiceypy.spiceypy.ekinsr(handle, segno, recno)`

Add a new, empty record to a specified E-kernel segment at a specified index.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekinsr_c.html

Parameters

- **handle** (*int*) – File handle.

- **segno** (*int*) – Segment number.
- **recno** (*int*) – Record number.

Return type *None*

`spiceypy.spiceypy.eklef` (*fname*)

Load an EK file, making it accessible to the EK readers.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/eklef_c.html

Parameters **fname** (*str*) – Name of EK file to load.

Return type *int*

Returns File handle of loaded EK file.

`spiceypy.spiceypy.eknelt` (*selidx*, *row*)

Return the number of elements in a specified column entry in the current row.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/eknelt_c.html

Parameters

- **selidx** (*int*) – Index of parent column in SELECT clause.
- **row** (*int*) – Row containing element.

Return type *int*

Returns The number of elements in entry in current row.

`spiceypy.spiceypy.eknseg` (*handle*)

Return the number of segments in a specified EK.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/eknseg_c.html

Parameters **handle** (*int*) – EK file handle.

Return type *int*

Returns The number of segments in the specified E-kernel.

`spiceypy.spiceypy.ekntab` ()

Return the number of loaded EK tables.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekntab_c.html

Return type *int*

Returns The number of loaded EK tables.

`spiceypy.spiceypy.ekopn` (*fname*, *ifname*, *ncomch*)

Open a new E-kernel file and prepare the file for writing.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekopn_c.html

Parameters

- **fname** (*str*) – Name of EK file.
- **ifname** (*str*) – Internal file name.
- **ncomch** (*int*) – The number of characters to reserve for comments.

Return type *int*

Returns Handle attached to new EK file.

`spiceypy.spiceypy.ekopr(fname)`
 Open an existing E-kernel file for reading.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekopr_c.html

Parameters `fname` (`str`) – Name of EK file.

Return type `int`

Returns Handle attached to EK file.

`spiceypy.spiceypy.ekops()`
 Open a scratch (temporary) E-kernel file and prepare the file for writing.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekops_c.html

Return type `int`

Returns Handle attached to new EK file.

`spiceypy.spiceypy.ekopw(fname)`
 Open an existing E-kernel file for writing.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekopw_c.html

Parameters `fname` (`str`) – Name of EK file.

Return type `int`

Returns Handle attached to EK file.

`spiceypy.spiceypy.ekpsel(query, msglen, tablen, collen)`
 Parse the SELECT clause of an EK query, returning full particulars concerning each selected item.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekpsel_c.html note: oddly docs at url are incomplete/incorrect.

Parameters

- **query** (`str`) – EK query.
- **msglen** (`int`) – Available space in the output error message string.
- **tablen** (`int`) – UNKNOWN? Length of Table?
- **collen** (`int`) – UNKNOWN? Length of Column?

Return type `Tuple[int, ndarray, ndarray, ndarray, ndarray, Iterable[str], Iterable[str], int, str]`

Returns Number of items in SELECT clause of query, Begin positions of expressions in SELECT clause, End positions of expressions in SELECT clause, Data types of expressions, Classes of expressions, Names of tables qualifying SELECT columns, Names of columns in SELECT clause of query, Error flag, Parse error message.

`spiceypy.spiceypy.ekrcec(handle, segno, recno, column, lenout, nelts=100)`
 Read data from a character column in a specified EK record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekrcec_c.html

Parameters

- **handle** (`int`) – Handle attached to EK file.
- **segno** (`int`) – Index of segment containing record.
- **recno** (`int`) – Record from which data is to be read.

- **column** (str) – Column name.
- **lenout** (int) – Maximum length of output strings.
- **nelts** (int) – Number of elements to allow for (default=100)

Return type Tuple[int, Iterable[str], bool]

Returns Number of values in column entry, Character values in column entry, Flag indicating whether column entry is null.

`spiceypy.spiceypy.ekrced(handle, segno, recno, column, nelts=100)`

Read data from a double precision column in a specified EK record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekrced_c.html

Parameters

- **handle** (int) – Handle attached to EK file.
- **segno** (int) – Index of segment containing record.
- **recno** (int) – Record from which data is to be read.
- **column** (str) – Column name.
- **nelts** (int) – room for data default 100

Return type Tuple[int, ndarray, bool]

Returns Number of values in column entry, Float values in column entry, Flag indicating whether column entry is null.

`spiceypy.spiceypy.ekrcei(handle, segno, recno, column, nelts=100)`

Read data from an integer column in a specified EK record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekrcei_c.html

Parameters

- **handle** (int) – Handle attached to EK file.
- **segno** (int) – Index of segment containing record.
- **recno** (int) – Record from which data is to be read.
- **column** (str) – Column name.
- **nelts** (int) – room for data default 100

Return type Tuple[int, ndarray, bool]

Returns Number of values in column entry, Integer values in column entry, Flag indicating whether column entry is null.

`spiceypy.spiceypy.ekssum(handle, segno)`

Return summary information for a specified segment in a specified EK.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekssum_c.html

Parameters

- **handle** (int) – Handle of EK.
- **segno** (int) – Number of segment to be summarized.

Return type *SpiceEKSegSum*

Returns EK segment summary.

`spiceypy.spiceypy.ektnam(n, lenout=256)`

Return the name of a specified, loaded table.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ektnam_c.html

Parameters

- **n** (`int`) – Index of table.
- **lenout** (`int`) – Maximum table name length.

Return type `str`

Returns Name of table.

`spiceypy.spiceypy.ekucec(handle, segno, recno, column, nvals, cvals, isnull)`

Update a character column entry in a specified EK record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekucec_c.html

Parameters

- **handle** (`int`) – EK file handle.
- **segno** (`int`) – Index of segment containing record.
- **recno** (`int`) – Record to which data is to be updated.
- **column** (`str`) – Column name.
- **nvals** (`int`) – Number of values in new column entry.
- **cvals** (`Iterable[str]`) – Character values comprising new column entry.
- **isnull** (`bool`) – Flag indicating whether column entry is null.

Return type `None`

`spiceypy.spiceypy.ekuced(handle, segno, recno, column, nvals, dvals, isnull)`

Update a double precision column entry in a specified EK record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekuced_c.html

Parameters

- **handle** (`int`) – EK file handle.
- **segno** (`int`) – Index of segment containing record.
- **recno** (`int`) – Record to which data is to be updated.
- **column** (`str`) – Column name.
- **nvals** (`int`) – Number of values in new column entry.
- **dvals** (`Iterable[float]`) – Double precision values comprising new column entry.
- **isnull** (`bool`) – Flag indicating whether column entry is null.

Return type `None`

`spiceypy.spiceypy.ekupei(handle, segno, recno, column, nvals, ival, isnull)`

Update an integer column entry in a specified EK record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekupei_c.html

Parameters

- **handle** (`int`) – EK file handle.

- **segno** (*int*) – Index of segment containing record.
- **recno** (*int*) – Record to which data is to be updated.
- **column** (*str*) – Column name.
- **nvals** (*int*) – Number of values in new column entry.
- **ivals** (*Iterable[int]*) – Integer values comprising new column entry.
- **isnull** (*bool*) – Flag indicating whether column entry is null.

Return type *None*

`spiceypy.spiceypy.ekuef` (*handle*)

Unload an EK file, making its contents inaccessible to the EK reader routines, and clearing space in order to allow other EK files to be loaded.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ekuef_c.html

Parameters **handle** (*int*) – Handle of EK file.

Return type *None*

`spiceypy.spiceypy.el2cgv` (*ellipse*)

Convert an ellipse to a center vector and two generating vectors. The selected generating vectors are semi-axes of the ellipse.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/el2cgv_c.html

Parameters **ellipse** (*Ellipse*) – An Ellipse

Return type *Tuple[ndarray, ndarray, ndarray]*

Returns Center and semi-axes of ellipse.

`spiceypy.spiceypy.elemc` (*item, inset*)

Determine whether an item is an element of a character set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/elemc_c.html

Parameters

- **item** (*str*) – Item to be tested.
- **inset** (*SpiceCell*) – Set to be tested.

Return type *bool*

Returns True if item is an element of set.

`spiceypy.spiceypy.elemnd` (*item, inset*)

Determine whether an item is an element of a double precision set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/elemd_c.html

Parameters

- **item** (*float*) – Item to be tested.
- **inset** (*SpiceCell*) – Set to be tested.

Return type *bool*

Returns True if item is an element of set.

`spiceypy.spiceypy.elemi(item, inset)`

Determine whether an item is an element of an integer set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/elemi_c.html

Parameters

- **item** (`int`) – Item to be tested.
- **inset** (`SpiceCell`) – Set to be tested.

Return type `bool`

Returns True if item is an element of set.

`spiceypy.spiceypy.eqncpv(et, epoch, eqel, rapol, decpol)`

Compute the state (position and velocity of an object whose trajectory is described via equinoctial elements relative to some fixed plane (usually the equatorial plane of some planet).

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/eqncpv_c.html

Parameters

- **et** (`float`) – Epoch in seconds past J2000 to find state.
- **epoch** (`float`) – Epoch of elements in seconds past J2000.
- **eqel** (`Iterable[float]`) – Array of equinoctial elements
- **rapol** (`float`) – Right Ascension of the pole of the reference plane.
- **decpol** (`float`) – Declination of the pole of the reference plane.

Return type `ndarray`

Returns State of the object described by eqel.

`spiceypy.spiceypy.eqstr(a, b)`

Determine whether two strings are equivalent.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/eqstr_c.html

Parameters

- **a** (`str`) – Arbitrary character string.
- **b** (`str`) – Arbitrary character string.

Return type `bool`

Returns True if A and B are equivalent.

`spiceypy.spiceypy.erract(op, lenout, action=None)`

Retrieve or set the default error action. spiceypy sets the default error action to “report” on init.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/erract_c.html

Parameters

- **op** (`str`) – peration, “GET” or “SET”.
- **lenout** (`int`) – Length of list for output.
- **action** (`Optional[str]`) – Error response action.

Return type `str`

Returns Error response action.

`spiceypy.spiceypy.errch(marker, string)`

Substitute a character string for the first occurrence of a marker in the current long error message.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/errch_c.html

Parameters

- **marker** (`str`) – A substring of the error message to be replaced.
- **string** (`str`) – The character string to substitute for marker.

Return type `None`

`spiceypy.spiceypy.errdev(op, lenout, device)`

Retrieve or set the name of the current output device for error messages.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/errdev_c.html

Parameters

- **op** (`str`) – The operation, “GET” or “SET”.
- **lenout** (`int`) – Length of device for output.
- **device** (`str`) – The device name.

Return type `str`

Returns The device name.

`spiceypy.spiceypy.errdp(marker, number)`

Substitute a double precision number for the first occurrence of a marker found in the current long error message.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/errdp_c.html

Parameters

- **marker** (`str`) – A substring of the error message to be replaced.
- **number** (`float`) – The d.p. number to substitute for marker.

Return type `None`

`spiceypy.spiceypy.errint(marker, number)`

Substitute an integer for the first occurrence of a marker found in the current long error message.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/errint_c.html

Parameters

- **marker** (`str`) – A substring of the error message to be replaced.
- **number** (`int`) – The integer to substitute for marker.

Return type `None`

`spiceypy.spiceypy.errprt(op, lenout, inlist)`

Retrieve or set the list of error message items to be output when an error is detected.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/errprt_c.html

Parameters

- **op** (`str`) – The operation, “GET” or “SET”.
- **lenout** (`int`) – Length of list for output.
- **inlist** (`str`) – Specification of error messages to be output.

Return type `str`

Returns A list of error message items.

`spiceypy.spiceypy.esrchc(value, array)`

Search for a given value within a character string array. Return the index of the first equivalent array entry, or -1 if no equivalent element is found.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/esrchc_c.html

Parameters

- **value** (`str`) – Key value to be found in array.
- **array** (`Sequence[str]`) – Character string array to search.

Return type `int`

Returns The index of the first array entry equivalent to value, or -1 if none is found.

`spiceypy.spiceypy.et2lst(et, body, lon, typein, timlen=256, ampm len=256)`

Given an ephemeris epoch, compute the local solar time for an object on the surface of a body at a specified longitude.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/et2lst_c.html

Parameters

- **et** (`float`) – Epoch in seconds past J2000 epoch.
- **body** (`int`) – ID-code of the body of interest.
- **lon** (`float`) – Longitude of surface point (RADIANS).
- **typein** (`str`) – Type of longitude “PLANETOCENTRIC”, etc.
- **timlen** (`int`) – Available room in output time string.
- **ampm len** (`int`) – Available room in output ampm string.

Return type `Tuple[int, int, int, str, str]`

Returns Local hour on a “24 hour” clock, Minutes past the hour, Seconds past the minute, String giving local time on 24 hour clock, String giving time on A.M. / P.M. scale.

`spiceypy.spiceypy.et2utc(et, format_str, prec, lenout=256)`

Convert an input time from ephemeris seconds past J2000 to Calendar, Day-of-Year, or Julian Date format, UTC.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/et2utc_c.html

Parameters

- **et** (`Union[float, Iterable[float]]`) – Input epoch, given in ephemeris seconds past J2000.
- **format_str** (`str`) – Format of output epoch.
- **prec** (`int`) – Digits of precision in fractional seconds or days.
- **lenout** (`int`) – The length of the output string plus 1.

Return type `Union[ndarray, str]`

Returns Output time string in UTC

`spiceypy.spiceypy.etcal(et, lenout=256)`

Convert from an ephemeris epoch measured in seconds past the epoch of J2000 to a calendar string format using a formal calendar free of leapseconds.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/etcal_c.html

Parameters

- **et** (Union[float, ndarray]) – Ephemeris time measured in seconds past J2000.
- **lenout** (int) – Length of output string.

Return type Union[str, Iterable[str]]

Returns A standard calendar representation of et.

`spiceypy.spiceypy.eul2m(angle3, angle2, angle1, axis3, axis2, axis1)`

Construct a rotation matrix from a set of Euler angles.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/eul2m_c.html

Parameters

- **angle3** (float) – Rotation angle about third rotation axis (radians).
- **angle2** (float) – Rotation angle about second rotation axis (radians).
- **angle1** (float) – Rotation angle about first rotation axis (radians).
- **axis3** (int) – Axis number of third rotation axis.
- **axis2** (int) – Axis number of second rotation axis.
- **axis1** (int) – Axis number of first rotation axis.]

Return type ndarray

Returns Product of the 3 rotations.

`spiceypy.spiceypy.eul2xf(eulang, axisa, axisb, axisc)`

This routine computes a state transformation from an Euler angle factorization of a rotation and the derivatives of those Euler angles.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/eul2xf_c.html

Parameters

- **eulang** (Sequence[float]) – An array of Euler angles and their derivatives.
- **axisa** (int) – Axis A of the Euler angle factorization.
- **axisb** (int) – Axis B of the Euler angle factorization.
- **axisc** (int) – Axis C of the Euler angle factorization.

Return type ndarray

Returns A state transformation matrix.

`spiceypy.spiceypy.ev2lin(et, geophs, elems)`

This routine evaluates NORAD two-line element data for near-earth orbiting spacecraft (that is spacecraft with orbital periods less than 225 minutes).

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/spicelib/ev2lin.html

Parameters

- **et** (float) – Epoch in seconds past ephemeris epoch J2000.
- **geophs** (Sequence[float]) – Geophysical constants
- **elems** (Sequence[float]) – Two-line element data

Return type ndarray

Returns Evaluated state

`spiceypy.spiceypy.exists(fname)`

Determine whether a file exists.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/exists_c.html

Parameters `fname` (`str`) – Name of the file in question.

Return type `bool`

Returns True if the file exists, False otherwise.

`spiceypy.spiceypy.expool(name)`

Confirm the existence of a kernel variable in the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/expool_c.html

Parameters `name` (`str`) – Name of the variable whose value is to be returned.

Return type `bool`

Returns True when the variable is in the pool.

`spiceypy.spiceypy.failed()`

True if an error condition has been signalled via `sigerr_c`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/failed_c.html

Return type `bool`

Returns a boolean

`spiceypy.spiceypy.fn2lun(fname)`

Internal undocumented command for mapping name of open file to its FORTRAN (F2C) logical unit.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/spicelib/fn2lun.html

Parameters `fname` (`str`) – name of the file to be mapped to its logical unit.

Return type `int`

Returns the FORTRAN (F2C) logical unit associated with the filename.

`spiceypy.spiceypy.found_check()`

Temporarily enables spiceypy default behavior which raises exceptions for false found flags for certain spice functions. All spice functions executed within the context manager will check the found flag return parameter and the found flag will be removed from the return for the given function. For Example `bodc2n` in spiceypy is normally called like:

```
name = spice.bodc2n(399)
```

With the possibility that an exception is thrown in the even of a invalid ID:

```
name = spice.bodc2n(-999991) # throws a SpiceyError
```

With this function however, we can use it as a context manager to do this:

```
with spice.found_check():
    found = spice.bodc2n(-999991) # will raise an exception!
```

Within the context any spice functions called that normally check the found flags will pass through the check without raising an exception if they are false.

Return type `Iterator[None]`

`spiceypy.spiceypy.found_check_off()`

Method that turns off found catching

Return type `None`

`spiceypy.spiceypy.found_check_on()`

Method that turns on found catching

Return type `None`

`spiceypy.spiceypy.fovray(inst, raydir, rframe, abcorr, observer, et)`

Determine if a specified ray is within the field-of-view (FOV) of a specified instrument at a given time.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/fovray_c.html

Parameters

- **inst** (`str`) – Name or ID code string of the instrument.
- **raydir** (`Iterable[float]`) – Ray’s direction vector.
- **rframe** (`str`) – Body-fixed, body-centered frame for target body.
- **abcorr** (`str`) – Aberration correction flag.
- **observer** (`str`) – Name or ID code string of the observer.
- **et** (`float`) – Time of the observation (seconds past J2000).

Return type `bool`

Returns Visibility flag

`spiceypy.spiceypy.fovtrg(inst, target, tshape, tframe, abcorr, observer, et)`

Determine if a specified ephemeris object is within the field-of-view (FOV) of a specified instrument at a given time.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/fovtrg_c.html

Parameters

- **inst** (`str`) – Name or ID code string of the instrument.
- **target** (`str`) – Name or ID code string of the target.
- **tshape** (`str`) – Type of shape model used for the target.
- **tframe** (`str`) – Body-fixed, body-centered frame for target body.
- **abcorr** (`str`) – Aberration correction flag.
- **observer** (`str`) – Name or ID code string of the observer.
- **et** (`float`) – Time of the observation (seconds past J2000).

Return type `bool`

Returns Visibility flag

`spiceypy.spiceypy.frame(x)`

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/frame_c.html

Parameters **x** (`Iterable[float]`) – Input vector. A parallel unit vector on output.

Return type `Tuple[ndarray, ndarray, ndarray]`

Returns a tuple of 3 list[3]

`spiceypy.spiceypy.finfo(frinfo)`

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/frinfo_c.html

Parameters `frinfo` (`int`) – the idcode for some frame.

Return type `Tuple[int, int, int, bool]`

Returns a tuple of attributes associated with the frame.

`spiceypy.spiceypy.frmnam(frinfo, lenout=256)`

Retrieve the name of a reference frame associated with a SPICE ID code.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/frmnam_c.html

Parameters

- **frinfo** (`int`) – an integer code for a reference frame
- **lenout** (`int`) – Maximum length of output string.

Return type `str`

Returns the name associated with the reference frame.

`spiceypy.spiceypy.ftncls(unit)`

Close a file designated by a Fortran-style integer logical unit.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ftncls_c.html

Parameters `unit` (`int`) – Fortran-style logical unit.

Return type `None`

`spiceypy.spiceypy.furnsh(path)`

Load one or more SPICE kernels into a program.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/furnsh_c.html

Parameters `path` (`Union[str, Iterable[str]]`) – one or more paths to kernels

Return type `None`

`spiceypy.spiceypy.gcpool(name, start, room, lenout=256)`

Return the character value of a kernel variable from the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gcpool_c.html

Parameters

- **name** (`str`) – Name of the variable whose value is to be returned.
- **start** (`int`) – Which component to start retrieving for name.
- **room** (`int`) – The largest number of values to return.
- **lenout** (`int`) – The length of the output string.

Return type `Tuple[Iterable[str], bool]`

Returns Values associated with name.

`spiceypy.spiceypy.gdpool(name, start, room)`

Return the d.p. value of a kernel variable from the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gdpool_c.html

Parameters

- **name** (`str`) – Name of the variable whose value is to be returned.

- **start** (int) – Which component to start retrieving for name.
- **room** (int) – The largest number of values to return.

Return type Tuple[ndarray, bool]

Returns Values associated with name.

`spiceypy.spiceypy.georec(lon, lat, alt, re, f)`

Convert geodetic coordinates to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/georec_c.html

Parameters

- **lon** (float) – Geodetic longitude of point (radians).
- **lat** (float) – Geodetic latitude of point (radians).
- **alt** (float) – Altitude of point above the reference spheroid.
- **re** (float) – Equatorial radius of the reference spheroid.
- **f** (float) – Flattening coefficient.

Return type ndarray

Returns Rectangular coordinates of point.

`spiceypy.spiceypy.get_found_catch_state()`

Returns the current found catch state

Return type bool

Returns

`spiceypy.spiceypy.getelm(frstyr, lineIn, lines)`

Given a the “lines” of a two-line element set, parse the lines and return the elements in units suitable for use in SPICE software.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/getelm_c.html

Parameters

- **frstyr** (int) – Year of earliest representable two-line elements.
- **lineIn** (int) – Length of strings in lines array.
- **lines** (Iterable[str]) – A pair of “lines” containing two-line elements.

Return type Tuple[float, ndarray]

Returns The epoch of the elements in seconds past J2000, The elements converted to SPICE units.

`spiceypy.spiceypy.getfat(file)`

Determine the file architecture and file type of most SPICE kernel files.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/getfat_c.html

Parameters **file** (str) – The name of a file to be examined.

Return type Tuple[str, str]

Returns The architecture of the kernel file, The type of the kernel file.

`spiceypy.spiceypy.getfov(instid, room, shapelen=256, framelen=256)`

This routine returns the field-of-view (FOV) parameters for a specified instrument.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/getfov_c.html

Parameters

- **instid** (*int*) – NAIF ID of an instrument.
- **room** (*int*) – Maximum number of vectors that can be returned.
- **shapelen** (*int*) – Space available in the string shape.
- **framelen** (*int*) – Space available in the string frame.

Return type `Tuple[str, str, ndarray, int, ndarray]`

Returns Instrument FOV shape, Name of the frame in which FOV vectors are defined, Boresight vector, Number of boundary vectors returned, FOV boundary vectors.

`spiceypy.spiceypy.getmsg(option, lenout=256)`

Retrieve the current short error message, the explanation of the short error message, or the long error message.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/getmsg_c.html

Parameters

- **option** (*str*) – Indicates type of error message.
- **lenout** (*int*) – Available space in the output string msg.

Return type `str`

Returns The error message to be retrieved.

`spiceypy.spiceypy.gfbail()`

Indicate whether an interrupt signal (SIGINT) has been received.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfbail_c.html

Return type `bool`

Returns True if an interrupt signal has been received by the GF handler.

`spiceypy.spiceypy.gfclrh()`

Clear the interrupt signal handler status, so that future calls to `gfbail()` will indicate no interrupt was received.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfclrh_c.html

Return type `None`

`spiceypy.spiceypy.gfdist(target, abcorr, obsrvr, relate, refval, adjust, step, nintvls, cnfine, result=None)`

Return the time window over which a specified constraint on observer-target distance is met.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfdist_c.html

Parameters

- **target** (*str*) – Name of the target body.
- **abcorr** (*str*) – Aberration correction flag.
- **obsrvr** (*str*) – Name of the observing body.
- **relate** (*str*) – Relational operator.
- **refval** (*int*) – Reference value.
- **adjust** (*float*) – Adjustment value for absolute extrema searches.
- **step** (*float*) – Step size used for locating extrema and roots.
- **nintvls** (*int*) – Workspace window interval count.

- **cnfine** (*SpiceCell*) – SPICE window to which the search is confined.
- **result** (Optional[*SpiceCell*]) – Optional SPICE window containing results.

Return type *SpiceCell*

`spiceypy.spiceypy.gfevnt` (*udstep*, *udrefn*, *gquant*, *qnpars*, *lenvals*, *qpnams*, *qcpars*, *qdpars*, *qipars*, *qlpars*, *op*, *refval*, *tol*, *adjust*, *rpt*, *udrepi*, *udrepu*, *udrepf*, *nintvls*, *bail*, *udbail*, *cnfine*, *result=None*)

Determine time intervals when a specified geometric quantity satisfies a specified mathematical condition.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfevnt_c.html

Parameters

- **udstep** (CFunctionType) – Name of the routine that computes and returns a
- **udrefn** (CFunctionType) – Name of the routine that computes a refined time
- **gquant** (str) – Type of geometric quantity
- **qnpars** (int) – Number of quantity definition parameters
- **lenvals** (int) – Length of strings in qpnams and qcpars
- **qpnams** (Iterable[str]) – Names of quantity definition parameters
- **qcpars** (Iterable[str]) – Array of character quantity definition parameters
- **qdpars** (Iterable[float]) – Array of double precision quantity definition
- **qipars** (Iterable[int]) – Array of integer quantity definition parameters
- **qlpars** (Iterable[int]) – Array of logical quantity definition parameters
- **op** (str) – Operator that either looks for an extreme value
- **refval** (float) – Reference value
- **tol** (float) – Convergence tolerance in seconds
- **adjust** (float) – Absolute extremum adjustment value
- **rpt** (int) – Progress reporter on TRUE or off FALSE
- **udrepi** (CFunctionType) – Function that initializes progress reporting
- **udrepu** (CFunctionType) – Function that updates the progress report
- **udrepf** (CFunctionType) – Function that finalizes progress reporting
- **nintvls** (int) – Workspace window interval count
- **bail** (int) – Logical indicating program interrupt monitoring
- **udbail** (CFunctionType) – Name of a routine that signals a program interrupt
- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted
- **result** (Optional[*SpiceCell*]) – Optional SPICE window containing results

`spiceypy.spiceypy.gffove` (*inst*, *tshape*, *raydir*, *target*, *tframe*, *abcorr*, *obsrvr*, *tol*, *udstep*, *udrefn*, *rpt*, *udrepi*, *udrepu*, *udrepf*, *bail*, *udbail*, *cnfine*, *result=None*)

Determine time intervals when a specified target body or ray intersects the space bounded by the field-of-view (FOV) of a specified instrument. Report progress and handle interrupts if so commanded.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gffove_c.html

Parameters

- **inst** (*str*) – Name of the instrument
- **tshape** (*str*) – Type of shape model used for target body
- **raydir** (*Iterable[float]*) – Ray's direction vector
- **target** (*str*) – Name of the target body
- **tframe** (*str*) – Body fixed body centered frame for target body
- **abcorr** (*str*) – Aberration correction flag
- **obsrvr** (*str*) – Name of the observing body
- **tol** (*float*) – Convergence tolerance in seconds
- **udstep** (*CFunctionType*) – Name of the routine that returns a time step
- **udrefn** (*CFunctionType*) – Name of the routine that computes a refined time
- **rpt** (*int*) – Progress report flag
- **udrepi** (*CFunctionType*) – Function that initializes progress reporting.
- **udrepu** (*CFunctionType*) – Function that updates the progress report
- **udrepf** (*CFunctionType*) – Function that finalizes progress reporting
- **bail** (*int*) – Logical indicating program interrupt monitoring
- **udbail** (*CFunctionType*) – Name of a routine that signals a program interrupt
- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted
- **result** (*Optional[SpiceCell]*) – Optional SPICE window containing results

`spiceypy.spiceypy.gfilum` (*method, angtyp, target, illumn, fixref, abcorr, obsrvr, spoint, relate, refval, adjust, step, nintvls, cnfine, result=None*)

Return the time window over which a specified constraint on the observed phase, solar incidence, or emission angle at a specified target body surface point is met.

Parameters

- **method** (*str*) – Shape model used to represent the surface of the target body.
- **angtyp** (*str*) – The type of illumination angle for which a search is to be performed.
- **target** (*str*) – Name of a target body.
- **illumn** (*str*) – Name of the illumination source.
- **fixref** (*str*) – Name of the body-fixed, body-centered reference frame associated with the target body.
- **abcorr** (*str*) – The aberration corrections to be applied.
- **obsrvr** (*str*) – Name of an observing body.
- **spoint** (*Iterable[float]*) – Body-fixed coordinates of a target surface point.
- **relate** (*str*) – Relational operator used to define a constraint on a specified illumination angle.
- **refval** (*float*) – Reference value used with 'relate' to define an equality or inequality to be satisfied by the specified illumination angle.
- **adjust** (*float*) – Parameter used to modify searches for absolute extrema.
- **step** (*float*) – Step size to be used in the search.

- **nintvls** (*int*) – Number of intervals that can be accommodated by each of the dynamically allocated workspace windows used internally by this routine.
- **cnfine** (*SpiceCell*) – Window that confines the time period over which the specified search is conducted. This can be updated by *gfilum*
- **result** (Optional[*SpiceCell*]) – Optional SPICE Window of intervals in the confinement window that the illumination angle constraint is satisfied.

Return type *SpiceCell*

`spiceypy.spiceypy.gfinth` (*sigcode*)

Respond to the interrupt signal SIGINT: save an indication that the signal has been received. This routine restores itself as the handler for SIGINT.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfinth_c.html

Parameters **sigcode** (*int*) – Interrupt signal ID code.

Return type *None*

`spiceypy.spiceypy.gfocce` (*occtyp*, *front*, *fshape*, *fframe*, *back*, *bshape*, *bframe*, *abcorr*, *obsrvr*, *tol*, *udstep*, *udrefn*, *rpt*, *udrepi*, *udrepu*, *udrepf*, *bail*, *udbail*, *cnfine*, *result=None*)

Determine time intervals when an observer sees one target occulted by another. Report progress and handle interrupts if so commanded.

The surfaces of the target bodies may be represented by triaxial ellipsoids or by topographic data provided by DSK files.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfocce_c.html

Parameters

- **occtyp** (*str*) – Type of occultation
- **front** (*str*) – Name of body occulting the other
- **fshape** (*str*) – Type of shape model used for front body
- **fframe** (*str*) – Body fixed body centered frame for front body
- **back** (*str*) – Name of body occulted by the other
- **bshape** (*str*) – Type of shape model used for back body
- **bframe** (*str*) – Body fixed body centered frame for back body
- **abcorr** (*str*) – Aberration correction flag
- **obsrvr** (*str*) – Name of the observing body
- **tol** (*float*) – Convergence tolerance in seconds
- **udstep** (*CFunctionType*) – Name of the routine that returns a time step
- **udrefn** (*CFunctionType*) – Name of the routine that computes a refined time
- **rpt** (*int*) – Progress report flag
- **udrepi** (*CFunctionType*) – Function that initializes progress reporting.
- **udrepu** (*CFunctionType*) – Function that updates the progress report
- **udrepf** (*CFunctionType*) – Function that finalizes progress reporting
- **bail** (*int*) – Logical indicating program interrupt monitoring
- **udbail** (*CFunctionType*) – Name of a routine that signals a program interrupt

- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted
- **result** (Optional[*SpiceCell*]) – Optional SPICE window containing results.

`spiceypy.spiceypy.gfoclt` (*occtyp*, *front*, *fshape*, *fframe*, *back*, *bshape*, *bframe*, *abcorr*, *obsrvr*, *step*, *cnfine*, *result=None*)

Determine time intervals when an observer sees one target occulted by, or in transit across, another.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfoclt_c.html

Parameters

- **occtyp** (*str*) – Type of occultation.
- **front** (*str*) – Name of body occulting the other.
- **fshape** (*str*) – Type of shape model used for front body.
- **fframe** (*str*) – Body-fixed, body-centered frame for front body.
- **back** (*str*) – Name of body occulted by the other.
- **bshape** (*str*) – Type of shape model used for back body.
- **bframe** (*str*) – Body-fixed, body-centered frame for back body.
- **abcorr** (*str*) – Aberration correction flag.
- **obsrvr** (*str*) – Name of the observing body.
- **step** (*float*) – Step size in seconds for finding occultation events.
- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted.
- **result** (Optional[*SpiceCell*]) – Optional SPICE window containing results.

Return type *SpiceCell*

`spiceypy.spiceypy.gfpa` (*target*, *illmin*, *abcorr*, *obsrvr*, *relate*, *refval*, *adjust*, *step*, *nintvals*, *cnfine*, *result=None*)

Determine time intervals for which a specified constraint on the phase angle between an illumination source, a target, and observer body centers is met.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfpa_c.html

Parameters

- **target** (*str*) – Name of the target body.
- **illmin** (*str*) – Name of the illuminating body.
- **abcorr** (*str*) – Aberration correction flag.
- **obsrvr** (*str*) – Name of the observing body.
- **relate** (*str*) – Relational operator.
- **refval** (*float*) – Reference value.
- **adjust** (*float*) – Adjustment value for absolute extrema searches.
- **step** (*float*) – Step size used for locating extrema and roots.
- **nintvals** (*int*) – Workspace window interval count.
- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted.
- **result** (Optional[*SpiceCell*]) – Optional SPICE window containing results.

Return type *SpiceCell*

`spiceypy.spiceypy.gfposc` (*target, inframe, abcorr, obsrvr, crdsys, coord, relate, refval, adjust, step, nintvals, cnfine, result=None*)

Determine time intervals for which a coordinate of an observer-target position vector satisfies a numerical constraint.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfposc_c.html

Parameters

- **target** (`str`) – Name of the target body.
- **inframe** (`str`) – Name of the reference frame for coordinate calculations.
- **abcorr** (`str`) – Aberration correction flag.
- **obsrvr** (`str`) – Name of the observing body.
- **crdsys** (`str`) – Name of the coordinate system containing COORD
- **coord** (`str`) – Name of the coordinate of interest
- **relate** (`str`) – Relational operator.
- **refval** (`float`) – Reference value.
- **adjust** (`float`) – Adjustment value for absolute extrema searches.
- **step** (`float`) – Step size used for locating extrema and roots.
- **nintvals** (`int`) – Workspace window interval count.
- **cnfine** (`SpiceCell`) – SPICE window to which the search is restricted.
- **result** (`Optional[SpiceCell]`) – Optional SPICE window containing results.

Return type `SpiceCell`

`spiceypy.spiceypy.gfrefn` (*t1, t2, s1, s2*)

For those times when we can't do better, we use a bisection method to find the next time at which to test for state change.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfrefn_c.html

Parameters

- **t1** (`float`) – One of two values bracketing a state change.
- **t2** (`float`) – The other value that brackets a state change.
- **s1** (`Union[bool, int]`) – State at t1.
- **s2** (`Union[bool, int]`) – State at t2.

Return type `float`

Returns New value at which to check for transition.

`spiceypy.spiceypy.gfrepf` ()

Finish a GF progress report.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfrepf_c.html

Return type `None`

`spiceypy.spiceypy.gfrepf` (*window, begmss, endmss*)

This entry point initializes a search progress report.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfrepf_c.html

Parameters

- **window** (*SpiceCell*) – A window over which a job is to be performed.
- **begmss** (*str*) – Beginning of the text portion of the output message.
- **endmss** (*str*) – End of the text portion of the output message.

Return type *None*

`spiceypy.spiceypy.gfrepu` (*ivbeg, ivend, time*)

This function tells the progress reporting system how far a search has progressed.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfrepu_c.html

Parameters

- **ivbeg** (*float*) – Start time of work interval.
- **ivend** (*float*) – End time of work interval.
- **time** (*float*) – Current time being examined in the search process.

Return type *None*

`spiceypy.spiceypy.gfrfov` (*inst, raydir, rframe, abcorr, obsrvr, step, cnfine, result=None*)

Determine time intervals when a specified ray intersects the space bounded by the field-of-view (FOV) of a specified instrument.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfrfov_c.html

Parameters

- **inst** (*str*) – Name of the instrument.
- **raydir** (*ndarray*) – Ray's direction vector.
- **rframe** (*str*) – Reference frame of ray's direction vector.
- **abcorr** (*str*) – Aberration correction flag.
- **obsrvr** (*str*) – Name of the observing body.
- **step** (*float*) – Step size in seconds for finding FOV events.
- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted.
- **result** (*Optional[SpiceCell]*) – Optional SPICE window containing results.

Return type *SpiceCell*

`spiceypy.spiceypy.gfrfrr` (*target, abcorr, obsrvr, relate, refval, adjust, step, nintvals, cnfine, result*)

Determine time intervals for which a specified constraint on the observer-target range rate is met.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfrfrr_c.html

Parameters

- **target** (*str*) – Name of the target body.
- **abcorr** (*str*) – Aberration correction flag.
- **obsrvr** (*str*) – Name of the observing body.
- **relate** (*str*) – Relational operator.
- **refval** (*float*) – Reference value.
- **adjust** (*float*) – Adjustment value for absolute extrema searches.
- **step** (*float*) – Step size used for locating extrema and roots.

- **nintvals** (*int*) – Workspace window interval count.
- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted.
- **result** (*SpiceCell*) – Optional SPICE window containing results.

Return type *SpiceCell*

`spiceypy.spiceypy.gfsep` (*target1, shape1, inframe1, targ2, shape2, inframe2, abcorr, obsrvr, relate, refval, adjust, step, nintvals, cnfine, result=None*)

Determine time intervals when the angular separation between the position vectors of two target bodies relative to an observer satisfies a numerical relationship.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfsep_c.html

Parameters

- **targ1** (*str*) – Name of first body.
- **shape1** (*str*) – Name of shape model describing the first body.
- **inframe1** (*str*) – The body-fixed reference frame of the first body.
- **targ2** (*str*) – Name of second body.
- **shape2** (*str*) – Name of the shape model describing the second body.
- **inframe2** (*str*) – The body-fixed reference frame of the second body.
- **abcorr** (*str*) – Aberration correction flag
- **obsrvr** (*str*) – Name of the observing body.
- **relate** (*str*) – Relational operator.
- **refval** (*float*) – Reference value.
- **adjust** (*float*) – Absolute extremum adjustment value.
- **step** (*float*) – Step size in seconds for finding angular separation events.
- **nintvals** (*int*) – Workspace window interval count.
- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted.
- **result** (*Optional[SpiceCell]*) – Optional SPICE window containing results.

Return type *SpiceCell*

`spiceypy.spiceypy.gfsntc` (*target, fixref, method, abcorr, obsrvr, dref, dvec, crdsys, coord, relate, refval, adjust, step, nintvals, cnfine, result=None*)

Determine time intervals for which a coordinate of an surface intercept position vector satisfies a numerical constraint.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfsntc_c.html

Parameters

- **target** (*str*) – Name of the target body.
- **fixref** (*str*) – Body fixed frame associated with the target.
- **method** (*str*) – Name of method type for surface intercept calculation.
- **abcorr** (*str*) – Aberration correction flag
- **obsrvr** (*str*) – Name of the observing body.
- **dref** (*str*) – Reference frame of direction vector of dvec.

- **dvec** (Iterable[float]) – Pointing direction vector from the observer.
- **crdsys** (str) – Name of the coordinate system containing COORD.
- **coord** (str) – Name of the coordinate of interest
- **relate** (str) – Relational operator.
- **refval** (float) – Reference value.
- **adjust** (float) – Absolute extremum adjustment value.
- **step** (float) – Step size in seconds for finding angular separation events.
- **nintvals** (int) – Workspace window interval count.
- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted.
- **result** (Optional[*SpiceCell*]) – Optional SPICE window containing results.

Return type *SpiceCell*

`spiceypy.spiceypy.gfsstp(step)`

Set the step size to be returned by *gfstep()*.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfsstp_c.html

Parameters **step** (float) – Time step to take.

Return type None

`spiceypy.spiceypy.gfstep(time)`

Return the time step set by the most recent call to *gfsstp()*.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfstep_c.html

Parameters **time** (float) – Ignored ET value.

Return type float

Returns Time step to take.

`spiceypy.spiceypy.gfstol(value)`

Override the default GF convergence value used in the high level GF routines.

Default value is 1.0e-6

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfstol_c.html

Parameters **value** (float) – Double precision value returned or to store.

Return type None

`spiceypy.spiceypy.gfsubc(target, fixref, method, abcorr, obsrvr, crdsys, coord, relate, refval, adjust, step, nintvals, cnfine, result)`

Determine time intervals for which a coordinate of an subpoint position vector satisfies a numerical constraint.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfsubc_c.html

Parameters

- **target** (str) – Name of the target body.
- **fixref** (str) – Body fixed frame associated with the target.
- **method** (str) – Name of method type for subpoint calculation.
- **abcorr** (str) – Aberration correction flag
- **obsrvr** (str) – Name of the observing body.

- **crdsys** (*str*) – Name of the coordinate system containing COORD.
- **coord** (*str*) – Name of the coordinate of interest
- **relate** (*str*) – Relational operator.
- **refval** (*float*) – Reference value.
- **adjust** (*float*) – Adjustment value for absolute extrema searches.
- **step** (*float*) – Step size used for locating extrema and roots.
- **nintvals** (*int*) – Workspace window interval count.
- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted.
- **result** (*SpiceCell*) – Optional SPICE window containing results.

Return type *SpiceCell*

`spiceypy.spiceypy.gftfov (inst, target, tshape, tframe, abcorr, obsrvr, step, cnfine, result=None)`

Determine time intervals when a specified ephemeris object intersects the space bounded by the field-of-view (FOV) of a specified instrument.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gftfov_c.html

Parameters

- **inst** (*str*) – Name of the instrument.
- **target** (*str*) – Name of the target body.
- **tshape** (*str*) – Type of shape model used for target body.
- **tframe** (*str*) – Body-fixed, body-centered frame for target body.
- **abcorr** (*str*) – Aberration correction flag.
- **obsrvr** (*str*) – Name of the observing body.
- **step** (*float*) – Step size in seconds for finding FOV events.
- **cnfine** (*SpiceCell*) – SPICE window to which the search is restricted.
- **result** (Optional [*SpiceCell*]) – Optional pass-in SpiceCell for results

Return type *SpiceCell*

Returns SpiceCell containing set of time intervals, within the confinement period, when the target body is visible

`spiceypy.spiceypy.gfudb (udfuns, udfunb, step, cnfine, result)`

Perform a GF search on a user defined boolean quantity.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfudb_c.html

Parameters

- **udfuns** – Name of the routine that computes a scalar quantity of interest corresponding to an ‘et’.
- **udfunb** – Name of the routine returning the boolean value corresponding to an ‘et’.
- **step** – Step size used for locating extrema and roots.
- **cnfine** – SPICE window to which the search is restricted.
- **result** – SPICE window containing results.

Returns result

`spiceypy.spiceypy.gfuds` (*udfuns, udqdec, relate, refval, adjust, step, nintvls, cnfine, result*)
 Perform a GF search on a user defined scalar quantity.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gfuds_c.html

Parameters

- **udfuns** – Name of the routine that computes the scalar quantity of interest at some time.
- **udqdec** – Name of the routine that computes whether the scalar quantity is decreasing.
- **relate** – Operator that either looks for an extreme value (max, min, local, absolute) or compares the geometric quantity value and a number.
- **refval** – Value used as reference for scalar quantity condition.
- **adjust** – Allowed variation for absolute extremal geometric conditions.
- **step** – Step size used for locating extrema and roots.
- **nintvls** – Workspace window interval count.
- **cnfine** – SPICE window to which the search is restricted.
- **result** – SPICE window containing results.

Returns `result`

`spiceypy.spiceypy.gipool` (*name, start, room*)
 Return the integer value of a kernel variable from the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gipool_c.html

Parameters

- **name** (`str`) – Name of the variable whose value is to be returned.
- **start** (`int`) – Which component to start retrieving for name.
- **room** (`int`) – The largest number of values to return.

Return type `Tuple[ndarray, bool]`

Returns Values associated with name.

`spiceypy.spiceypy.gnpool` (*name, start, room, lenout=256*)
 Return names of kernel variables matching a specified template.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/gnpool_c.html

Parameters

- **name** (`str`) – Template that names should match.
- **start** (`int`) – Index of first matching name to retrieve.
- **room** (`int`) – The largest number of values to return.
- **lenout** (`int`) – Length of strings in output array kvars.

Return type `Tuple[Iterable[str], bool]`

Returns Kernel pool variables whose names match name.

`spiceypy.spiceypy.halfpi` ()
 Return half the value of pi (the ratio of the circumference of a circle to its diameter).

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/halfpi_c.html

Return type `float`

Returns Half the value of pi.

`spiceypy.spiceypy.hrmint(xvals, yvals, x)`

Evaluate a Hermite interpolating polynomial at a specified abscissa value.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/hrmint_c.html

Parameters

- **xvals** (Sequence[float]) – Abscissa values.
- **yvals** (Sequence[float]) – Ordinate and derivative values.
- **x** (int) – Point at which to interpolate the polynomial.

Return type Tuple[float, float]

Returns Interpolated function value at x and the Interpolated function's derivative at x

`spiceypy.spiceypy.hx2dp(string)`

Convert a string representing a double precision number in a base 16 scientific notation into its equivalent double precision number.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/hx2dp_c.html

Parameters **string** (str) – Hex form string to convert to double precision.

Return type Union[float, str]

Returns Double precision value to be returned, Or Error Message.

`spiceypy.spiceypy.ident()`

This routine returns the 3x3 identity matrix.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ident_c.html

Return type ndarray

Returns The 3x3 identity matrix.

`spiceypy.spiceypy.illum(target, et, abcorr, obsrvr, spoint)`

Deprecated: This routine has been superseded by the CSPICE routine `ilumin`. This routine is supported for purposes of backward compatibility only.

Find the illumination angles at a specified surface point of a target body.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/illum_c.html

Parameters

- **target** (str) – Name of target body.
- **et** (float) – Epoch in ephemeris seconds past J2000.
- **abcorr** (str) – Desired aberration correction.
- **obsrvr** (str) – Name of observing body.
- **spoint** (ndarray) – Body-fixed coordinates of a target surface point.

Return type Tuple[float, float, float]

Returns Phase angle, Solar incidence angle, and Emission angle at the surface point.

`spiceypy.spiceypy.illumf(method, target, ilusrc, et, fixref, abcorr, obsrvr, spoint)`

Compute the illumination angles—phase, incidence, and emission—at a specified point on a target body. Return logical flags indicating whether the surface point is visible from the observer's position and whether the surface point is illuminated.

The target body's surface is represented using topographic data provided by DSK files, or by a reference ellipsoid.

The illumination source is a specified ephemeris object.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/illumf_c.html

Parameters

- **method** (*str*) – Computation method.
- **target** (*str*) – Name of target body.
- **ilusrc** (*str*) – Name of illumination source.
- **et** (*float*) – Epoch in ephemeris seconds past J2000.
- **fixref** (*str*) – Body-fixed, body-centered target body frame.
- **abcorr** (*str*) – Desired aberration correction.
- **obsrvr** (*str*) – Name of observing body.
- **spoint** (*ndarray*) – Body-fixed coordinates of a target surface point.

Return type `Tuple[float, ndarray, float, float, float, bool, bool]`

Returns Target surface point epoch, Vector from observer to target surface point, Phase angle at the surface point, Source incidence angle at the surface point, Emission angle at the surface point, Visibility flag, Illumination flag

`spiceypy.spiceypy.illumg` (*method, target, ilusrc, et, fixref, abcorr, obsrvr, spoint*)

Find the illumination angles (phase, incidence, and emission) at a specified surface point of a target body.

The surface of the target body may be represented by a triaxial ellipsoid or by topographic data provided by DSK files.

The illumination source is a specified ephemeris object. param method: Computation method.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/illumg_c.html

Parameters

- **method** (*str*) – Computation method.
- **target** (*str*) – Name of target body.
- **ilusrc** (*str*) – Name of illumination source.
- **et** (*float*) – Epoch in ephemeris seconds past J2000.
- **fixref** (*str*) – Body-fixed, body-centered target body frame.
- **abcorr** (*str*) – Desired aberration correction.
- **obsrvr** (*str*) – Name of observing body.
- **spoint** (*ndarray*) – Body-fixed coordinates of a target surface point.

Return type `Tuple[float, ndarray, float, float, float]`

Returns Target surface point epoch, Vector from observer to target surface point, Phase angle at the surface point, Source incidence angle at the surface point, Emission angle at the surface point,

`spiceypy.spiceypy.illum` (*method, target, et, fixref, abcorr, obsrvr, spoint*)

Find the illumination angles (phase, solar incidence, and emission) at a specified surface point of a target body.

This routine supersedes illum.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ilumin_c.html

Parameters

- **method** (*str*) – Computation method.
- **target** (*str*) – Name of target body.
- **et** (*float*) – Epoch in ephemeris seconds past J2000.
- **fixref** (*str*) – Body-fixed, body-centered target body frame.
- **abcorr** (*str*) – Desired aberration correction.
- **obsrvr** (*str*) – Name of observing body.
- **spoint** (*ndarray*) – Body-fixed coordinates of a target surface point.

Return type `Tuple[float, ndarray, float, float, float]`

Returns Target surface point epoch, Vector from observer to target surface point, Phase angle, Solar incidence angle, and Emission angle at the surface point.

`spiceypy.spiceypy.inedpl(a, b, c, plane)`

Find the intersection of a triaxial ellipsoid and a plane.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/inedpl_c.html

Parameters

- **a** (*float*) – Length of ellipsoid semi-axis lying on the x-axis.
- **b** (*float*) – Length of ellipsoid semi-axis lying on the y-axis.
- **c** (*float*) – Length of ellipsoid semi-axis lying on the z-axis.
- **plane** (*Plane*) – Plane that intersects ellipsoid.

Return type `Tuple[Ellipse, bool]`

Returns Intersection ellipse.

`spiceypy.spiceypy.inelpl(ellips, plane)`

Find the intersection of an ellipse and a plane.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/inelpl_c.html

Parameters

- **ellips** (*Ellipse*) – A SPICE ellipse.
- **plane** (*Plane*) – A SPICE plane.

Return type `Tuple[int, ndarray, ndarray]`

Returns Number of intersection points of plane and ellipse, Point 1, Point 2.

`spiceypy.spiceypy.inrypl(vertex, direct, plane)`

Find the intersection of a ray and a plane.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/inrypl_c.html

Parameters

- **vertex** (*Iterable[float]*) – Vertex vector of ray.
- **direct** (*Iterable[float]*) – Direction vector of ray.
- **plane** (*Plane*) – A SPICE plane.

Return type Tuple[int, ndarray]

Returns Number of intersection points of ray and plane, Intersection point, if nxpts == 1.

`spiceypy.spiceypy.insrtc(item, inset)`

Insert an item into a character set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/insrtc_c.html

Parameters

- **item** (Union[str, Iterable[str]]) – Item to be inserted.
- **inset** (*SpiceCell*) – Insertion set.

Return type None

`spiceypy.spiceypy.insrtd(item, inset)`

Insert an item into a double precision set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/insrtd_c.html

Parameters

- **item** (Union[float, Iterable[float]]) – Item to be inserted.
- **inset** (*SpiceCell*) – Insertion set.

Return type None

`spiceypy.spiceypy.insrti(item, inset)`

Insert an item into an integer set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/insrti_c.html

Parameters

- **item** (Union[Iterable[int], int]) – Item to be inserted.
- **inset** (*SpiceCell*) – Insertion set.

Return type None

`spiceypy.spiceypy.inter(a, b)`

Intersect two sets of any data type to form a third set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/inter_c.html

Parameters

- **a** (*SpiceCell*) – First input set.
- **b** (*SpiceCell*) – Second input set.

Return type *SpiceCell*

Returns Intersection of a and b.

`spiceypy.spiceypy.intmax()`

Return the value of the largest (positive) number representable in a int variable.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/intmax_c.html

Return type int

Returns The largest (positive) number representable in a Int variable.

`spiceypy.spiceypy.intmin()`

Return the value of the smallest (negative) number representable in a SpiceInt variable.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/intmin_c.html

Return type `int`

Returns The smallest (negative) number representable in a Int variable.

`spiceypy.spiceypy.invert(m)`

Generate the inverse of a 3x3 matrix.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/invert_c.html

Parameters `m` (`ndarray`) – Matrix to be inverted.

Return type `ndarray`

Returns Inverted matrix $(m1)^{-1}$

`spiceypy.spiceypy.invort(m)`

Given a matrix, construct the matrix whose rows are the columns of the first divided by the length squared of the the corresponding columns of the input matrix.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/invort_c.html

Parameters `m` (`ndarray`) – A 3x3 Matrix.

Return type `ndarray`

Returns `m` after transposition and scaling of rows.

`spiceypy.spiceypy.isordv(array, n)`

Determine whether an array of `n` items contains the integers 0 through `n-1`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/isordv_c.html

Parameters

- **array** (`Iterable[int]`) – Array of integers.
- **n** (`int`) – Number of integers in array.

Return type `bool`

Returns The function returns `True` if the array contains the integers 0 through `n-1`, otherwise it returns `False`.

`spiceypy.spiceypy.isrchc(value, ndim, lenvals, array)`

Search for a given value within a character string array. Return the index of the first matching array entry, or -1 if the key value was not found.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/isrchc_c.html

Parameters

- **value** (`str`) – Key value to be found in array.
- **ndim** (`int`) – Dimension of array.
- **lenvals** (`int`) – String length.
- **array** (`Iterable[str]`) – Character string array to search.

Return type `int`

Returns The index of the first matching array element or -1 if the value is not found.

`spiceypy.spiceypy.isrchk(value, ndim, array)`

Search for a given value within a double precision array. Return the index of the first matching array entry, or -1 if the key value was not found.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/isrchk_c.html

Parameters

- **value** (float) – Key value to be found in array.
- **ndim** (int) – Dimension of array.
- **array** (Iterable[float]) – Double Precision array to search.

Return type int

Returns The index of the first matching array element or -1 if the value is not found.

`spiceypy.spiceypy.isrchi(value, ndim, array)`

Search for a given value within an integer array. Return the index of the first matching array entry, or -1 if the key value was not found.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/isrchi_c.html

Parameters

- **value** (int) – Key value to be found in array.
- **ndim** (int) – Dimension of array.
- **array** (Iterable[int]) – Integer array to search.

Return type int

Returns The index of the first matching array element or -1 if the value is not found.

`spiceypy.spiceypy.isrot(m, ntol, dtol)`

Indicate whether a 3x3 matrix is a rotation matrix.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/isrot_c.html

Parameters

- **m** (ndarray) – A matrix to be tested.
- **ntol** (float) – Tolerance for the norms of the columns of m.
- **dtol** (float) – Tolerance for the determinant of a matrix whose columns are the unitized columns of m.

Return type bool

Returns True if and only if m is a rotation matrix.

`spiceypy.spiceypy.iswhsp(string)`

Return a boolean value indicating whether a string contains only white space characters.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/iswhsp_c.html

Parameters **string** (str) – String to be tested.

Return type bool

Returns the boolean value True if the string is empty or contains only white space characters; otherwise it returns the value False.

`spiceypy.spiceypy.j1900()`

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/j1900_c.html

Return type float

Returns Julian Date of 1899 DEC 31 12:00:00

`spiceypy.spiceypy.j1950()`
http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/j1950_c.html

Return type float

Returns Julian Date of 1950 JAN 01 00:00:00

`spiceypy.spiceypy.j2000()`
http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/j2000_c.html

Return type float

Returns Julian Date of 2000 JAN 01 12:00:00

`spiceypy.spiceypy.j2100()`
http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/j2100_c.html

Return type float

Returns Julian Date of 2100 JAN 01 12:00:00

`spiceypy.spiceypy.jyear()`
http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/jyear_c.html

Return type float

Returns number of seconds in a julian year

`spiceypy.spiceypy.kclear()`
 Clear the KEEPER subsystem: unload all kernels, clear the kernel pool, and re-initialize the subsystem. Existing watches on kernel variables are retained.
http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/kclear_c.html

Return type None

`spiceypy.spiceypy.kdata(which, kind, fillen=256, typlen=256, srclen=256)`
 Return data for the nth kernel that is among a list of specified kernel types.
http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/kdata_c.html

Parameters

- **which** (int) – Index of kernel to fetch from the list of kernels.
- **kind** (str) – The kind of kernel to which fetches are limited.
- **fillen** (int) – Available space in output file string.
- **typlen** (int) – Available space in output kernel type string.
- **srclen** (int) – Available space in output source string.

Return type Tuple[str, str, str, int, bool]

Returns The name of the kernel file, The type of the kernel, Name of the source file used to load file, The handle attached to file.

`spiceypy.spiceypy.kinfo(file, typlen=256, srclen=256)`
 Return information about a loaded kernel specified by name.
http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/kinfo_c.html

Parameters

- **file** (str) – Name of a kernel to fetch information for
- **typlen** (int) – Available space in output kernel type string.
- **srcrlen** (int) – Available space in output source string.

Return type Tuple[str, str, int, bool]

Returns The type of the kernel, Name of the source file used to load file, The handle attached to file.

`spiceypy.spiceypy.kplfrm (frmcls, out_cell=None)`

Return a SPICE set containing the frame IDs of all reference frames of a given class having specifications in the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/kplfrm_c.html

Parameters

- **frmcls** (int) – Frame class.
- **out_cell** (Optional[*SpiceCell*]) – Optional output Spice Int Cell

Return type *SpiceCell*

Returns Set of ID codes of frames of the specified class.

`spiceypy.spiceypy.ktotal (kind)`

Return the current number of kernels that have been loaded via the KEEPER interface that are of a specified type.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ktotal_c.html

Parameters **kind** (str) – A list of kinds of kernels to count.

Return type int

Returns The number of kernels of type kind.

`spiceypy.spiceypy.kxtrct (keywd, terms, nterms, instring, termrlen=256, stringlen=256, sub-
strlen=256)`

Locate a keyword in a string and extract the substring from the beginning of the first word following the keyword to the beginning of the first subsequent recognized terminator of a list.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/kxtrct_c.html

Parameters

- **keywd** (str) – Word that marks the beginning of text of interest.
- **terms** (Sequence[str]) – Set of words, any of which marks the end of text.
- **nterms** (int) – Number of terms.
- **instring** (str) – String containing a sequence of words.
- **termrlen** (int) – Length of strings in string array term.
- **stringlen** (int) – Available space in argument string.
- **substrlen** (int) – Available space in output substring.

Return type Tuple[str, str, bool]

Returns String containing a sequence of words, String from end of keywd to beginning of first terms item found.

`spiceypy.spiceypy.lastnb(string)`

Return the zero based index of the last non-blank character in a character string.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lastnb_c.html

Parameters `string` (`str`) – Input character string.

Return type `int`

Returns

`spiceypy.spiceypy.latcyl(radius, lon, lat)`

Convert from latitudinal coordinates to cylindrical coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/latcyl_c.html

Parameters

- **radius** (`float`) – Distance of a point from the origin.
- **lon** (`float`) – Angle of the point from the XZ plane in radians.
- **lat** (`float`) – Angle of the point from the XY plane in radians.

Return type `Tuple[float, float, float]`

Returns (`r`, `lonc`, `z`)

`spiceypy.spiceypy.latrec(radius, longitude, latitude)`

Convert from latitudinal coordinates to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/latrec_c.html

Parameters

- **radius** (`float`) – Distance of a point from the origin.
- **longitude** (`float`) – Longitude of point in radians.
- **latitude** (`float`) – Latitude of point in radians.

Return type `ndarray`

Returns Rectangular coordinates of the point.

`spiceypy.spiceypy.latsph(radius, lon, lat)`

Convert from latitudinal coordinates to spherical coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/latsph_c.html

Parameters

- **radius** (`float`) – Distance of a point from the origin.
- **lon** (`float`) – Angle of the point from the XZ plane in radians.
- **lat** (`float`) – Angle of the point from the XY plane in radians.

Return type `Tuple[float, float, float]`

Returns (`rho` `colat`, `lons`)

`spiceypy.spiceypy.latsrf(method, target, et, fixref, lonlat)`

Map array of planetocentric longitude/latitude coordinate pairs to surface points on a specified target body.

The surface of the target body may be represented by a triaxial ellipsoid or by topographic data provided by DSK files.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/latsrf_c.html

Parameters

- **method** (*str*) – Computation method.
- **target** (*str*) – Name of target body.
- **et** (*float*) – Epoch in TDB seconds past J2000 TDB.
- **fixref** (*str*) – Body-fixed, body-centered target body frame.
- **lonlat** (*Sequence[Sequence[float]]*) – Array of longitude/latitude coordinate pairs.

Return type *ndarray***Returns** Array of surface points.`spiceypy.spiceypy.lcase (instr, lenout=256)`

Convert the characters in a string to lowercase.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lcase_c.html**Parameters**

- **instr** (*str*) – Input string.
- **lenout** (*int*) – Maximum length of output string.

Return type *str***Returns** Output string, all lowercase.`spiceypy.spiceypy.ldpool (filename)`

Load the variables contained in a NAIF ASCII kernel file into the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ldpool_c.html**Parameters** **filename** (*str*) – Name of the kernel file.**Return type** *None*`spiceypy.spiceypy.lgrind (xvals, yvals, x)`

Evaluate a Lagrange interpolating polynomial for a specified set of coordinate pairs, at a specified abscissa value. Return the value of both polynomial and derivative.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lgrind_c.html**Parameters**

- **xvals** (*Sequence[float]*) – Abscissa values.
- **yvals** (*Sequence[float]*) – Ordinate values.
- **x** (*float*) – Point at which to interpolate the polynomial.

Return type *Tuple[float, float]***Returns** Polynomial value at x, Polynomial derivative at x.`spiceypy.spiceypy.limbpt (method, target, et, fixref, abcorr, corloc, obsrvr, refvec, rolstp, ncuts, schstp, soltol, maxn)`

Find limb points on a target body. The limb is the set of points of tangency on the target of rays emanating from the observer. The caller specifies half-planes bounded by the observer-target center vector in which to search for limb points.

The surface of the target body may be represented either by a triaxial ellipsoid or by topographic data.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/limbpt_c.html

Parameters

- **method** (*str*) – Computation method.
- **target** (*str*) – Name of target body.
- **et** (*float*) – Epoch in ephemeris seconds past J2000 TDB.
- **fixref** (*str*) – Body-fixed, body-centered target body frame.
- **abcorr** (*str*) – Aberration correction.
- **corloc** (*str*) – Aberration correction locus.
- **obsrvr** (*str*) – Name of observing body.
- **refvec** (*Iterable[float]*) – Reference vector for cutting half-planes.
- **rolstp** (*float*) – Roll angular step for cutting half-planes.
- **ncuts** (*int*) – Number of cutting half-planes.
- **schstp** (*float*) – Angular step size for searching.
- **soltol** (*float*) – Solution convergence tolerance.
- **maxn** (*int*) – Maximum number of entries in output arrays.

Return type *Tuple[ndarray, ndarray, ndarray, ndarray]*

Returns Counts of limb points corresponding to cuts, Limb points, Times associated with limb points, Tangent vectors emanating from the observer

`spiceypy.spiceypy.lmpool` (*cvals*)

Load the variables contained in an internal buffer into the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lmpool_c.html

Parameters **cvals** (*Union[ndarray, Iterable[str]]*) – list of strings.

Return type *None*

`spiceypy.spiceypy.lparse` (*inlist, delim, nmax*)

Parse a list of items delimited by a single character.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lparse_c.html

Parameters

- **inlist** (*str*) – list of items delimited by *delim*.
- **delim** (*str*) – Single character used to delimit items.
- **nmax** (*int*) – Maximum number of items to return.

Return type *Iterable[str]*

Returns Items in the list, left justified.

`spiceypy.spiceypy.lparsm` (*inlist, delims, nmax, lenout=None*)

Parse a list of items separated by multiple delimiters.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lparsm_c.html

Parameters

- **inlist** (*str*) – list of items delimited by *delims*.
- **delims** (*str*) – Single characters which delimit items.

- **nmax** (int) – Maximum number of items to return.
- **lenout** (Optional[int]) – Optional Length of strings in item array.

Return type Iterable[str]

Returns Items in the list, left justified.

`spiceypy.spiceypy.lparss (inlist, delims, nmax=20, length=50)`

Parse a list of items separated by multiple delimiters, placing the resulting items into a set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lparss_c.html

Parameters

- **inlist** (str) – list of items delimited by delims.
- **delims** (str) – Single characters which delimit items.
- **nmax** (int) – Optional nmax of spice set.
- **length** (int) – Optional length of strings in spice set

Return type SpiceCell

Returns Set containing items in the list, left justified.

`spiceypy.spiceypy.lspcn (body, et, abcorr)`

Compute L_s, the planetocentric longitude of the sun, as seen from a specified body.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lspcn_c.html

Parameters

- **body** (str) – Name of central body.
- **et** (float) – Epoch in seconds past J2000 TDB.
- **abcorr** (str) – Aberration correction.

Return type float

Returns planetocentric longitude of the sun

`spiceypy.spiceypy.lstlec (string, n, lenvals, array)`

Given a character string and an ordered array of character strings, find the index of the largest array element less than or equal to the given string.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lstlec_c.html

Parameters

- **string** (str) – Upper bound value to search against.
- **n** (int) – Number elements in array.
- **lenvals** (int) – String length.
- **array** (Iterable[str]) – Array of possible lower bounds.

Return type int

Returns index of the last element of array that is lexically less than or equal to string.

`spiceypy.spiceypy.lstled (x, n, array)`

Given a number x and an array of non-decreasing floats find the index of the largest array element less than or equal to x.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lstled_c.html

Parameters

- **x** (float) – Value to search against.
- **n** (int) – Number elements in array.
- **array** (Iterable[float]) – Array of possible lower bounds

Return type int

Returns index of the last element of array that is less than or equal to x.

`spiceypy.spiceypy.lstlei(x, n, array)`

Given a number x and an array of non-decreasing ints, find the index of the largest array element less than or equal to x.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lstlei_c.html

Parameters

- **x** (int) – Value to search against.
- **n** (int) – Number elements in array.
- **array** (Iterable[int]) – Array of possible lower bounds

Return type int

Returns index of the last element of array that is less than or equal to x.

`spiceypy.spiceypy.lstltc(string, n, lenvals, array)`

Given a character string and an ordered array of character strings, find the index of the largest array element less than the given string.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lstltc_c.html

Parameters

- **string** (str) – Upper bound value to search against.
- **n** (int) – Number elements in array.
- **lenvals** (int) – String length.
- **array** (Iterable[str]) – Array of possible lower bounds

Return type int

Returns index of the last element of array that is lexically less than string.

`spiceypy.spiceypy.lstltd(x, n, array)`

Given a number x and an array of non-decreasing floats find the index of the largest array element less than x.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lstltd_c.html

Parameters

- **x** (float) – Value to search against
- **n** (int) – Number elements in array
- **array** (Iterable[float]) – Array of possible lower bounds

Return type int

Returns index of the last element of array that is less than x.

`spiceypy.spiceypy.lstlti(x, n, array)`

Given a number `x` and an array of non-decreasing `int`, find the index of the largest array element less than `x`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lstlti_c.html

Parameters

- **x** (`int`) – Value to search against
- **n** (`int`) – Number elements in array
- **array** (`Iterable[int]`) – Array of possible lower bounds

Return type `int`

Returns index of the last element of array that is less than `x`.

`spiceypy.spiceypy.ltime(etobs, obs, direct, targ)`

This routine computes the transmit (or receive) time of a signal at a specified target, given the receive (or transmit) time at a specified observer. The elapsed time between transmit and receive is also returned.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ltime_c.html

Parameters

- **etobs** (`float`) – Epoch of a signal at some observer
- **obs** (`int`) – NAIF ID of some observer
- **direct** (`str`) – Direction the signal travels (“->” or “<-”)
- **targ** (`int`) – NAIF ID of the target object

Return type `Tuple[float, float]`

Returns epoch and time

`spiceypy.spiceypy.lx4dec(string, first)`

Scan a string from a specified starting position for the end of a decimal number.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lx4dec_c.html

Parameters

- **string** (`str`) – Any character string.
- **first** (`int`) – First character to scan from in string.

Return type `Tuple[int, int]`

Returns last and nchar

`spiceypy.spiceypy.lx4num(string, first)`

Scan a string from a specified starting position for the end of a number.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lx4num_c.html

Parameters

- **string** (`str`) – Any character string.
- **first** (`int`) – First character to scan from in string.

Return type `Tuple[int, int]`

Returns last and nchar

`spiceypy.spiceypy.lx4sgn(string, first)`

Scan a string from a specified starting position for the end of a signed integer.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lx4sgn_c.html

Parameters

- **string** (`str`) – Any character string.
- **first** (`int`) – First character to scan from in string.

Return type `Tuple[int, int]`

Returns last and nchar

`spiceypy.spiceypy.lx4uns(string, first)`

Scan a string from a specified starting position for the end of an unsigned integer.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lx4uns_c.html

Parameters

- **string** (`str`) – Any character string.
- **first** (`int`) – First character to scan from in string.

Return type `Tuple[int, int]`

Returns last and nchar

`spiceypy.spiceypy.lxqstr(string, qchar, first)`

Lex (scan) a quoted string.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/lxqstr_c.html

Parameters

- **string** (`str`) – String to be scanned.
- **qchar** (`str`) – Quote delimiter character.
- **first** (`int`) – Character position at which to start scanning.

Return type `Tuple[int, int]`

Returns last and nchar

`spiceypy.spiceypy.m2eul(r, axis3, axis2, axis1)`

Factor a rotation matrix as a product of three rotations about specified coordinate axes.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/m2eul_c.html

Parameters

- **r** (`Iterable[Iterable[float]]`) – A rotation matrix to be factored
- **axis3** (`int`) – third rotation axes.
- **axis2** (`int`) – second rotation axes.
- **axis1** (`int`) – first rotation axes.

Return type `Tuple[float, float, float]`

Returns Third, second, and first Euler angles, in radians.

`spiceypy.spiceypy.m2q(r)`

Find a unit quaternion corresponding to a specified rotation matrix.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/m2q_c.html

Parameters `r` (ndarray) – A rotation matrix to be factored

Return type ndarray

Returns A unit quaternion representing the rotation matrix

`spiceypy.spiceypy.matchi(string, templ, wstr, wchr)`

Determine whether a string is matched by a template containing wild cards. The pattern comparison is case-insensitive.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/matchi_c.html

Parameters

- **string** (str) – String to be tested.
- **templ** (str) – Template (with wild cards) to test against string.
- **wstr** (str) – Wild string token.
- **wchr** (str) – Wild character token.

Return type bool

Returns The function returns True if string matches templ, else False

`spiceypy.spiceypy.matchw(string, templ, wstr, wchr)`

Determine whether a string is matched by a template containing wild cards.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/matchw_c.html

Parameters

- **string** (str) – String to be tested.
- **templ** (str) – Template (with wild cards) to test against string.
- **wstr** (str) – Wild string token.
- **wchr** (str) – Wild character token.

Return type bool

Returns The function returns True if string matches templ, else False

`spiceypy.spiceypy.mequ(m1)`

Set one double precision 3x3 matrix equal to another.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mequ_c.html

Parameters `m1` (ndarray) – input matrix.

Return type ndarray

Returns Output matrix equal to m1.

`spiceypy.spiceypy.mequg(m1, nr, nc)`

Set one double precision matrix of arbitrary size equal to another.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mequg_c.html

Parameters

- **m1** (ndarray) – Input matrix.
- **nr** (int) – Row dimension of m1.
- **nc** (int) – Column dimension of m1.

Return type ndarray

Returns Output matrix equal to `m1`

`spiceypy.spiceypy.mtxm(m1, m2)`

Multiply the transpose of a 3x3 matrix and a 3x3 matrix.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mtx_m_c.html

Parameters

- **m1** (ndarray) – 3x3 double precision matrix.
- **m2** (ndarray) – 3x3 double precision matrix.

Return type ndarray

Returns The produce `m1` transpose times `m2`.

`spiceypy.spiceypy.mtxmg(m1, m2, ncol1, nr1r2, ncol2)`

Multiply the transpose of a matrix with another matrix, both of arbitrary size.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mtxmg_c.html

Parameters

- **m1** (ndarray) – `nr1r2` X `ncol1` double precision matrix.
- **m2** (ndarray) – `nr1r2` X `ncol2` double precision matrix.
- **ncol1** (int) – Column dimension of `m1` and row dimension of `mout`.
- **nr1r2** (int) – Row dimension of `m1` and `m2`.
- **ncol2** (int) – Column dimension of `m2`.

Return type ndarray

Returns Transpose of `m1` times `m2`.

`spiceypy.spiceypy.mtxv(m1, vin)`

Multiplies the transpose of a 3x3 matrix on the left with a vector on the right.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mtxv_c.html

Parameters

- **m1** (ndarray) – 3x3 double precision matrix.
- **vin** (ndarray) – 3-dimensional double precision vector.

Return type ndarray

Returns 3-dimensional double precision vector.

`spiceypy.spiceypy.mtxvg(m1, v2, ncol1, nr1r2)`

Multiply the transpose of a matrix and a vector of arbitrary size.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mtxvg_c.html

Parameters

- **m1** (ndarray) – Left-hand matrix to be multiplied.
- **v2** (ndarray) – Right-hand vector to be multiplied.
- **ncol1** (int) – Column dimension of `m1` and length of `vout`.
- **nr1r2** (int) – Row dimension of `m1` and length of `v2`.

Return type ndarray

Returns Product vector $m1$ transpose * $v2$.

`spiceypy.spiceypy.mxm(m1, m2)`

Multiply two 3x3 matrices.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mxm_c.html

Parameters

- **m1** (Iterable[Iterable[float]]) – 3x3 double precision matrix.
- **m2** (Iterable[Iterable[float]]) – 3x3 double precision matrix.

Return type ndarray

Returns 3x3 double precision matrix.

`spiceypy.spiceypy.mxmg(m1, m2, nrow1, ncol1, ncol2)`

Multiply two double precision matrices of arbitrary size.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mxmg_c.html

Parameters

- **m1** (Iterable[Iterable[float]]) – nrow1 X ncol1 double precision matrix.
- **m2** (Iterable[Iterable[float]]) – ncol1 X ncol2 double precision matrix.
- **nrow1** (int) – Row dimension of m1
- **ncol1** (int) – Column dimension of m1 and row dimension of m2.
- **ncol2** (int) – Column dimension of m2

Return type ndarray

Returns nrow1 X ncol2 double precision matrix.

`spiceypy.spiceypy.mxmt(m1, m2)`

Multiply a 3x3 matrix and the transpose of another 3x3 matrix.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mxmt_c.html

Parameters

- **m1** (Iterable[Iterable[float]]) – 3x3 double precision matrix.
- **m2** (Iterable[Iterable[float]]) – 3x3 double precision matrix.

Return type ndarray

Returns The product $m1$ times $m2$ transpose.

`spiceypy.spiceypy.mxmtg(m1, m2, nrow1, ncol2, nrow2)`

Multiply a matrix and the transpose of a matrix, both of arbitrary size.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mxmtg_c.html

Parameters

- **m1** (ndarray) – Left-hand matrix to be multiplied.
- **m2** (ndarray) – Right-hand matrix whose transpose is to be multiplied
- **nrow1** (int) – Row dimension of m1 and row dimension of mout.
- **ncol2** (int) – Column dimension of m1 and column dimension of m2.
- **nrow2** (int) – Row dimension of m2 and column dimension of mout.

Return type ndarray

Returns Product matrix.

`spiceypy.spiceypy.mxv(m1, vin)`

Multiply a 3x3 double precision matrix with a 3-dimensional double precision vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mxv_c.html

Parameters

- **m1** (ndarray) – 3x3 double precision matrix.
- **vin** (ndarray) – 3-dimensional double precision vector.

Return type ndarray

Returns 3-dimensional double precision vector.

`spiceypy.spiceypy.mxvg(m1, v2, nrow1, nc1r2)`

Multiply a matrix and a vector of arbitrary size.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/mxvg_c.html

Parameters

- **m1** (ndarray) – Left-hand matrix to be multiplied.
- **v2** (ndarray) – Right-hand vector to be multiplied.
- **nrow1** (int) – Row dimension of m1 and length of vout.
- **nc1r2** (int) – Column dimension of m1 and length of v2.

Return type ndarray

Returns Product vector $m1*v2$

`spiceypy.spiceypy.namfrm(fname)`

Look up the frame ID code associated with a string.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/namfrm_c.html

Parameters **fname** (str) – The name of some reference frame.

Return type int

Returns The SPICE ID code of the frame.

`spiceypy.spiceypy.ncpos(string, chars, start)`

Find the first occurrence in a string of a character NOT belonging to a collection of characters, starting at a specified location searching forward.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ncpos_c.html

Parameters

- **string** (str) – Any character string.
- **chars** (str) – A collection of characters.
- **start** (int) – Position to begin looking for one not in chars.

Return type int

Returns index

`spiceypy.spiceypy.ncposr(string, chars, start)`

Find the first occurrence in a string of a character NOT belonging to a collection of characters, starting at a specified location, searching in reverse.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ncposr_c.html

Parameters

- **string** (`str`) – Any character string.
- **chars** (`str`) – A collection of characters.
- **start** (`int`) – Position to begin looking for one of chars.

Return type `int`

Returns index

`spiceypy.spiceypy.nearpt(positn, a, b, c)`

locates the point on the surface of an ellipsoid that is nearest to a specified position. It also returns the altitude of the position above the ellipsoid.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/nearpt_c.html

Parameters

- **positn** (`Iterable[float]`) – Position of a point in bodyfixed frame.
- **a** (`float`) – Length of semi-axis parallel to x-axis.
- **b** (`float`) – Length of semi-axis parallel to y-axis.
- **c** (`float`) – Length on semi-axis parallel to z-axis.

Return type `Tuple[ndarray, float]`

Returns Point on the ellipsoid closest to positn, Altitude of positn above the ellipsoid.

`spiceypy.spiceypy.no_found_check()`

Temporarily disables spiceypy default behavior which raises exceptions for false found flags for certain spice functions. All spice functions executed within the context manager will no longer check the found flag return parameter and the found flag will be included in the return for the given function. For Example `bodc2n` in spiceypy is normally called like:

```
name = spice.bodc2n(399)
```

With the possibility that an exception is thrown in the even of a invalid ID:

```
name = spice.bodc2n(-999991) # throws a SpiceyError
```

With this function however, we can use it as a context manager to do this:

```
with spice.no_found_check():
    name, found = spice.bodc2n(-999991) # found is false, no exception raised!
```

Within the context any spice functions called that normally check the found flags will pass through the check without raising an exception if they are false.

Return type `Iterator[None]`

`spiceypy.spiceypy.npedln(a, b, c, linept, linedr)`

Find nearest point on a triaxial ellipsoid to a specified line and the distance from the ellipsoid to the line.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/npedln_c.html

Parameters

- **a** (float) – Length of ellipsoid’s semi-axis in the x direction
- **b** (float) – Length of ellipsoid’s semi-axis in the y direction
- **c** (float) – Length of ellipsoid’s semi-axis in the z direction
- **linept** (Iterable[float]) – Length of ellipsoid’s semi-axis in the z direction
- **linedr** (Iterable[float]) – Direction vector of line

Return type Tuple[ndarray, float]

Returns Nearest point on ellipsoid to line, Distance of ellipsoid from line

`spiceypy.spiceypy.npelpt (point, ellips)`

Find the nearest point on an ellipse to a specified point, both in three-dimensional space, and find the distance between the ellipse and the point. http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/npelpt_c.html

Parameters

- **point** (Iterable[float]) – Point whose distance to an ellipse is to be found.
- **ellips** (*Ellipse*) – An ellipse.

Return type Tuple[ndarray, float]

Returns Nearest point on ellipsoid to line, Distance of ellipsoid from line

`spiceypy.spiceypy.nplnpt (linpt, lindir, point)`

Find the nearest point on a line to a specified point, and find the distance between the two points.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/nplnpt_c.html

Parameters

- **linpt** (Iterable[float]) – Point on a line
- **lindir** (Iterable[float]) – line’s direction vector
- **point** (Iterable[float]) – A second point.

Return type Tuple[ndarray, float]

Returns Nearest point on the line to point, Distance between point and pnear

`spiceypy.spiceypy.nvc2pl (normal, constant)`

Make a plane from a normal vector and a constant.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/nvc2pl_c.html

Parameters

- **normal** (Iterable[float]) – A normal vector defining a plane.
- **constant** (float) – A constant defining a plane.

Return type *Plane*

Returns plane

`spiceypy.spiceypy.nvp2pl (normal, point)`

Make a plane from a normal vector and a point.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/nvp2pl_c.html

Parameters

- **normal** (Iterable[float]) – A normal vector defining a plane.

- **point** (Iterable[float]) – A point defining a plane.

Return type *Plane*

Returns plane

`spiceypy.spiceypy.occult(target1, shape1, frame1, target2, shape2, frame2, abcorr, observer, et)`

Determines the occultation condition (not occulted, partially, etc.) of one target relative to another target as seen by an observer at a given time.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/occult_c.html

Parameters

- **target1** (str) – Name or ID of first target.
- **shape1** (str) – Type of shape model used for first target.
- **frame1** (str) – Body-fixed, body-centered frame for first body.
- **target2** (str) – Name or ID of second target.
- **shape2** (str) – Type of shape model used for second target.
- **frame2** (str) – Body-fixed, body-centered frame for second body.
- **abcorr** (str) – Aberration correction flag.
- **observer** (str) – Name or ID of the observer.
- **et** (float) – Time of the observation (seconds past J2000).

Return type int

Returns Occultation identification code.

`spiceypy.spiceypy.ordc(item, inset)`

The function returns the ordinal position of any given item in a character set. If the item does not appear in the set, the function returns -1.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ordc_c.html

Parameters

- **item** (str) – An item to locate within a set.
- **inset** (*SpiceCell*) – A set to search for a given item.

Return type int

Returns the ordinal position of item within the set

`spiceypy.spiceypy.ordd(item, inset)`

The function returns the ordinal position of any given item in a double precision set. If the item does not appear in the set, the function returns -1.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ordd_c.html

Parameters

- **item** (float) – An item to locate within a set.
- **inset** (*SpiceCell*) – A set to search for a given item.

Return type int

Returns the ordinal position of item within the set

`spiceypy.spiceypy.orderc (array, ndim=None)`

Determine the order of elements in an array of character strings.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/orderc_c.html

Parameters

- **array** (Sequence[str]) – Input array.
- **ndim** (Optional[int]) – Optional Length of input array

Return type ndarray

Returns Order vector for array.

`spiceypy.spiceypy.orderd (array, ndim=None)`

Determine the order of elements in a double precision array.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/orderd_c.html

Parameters

- **array** (Sequence[float]) – Input array.
- **ndim** (Optional[int]) – Optional Length of input array

Return type ndarray

Returns Order vector for array.

`spiceypy.spiceypy.orderi (array, ndim=None)`

Determine the order of elements in an integer array.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/orderi_c.html

Parameters

- **array** (Sequence[int]) – Input array.
- **ndim** (Optional[int]) – Optional Length of input array

Return type ndarray

Returns Order vector for array.

`spiceypy.spiceypy.ordi (item, inset)`

The function returns the ordinal position of any given item in an integer set. If the item does not appear in the set, the function returns -1.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ordi_c.html

Parameters

- **item** (int) – An item to locate within a set.
- **inset** (*SpiceCell*) – A set to search for a given item.

Return type int

Returns the ordinal position of item within the set

`spiceypy.spiceypy.oscelt (state, et, mu)`

Determine the set of osculating conic orbital elements that corresponds to the state (position, velocity) of a body at some epoch.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/oscelt_c.html

Parameters

- **state** (ndarray) – State of body at epoch of elements.
- **et** (float) – Epoch of elements.
- **mu** (Union[float, int]) – Gravitational parameter (GM) of primary body.

Return type ndarray

Returns Equivalent conic elements

`spiceypy.spiceypy.oscltx(state, et, mu)`

Determine the set of osculating conic orbital elements that corresponds to the state (position, velocity) of a body at some epoch. In addition to the classical elements, return the true anomaly, semi-major axis, and period, if applicable.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/oscltx_c.html

Parameters

- **state** (ndarray) – State of body at epoch of elements.
- **et** (float) – Epoch of elements.
- **mu** (int) – Gravitational parameter (GM) of primary body.

Return type ndarray

Returns Extended set of classical conic elements.

`spiceypy.spiceypy.pckcls(handle)`

Close an open PCK file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pckcls_c.html

Parameters **handle** (int) – Handle of the PCK file to be closed.

Return type None

`spiceypy.spiceypy.pckcov(pck, idcode, cover)`

Find the coverage window for a specified reference frame in a specified binary PCK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pckcov_c.html

Parameters

- **pck** (str) – Name of PCK file.
- **idcode** (int) – Class ID code of PCK reference frame.
- **cover** (*SpiceCell*) – Window giving coverage in pck for idcode.

Return type None

`spiceypy.spiceypy.pckfrm(pck, ids)`

Find the set of reference frame class ID codes of all frames in a specified binary PCK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pckfrm_c.html

Parameters

- **pck** (str) – Name of PCK file.
- **ids** (*SpiceCell*) – Set of frame class ID codes of frames in PCK file.

Return type None

`spiceypy.spiceypy.pcklof(filename)`

Load a binary PCK file for use by the readers. Return the handle of the loaded file which is used by other PCK routines to refer to the file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pcklof_c.html

Parameters `filename` (`str`) – Name of the file to be loaded.

Return type `int`

Returns Loaded file's handle.

`spiceypy.spiceypy.pckopn(name, ifname, ncomch)`

Create a new PCK file, returning the handle of the opened file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pckopn_c.html

Parameters

- **name** (`str`) – The name of the PCK file to be opened.
- **ifname** (`str`) – The internal filename for the PCK.
- **ncomch** (`int`) – The number of characters to reserve for comments.

Return type `int`

Returns The handle of the opened PCK file.

`spiceypy.spiceypy.pckuof(handle)`

Unload a binary PCK file so that it will no longer be searched by the readers.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pckuof_c.html

Parameters `handle` (`int`) – Handle of PCK file to be unloaded

Return type `None`

`spiceypy.spiceypy.pckw02(handle, classid, ffname, first, last, segid, intlen, n, polydg, cdata, btime)`

Write a type 2 segment to a PCK binary file given the file handle, frame class ID, base frame, time range covered by the segment, and the Chebyshev polynomial coefficients.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pckw02_c.html

Parameters

- **handle** (`int`) – Handle of binary PCK file open for writing.
- **classid** (`int`) – Frame class ID of body-fixed frame.
- **ffname** (`str`) – Name of base reference frame.
- **first** (`float`) – Start time of interval covered by segment.
- **last** (`float`) – End time of interval covered by segment.
- **segid** (`str`) – Segment identifier.
- **intlen** (`float`) – Length of time covered by logical record.
- **n** (`int`) – Number of logical records in segment.
- **polydg** (`int`) – Chebyshev polynomial degree.
- **cdata** (`Iterable[float]`) – Array of Chebyshev coefficients.
- **btime** (`float`) – Begin time of first logical record.

Return type `None`

`spiceypy.spiceypy.pcpool` (*name*, *cvals*)

This entry point provides toolkit programmers a method for programmatically inserting character data into the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pcpool_c.html

Parameters

- **name** (*str*) – The kernel pool name to associate with *cvals*.
- **cvals** (*Sequence[str]*) – An array of strings to insert into the kernel pool.

Return type `None`

`spiceypy.spiceypy.pdpool` (*name*, *dvals*)

This entry point provides toolkit programmers a method for programmatically inserting double precision data into the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pdpool_c.html

Parameters

- **name** (*str*) – The kernel pool name to associate with *dvals*.
- **dvals** (*Union[ndarray, Iterable[float]]*) – An array of values to insert into the kernel pool.

Return type `None`

`spiceypy.spiceypy.pgrrec` (*body*, *lon*, *lat*, *alt*, *re*, *f*)

Convert planetographic coordinates to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pgrrec_c.html

Parameters

- **body** (*str*) – Body with which coordinate system is associated.
- **lon** (*float*) – Planetographic longitude of a point (radians).
- **lat** (*float*) – Planetographic latitude of a point (radians).
- **alt** (*int*) – Altitude of a point above reference spheroid.
- **re** (*float*) – Equatorial radius of the reference spheroid.
- **f** (*float*) – Flattening coefficient.

Return type `ndarray`

Returns Rectangular coordinates of the point.

`spiceypy.spiceypy.phaseq` (*et*, *target*, *illmn*, *obsrvr*, *abcorr*)

Compute the apparent phase angle for a target, observer, illuminator set of ephemeris objects.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/phaseq_c.html

Parameters

- **et** (*float*) – Ephemeris seconds past J2000 TDB.
- **target** (*str*) – Target body name.
- **illmn** (*str*) – Illuminating body name.
- **obsrvr** (*str*) – Observer body.
- **abcorr** (*str*) – Aberration correction flag.

Return type float

Returns Value of phase angle.

`spiceypy.spiceypy.pi()`

Return the value of pi (the ratio of the circumference of a circle to its diameter).

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pi_c.html

Return type float

Returns value of pi.

`spiceypy.spiceypy.pipool(name, ival)`

This entry point provides toolkit programmers a method for programmatically inserting integer data into the kernel pool.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pipool_c.html

Parameters

- **name** (str) – The kernel pool name to associate with values.
- **ival** (ndarray) – An array of integers to insert into the pool.

Return type None

`spiceypy.spiceypy.pjelp1(elin, plane)`

Project an ellipse onto a plane, orthogonally.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pjelp1_c.html

Parameters

- **elin** (*Ellipse*) – A SPICE ellipse to be projected.
- **plane** (*Plane*) – A plane onto which elin is to be projected.

Return type *Ellipse*

Returns A SPICE ellipse resulting from the projection.

`spiceypy.spiceypy.pl2nvc(plane)`

Return a unit normal vector and constant that define a specified plane.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pl2nvc_c.html

Parameters **plane** (*Plane*) – A SPICE plane.

Return type Tuple[ndarray, float]

Returns A normal vector and constant defining the geometric plane represented by plane.

`spiceypy.spiceypy.pl2nvp(plane)`

Return a unit normal vector and point that define a specified plane.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pl2nvp_c.html

Parameters **plane** (*Plane*) – A SPICE plane.

Return type Tuple[ndarray, ndarray]

Returns A unit normal vector and point that define plane.

`spiceypy.spiceypy.pl2psv(plane)`

Return a point and two orthogonal spanning vectors that generate a specified plane.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pl2psv_c.html

Parameters **plane** (*Plane*) – A SPICE plane.

Return type Tuple[ndarray, ndarray, ndarray]

Returns A point in the input plane and two vectors spanning the input plane.

`spiceypy.spiceypy.pltar` (*vtces, plates*)

Compute the total area of a collection of triangular plates.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pltar_c.html

Parameters

- **vtces** (Sequence[Iterable[float]]) – Array of vertices.
- **plates** (Sequence[Iterable[int]]) – Array of plates.

Return type float

Returns total area of the set of plates

`spiceypy.spiceypy.pltexp` (*iverts, delta*)

Expand a triangular plate by a specified amount. The expanded plate is co-planar with, and has the same orientation as, the original. The centroids of the two plates coincide.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pltexp_c.html

Parameters

- **iverts** (Iterable[Iterable[float]]) – Vertices of the plate to be expanded.
- **delta** (float) – Fraction by which the plate is to be expanded.

Return type ndarray

Returns Vertices of the expanded plate.

`spiceypy.spiceypy.pltnp` (*point, v1, v2, v3*)

Find the nearest point on a triangular plate to a given point.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pltnp_c.html

Parameters

- **point** (Iterable[float]) – A point in 3-dimensional space.
- **v1** (Iterable[float]) – Vertices of a triangular plate.
- **v2** (Iterable[float]) – Vertices of a triangular plate.
- **v3** (Iterable[float]) – Vertices of a triangular plate.

Return type Tuple[ndarray, float]

Returns the nearest point on a triangular plate to a given point and distance

`spiceypy.spiceypy.pltnrm` (*v1, v2, v3*)

Compute an outward normal vector of a triangular plate. The vector does not necessarily have unit length.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pltnrm_c.html

Parameters

- **v1** (Iterable[float]) – Vertices of a plate.
- **v2** (Iterable[float]) – Vertices of a plate.
- **v3** (Iterable[float]) – Vertices of a plate.

Return type ndarray

Returns Plate's outward normal vector.

`spiceypy.spiceypy.pltvol (vrtces, plates)`

Compute the volume of a three-dimensional region bounded by a collection of triangular plates.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pltvol_c.html

Parameters

- **vrtces** (Sequence[Iterable[float]]) – Array of vertices.
- **plates** (Sequence[Iterable[int]]) – Array of plates.

Return type float

Returns the volume of the spatial region bounded by the plates.

`spiceypy.spiceypy.polyds (coeffs, deg, nderiv, t)`

Compute the value of a polynomial and it's first n derivatives at the value t.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/polyds_c.html

Parameters

- **coeffs** (Iterable[float]) – Coefficients of the polynomial to be evaluated.
- **deg** (int) – Degree of the polynomial to be evaluated.
- **nderiv** (int) – Number of derivatives to compute.
- **t** (int) – Point to evaluate the polynomial and derivatives

Return type ndarray

Returns Value of polynomial and derivatives.

`spiceypy.spiceypy.pos (string, substr, start)`

Find the first occurrence in a string of a substring, starting at a specified location, searching forward.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pos_c.html

Parameters

- **string** (str) – Any character string.
- **substr** (str) – Substring to locate in the character string.
- **start** (int) – Position to begin looking for substr in string.

Return type int

Returns The index of the first occurrence of substr in string at or following index start.

`spiceypy.spiceypy.posr (string, substr, start)`

Find the first occurrence in a string of a substring, starting at a specified location, searching backward.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/posr_c.html

Parameters

- **string** (str) – Any character string.
- **substr** (str) – Substring to locate in the character string.
- **start** (int) – Position to begin looking for substr in string.

Return type int

Returns The index of the last occurrence of substr in string at or preceding index start.

`spiceypy.spiceypy.prop2b(gm, pvinit, dt)`

Given a central mass and the state of massless body at time `t_0`, this routine determines the state as predicted by a two-body force model at time `t_0 + dt`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/prop2b_c.html

Parameters

- **gm** (float) – Gravity of the central mass.
- **pvinit** (ndarray) – Initial state from which to propagate a state.
- **dt** (float) – Time offset from initial state to propagate to.

Return type ndarray

Returns The propagated state.

`spiceypy.spiceypy.prstdp(string)`

Parse a string as a double precision number, encapsulating error handling.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/prstdp_c.html

Parameters **string** (str) – String representing a d.p. number.

Return type float

Returns D.p. value obtained by parsing string.

`spiceypy.spiceypy.prsint(string)`

Parse a string as an integer, encapsulating error handling.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/prsint_c.html

Parameters **string** (str) – String representing an integer.

Return type int

Returns Integer value obtained by parsing string.

`spiceypy.spiceypy.psv2pl(point, span1, span2)`

Make a CSPICE plane from a point and two spanning vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/psv2pl_c.html

Parameters

- **point** (ndarray) – A Point.
- **span1** (ndarray) – First Spanning vector.
- **span2** (ndarray) – Second Spanning vector.

Return type *Plane*

Returns A SPICE plane.

`spiceypy.spiceypy.pxform(fromstr, tostr, et)`

Return the matrix that transforms position vectors from one specified frame to another at a specified epoch.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pxform_c.html

Parameters

- **fromstr** (str) – Name of the frame to transform from.
- **tostr** (str) – Name of the frame to transform to.
- **et** (float) – Epoch of the rotation matrix.

Return type ndarray

Returns A rotation matrix.

`spiceypy.spiceypy.pxfrm2(frame_from, frame_to, etfrom, etto)`

Return the 3x3 matrix that transforms position vectors from one specified frame at a specified epoch to another specified frame at another specified epoch.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/pxfrm2_c.html

Parameters

- **frame_from** (str) – Name of the frame to transform from.
- **frame_to** (str) – Name of the frame to transform to.
- **etfrom** (float) – Evaluation time of frame_from.
- **etto** (float) – Evaluation time of frame_to.

Return type ndarray

Returns A position transformation matrix from frame_from to frame_to

`spiceypy.spiceypy.q2m(q)`

Find the rotation matrix corresponding to a specified unit quaternion.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/q2m_c.html

Parameters **q** (ndarray) – A unit quaternion.

Return type ndarray

Returns A rotation matrix corresponding to q

`spiceypy.spiceypy.qcktrc(tracelen=256)`

Return a string containing a traceback.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/qcktrc_c.html

Parameters **tracelen** (int) – Maximum length of output traceback string.

Return type str

Returns A traceback string.

`spiceypy.spiceypy.qdq2av(q, dq)`

Derive angular velocity from a unit quaternion and its derivative with respect to time.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/qdq2av_c.html

Parameters

- **q** (ndarray) – Unit SPICE quaternion.
- **dq** (Iterable[float]) – Derivative of q with respect to time

Return type ndarray

Returns Angular velocity defined by q and dq.

`spiceypy.spiceypy.qxq(q1, q2)`

Multiply two quaternions.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/qxq_c.html

Parameters

- **q1** (Union[ndarray, Iterable[float]]) – First SPICE quaternion.

- **q2** (Iterable[float]) – Second SPICE quaternion.

Return type ndarray

Returns Product of q1 and q2.

`spiceypy.spiceypy.radrec (inrange, re, dec)`

Convert from range, right ascension, and declination to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/radrec_c.html

Parameters

- **inrange** (float) – Distance of a point from the origin.
- **re** (float) – Right ascension of point in radians.
- **dec** (float) – Declination of point in radians.

Return type ndarray

Returns Rectangular coordinates of the point.

`spiceypy.spiceypy.rav2xf (rot, av)`

This routine determines a state transformation matrix from a rotation matrix and the angular velocity of the rotation.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/rav2xf_c.html

Parameters

- **rot** (Iterable[Iterable[float]]) – Rotation matrix.
- **av** (Iterable[float]) – Angular velocity vector.

Return type ndarray

Returns State transformation associated with rot and av.

`spiceypy.spiceypy.raxisa (matrix)`

Compute the axis of the rotation given by an input matrix and the angle of the rotation about that axis.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/raxisa_c.html

Parameters **matrix** (ndarray) – Rotation matrix.

Return type Tuple[ndarray, float]

Returns Axis of the rotation, Angle through which the rotation is performed

`spiceypy.spiceypy.rdtype (file, lenout=256)`

Read the next line of text from a text file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/rdtext_c.html

Parameters

- **file** (str) – Name of text file.
- **lenout** (int) – Available room in output line.

Return type Tuple[str, bool]

Returns Next line from the text file, End-of-file indicator

`spiceypy.spiceypy.reccyl (rectan)`

Convert from rectangular to cylindrical coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/reccyl_c.html

Parameters **rectan** (Iterable[float]) – Rectangular coordinates of a point.

Return type Tuple[float, float, float]

Returns Distance from z axis, Angle (radians) from xZ plane, Height above xY plane.

`spiceypy.spiceypy.recgeo (rectan, re, f)`

Convert from rectangular coordinates to geodetic coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/recgeo_c.html

Parameters

- **rectan** (Iterable[float]) – Rectangular coordinates of a point.
- **re** (float) – Equatorial radius of the reference spheroid.
- **f** (float) – Flattening coefficient.

Return type Tuple[float, float, float]

Returns Geodetic longitude (radians), Geodetic latitude (radians), Altitude above reference spheroid

`spiceypy.spiceypy.reclat (rectan)`

Convert from rectangular coordinates to latitudinal coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/reclat_c.html

Parameters **rectan** (Union[ndarray, Iterable[float]]) – Rectangular coordinates of a point.

Return type Tuple[float, float, float]

Returns Distance from the origin, Longitude in radians, Latitude in radians

`spiceypy.spiceypy.recpgr (body, rectan, re, f)`

Convert rectangular coordinates to planetographic coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/recpgr_c.html

Parameters

- **body** (str) – Body with which coordinate system is associated.
- **rectan** (Union[ndarray, Iterable[float]]) – Rectangular coordinates of a point.
- **re** (float) – Equatorial radius of the reference spheroid.
- **f** (float) – Flattening coefficient.

Return type Tuple[float, float, float]

Returns Planetographic longitude (radians), Planetographic latitude (radians), Altitude above reference spheroid

`spiceypy.spiceypy.recrad (rectan)`

Convert rectangular coordinates to range, right ascension, and declination.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/recrad_c.html

Parameters **rectan** (Union[ndarray, Iterable[float]]) – Rectangular coordinates of a point.

Return type Tuple[float, float, float]

Returns Distance of the point from the origin, Right ascension in radians, Declination in radians

`spiceypy.spiceypy.rectsph (rectan)`

Convert from rectangular coordinates to spherical coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/recrad_c.html

Parameters `rectan` (ndarray) – Rectangular coordinates of a point.

Return type Tuple[float, float, float]

Returns Distance from the origin, Angle from the positive Z-axis, Longitude in radians.

`spiceypy.spiceypy.removec (item, inset)`

Remove an item from a character set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/removc_c.html

Parameters

- `item` (str) – Item to be removed.
- `inset` (*SpiceCell*) – Set to be updated.

Return type None

`spiceypy.spiceypy.removevd (item, inset)`

Remove an item from a double precision set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/removd_c.html

Parameters

- `item` (float) – Item to be removed.
- `inset` (*SpiceCell*) – Set to be updated.

Return type None

`spiceypy.spiceypy.removevi (item, inset)`

Remove an item from an integer set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/removi_c.html

Parameters

- `item` (int) – Item to be removed.
- `inset` (*SpiceCell*) – Set to be updated.

Return type None

`spiceypy.spiceypy.reordc (iorder, ndim, lenvals, array)`

Re-order the elements of an array of character strings according to a given order vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/reordc_c.html

Parameters

- `iorder` (Iterable[int]) – Order vector to be used to re-order array.
- `ndim` (int) – Dimension of array.
- `lenvals` (int) – String length.
- `array` (Iterable[str]) – Array to be re-ordered.

Return type Iterable[str]

Returns Re-ordered Array.

`spiceypy.spiceypy.reordd(iorder, ndim, array)`

Re-order the elements of a double precision array according to a given order vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/reordd_c.html

Parameters

- **iorder** (Iterable[int]) – Order vector to be used to re-order array.
- **ndim** (int) – Dimension of array.
- **array** (Iterable[float]) – Array to be re-ordered.

Return type ndarray

Returns Re-ordered Array.

`spiceypy.spiceypy.reordi(iorder, ndim, array)`

Re-order the elements of an integer array according to a given order vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/reordi_c.html

Parameters

- **iorder** (Iterable[int]) – Order vector to be used to re-order array.
- **ndim** (int) – Dimension of array.
- **array** (Iterable[int]) – Array to be re-ordered.

Return type ndarray

Returns Re-ordered Array.

`spiceypy.spiceypy.reordl(iorder, ndim, array)`

Re-order the elements of a logical (Boolean) array according to a given order vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/reordl_c.html

Parameters

- **iorder** (Iterable[int]) – Order vector to be used to re-order array.
- **ndim** (int) – Dimension of array.
- **array** (Iterable[bool]) – Array to be re-ordered.

Return type ndarray

Returns Re-ordered Array.

`spiceypy.spiceypy.repmc(instr, marker, value, lenout=None)`

Replace a marker with a character string.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/repmc_c.html

Parameters

- **instr** (str) – Input string.
- **marker** (str) – Marker to be replaced.
- **value** (str) – Replacement value.
- **lenout** (Optional[int]) – Optional available space in output string

Return type str

Returns Output string.

`spiceypy.spiceypy.repmct(instr, marker, value, repcase, lenout=None)`

Replace a marker with the text representation of a cardinal number.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/repmc_c.html

Parameters

- **instr** (`str`) – Input string.
- **marker** (`str`) – Marker to be replaced.
- **value** (`int`) – Replacement value.
- **repcase** (`str`) – Case of replacement text.
- **lenout** (`Optional[int]`) – Optional available space in output string

Return type `str`

Returns Output string.

`spiceypy.spiceypy.repmd(instr, marker, value, sigdig)`

Replace a marker with a double precision number.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/repmd_c.html

Parameters

- **instr** (`str`) – Input string.
- **marker** (`str`) – Marker to be replaced.
- **value** (`float`) – Replacement value.
- **sigdig** (`int`) – Significant digits in replacement text.

Return type `str`

Returns Output string.

`spiceypy.spiceypy.repmf(instr, marker, value, sigdig, informat, lenout=None)`

Replace a marker in a string with a formatted double precision value.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/repmf_c.html

Parameters

- **instr** (`str`) – Input string.
- **marker** (`str`) – Marker to be replaced.
- **value** (`float`) – Replacement value.
- **sigdig** (`int`) – Significant digits in replacement text.
- **informat** (`str`) – Format ‘E’ or ‘F’.
- **lenout** (`Optional[int]`) – Optional available space in output string.

Return type `str`

Returns Output string.

`spiceypy.spiceypy.repmi(instr, marker, value, lenout=None)`

Replace a marker with an integer.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/repmi_c.html

Parameters

- **instr** (*str*) – Input string.
- **marker** (*str*) – Marker to be replaced.
- **value** (*int*) – Replacement value.
- **lenout** (*Optional[int]*) – Optional available space in output string.

Return type *str*

Returns Output string.

`spiceypy.spiceypy.repmot (instr, marker, value, repcase, lenout=None)`

Replace a marker with the text representation of an ordinal number.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/repmot_c.html

Parameters

- **instr** (*str*) – Input string.
- **marker** (*str*) – Marker to be replaced.
- **value** (*int*) – Replacement value.
- **repcase** (*str*) – Case of replacement text.
- **lenout** (*Optional[int]*) – Optional available space in output string.

Return type *str*

Returns Output string.

`spiceypy.spiceypy.reset ()`

Reset the SPICE error status to a value of “no error.” As a result, the status routine, failed, will return a value of False

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/reset_c.html

Return type *None*

`spiceypy.spiceypy.return_c ()`

True if SPICE routines should return immediately upon entry.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/return_c.html

Return type *bool*

Returns True if SPICE routines should return immediately upon entry.

`spiceypy.spiceypy.rotate (angle, iaxis)`

Calculate the 3x3 rotation matrix generated by a rotation of a specified angle about a specified axis. This rotation is thought of as rotating the coordinate system.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/rotate_c.html

Parameters

- **angle** (*float*) – Angle of rotation (radians).
- **iaxis** (*int*) – Axis of rotation X=1, Y=2, Z=3.

Return type *ndarray*

Returns Resulting rotation matrix

`spiceypy.spiceypy.rotmat(m1, angle, iaxis)`

Rotmat applies a rotation of angle radians about axis iaxis to a matrix. This rotation is thought of as rotating the coordinate system.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/rotmat_c.html

Parameters

- **m1** (ndarray) – Matrix to be rotated.
- **angle** (float) – Angle of rotation (radians).
- **iaxis** (int) – Axis of rotation X=1, Y=2, Z=3.

Return type ndarray

Returns Resulting rotated matrix.

`spiceypy.spiceypy.rotvec(v1, angle, iaxis)`

Transform a vector to a new coordinate system rotated by angle radians about axis iaxis. This transformation rotates v1 by angle radians about the specified axis.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/rotvec_c.html

Parameters

- **v1** (Iterable[float]) – Vector whose coordinate system is to be rotated.
- **angle** (float) – Angle of rotation (radians).
- **iaxis** (int) – Axis of rotation X=1, Y=2, Z=3.

Return type ndarray

Returns the vector expressed in the new coordinate system.

`spiceypy.spiceypy.rpd()`

Return the number of radians per degree.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/rpd_c.html

Return type float

Returns The number of radians per degree, pi/180.

`spiceypy.spiceypy.rquad(a, b, c)`

Find the roots of a quadratic equation.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/rquad_c.html

Parameters

- **a** (float) – Coefficient of quadratic term.
- **b** (float) – Coefficient of linear term.
- **c** (float) – Constant.

Return type Tuple[ndarray, ndarray]

Returns Root built from positive and negative discriminant term.

`spiceypy.spiceypy.saelgv(vec1, vec2)`

Find semi-axis vectors of an ellipse generated by two arbitrary three-dimensional vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/saelgv_c.html

Parameters

- **vec1** (Iterable[float]) – First vector used to generate an ellipse.
- **vec2** (Iterable[float]) – Second vector used to generate an ellipse.

Return type Tuple[ndarray, ndarray]

Returns Semi-major axis of ellipse, Semi-minor axis of ellipse.

`spiceypy.spiceypy.scard(incard, cell)`

Set the cardinality of a SPICE cell of any data type.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/scard_c.html

Parameters

- **incard** (int) – Cardinality of (number of elements in) the cell.
- **cell** (*SpiceCell*) – The cell.

Return type *SpiceCell*

Returns The updated Cell.

`spiceypy.spiceypy.scdect(sc, sclkdp, lenout=256, mxpart=None)`

Convert double precision encoding of spacecraft clock time into a character representation.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/scdect_c.html

Parameters

- **sc** (int) – NAIF spacecraft identification code.
- **sclkdp** (float) – Encoded representation of a spacecraft clock count.
- **lenout** (int) – Maximum allowed length of output SCLK string.
- **mxpart** (Optional[int]) – Maximum number of spacecraft clock partitions.

Return type str

Returns Character representation of a clock count.

`spiceypy.spiceypy.sce2c(sc, et)`

Convert ephemeris seconds past J2000 (ET) to continuous encoded spacecraft clock “ticks”. Non-integral tick values may be returned.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sce2c_c.html

Parameters

- **sc** (int) – NAIF spacecraft ID code.
- **et** (float) – Ephemeris time, seconds past J2000.

Return type float

Returns SCLK, encoded as ticks since spacecraft clock start. sclkdp need not be integral.

`spiceypy.spiceypy.sce2s(sc, et, lenout=256)`

Convert an epoch specified as ephemeris seconds past J2000 (ET) to a character string representation of a spacecraft clock value (SCLK).

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sce2s_c.html

Parameters

- **sc** (int) – NAIF spacecraft clock ID code.
- **et** (float) – Ephemeris time, specified as seconds past J2000.

- **lenout** (int) – Maximum length of output string.

Return type str

Returns An SCLK string.

`spiceypy.spiceypy.sce2t(sc, et)`

Convert ephemeris seconds past J2000 (ET) to integral encoded spacecraft clock (“ticks”). For conversion to fractional ticks, (required for C-kernel production), see the routine `sce2c()`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sce2t_c.html

Parameters

- **sc** (int) – NAIF spacecraft ID code.
- **et** (float) – Ephemeris time, seconds past J2000.

Return type float

Returns SCLK, encoded as ticks since spacecraft clock start.

`spiceypy.spiceypy.scncd(sc, sclch, mxpart=None)`

Encode character representation of spacecraft clock time into a double precision number.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/scncd_c.html

Parameters

- **sc** (int) – NAIF spacecraft identification code.
- **sclch** (Union[str, Iterable[str]]) – Character representation of a spacecraft clock.
- **mxpart** (Optional[int]) – Maximum number of spacecraft clock partitions.

Return type Union[float, ndarray]

Returns Encoded representation of the clock count.

`spiceypy.spiceypy.scfmt(sc, ticks, lenout=256)`

Convert encoded spacecraft clock ticks to character clock format.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/scfmt_c.html

Parameters

- **sc** (int) – NAIF spacecraft identification code.
- **ticks** (float) – Encoded representation of a spacecraft clock count.
- **lenout** (int) – Maximum allowed length of output string.

Return type str

Returns Character representation of a clock count.

`spiceypy.spiceypy.scpart(sc)`

Get spacecraft clock partition information from a spacecraft clock kernel file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/scpart_c.html

Parameters **sc** (int) – NAIF spacecraft identification code.

Return type Tuple[ndarray, ndarray]

Returns The number of spacecraft clock partitions, Array of partition start times, Array of partition stop times.

`spiceypy.spiceypy.scs2e(sc, sclkch)`

Convert a spacecraft clock string to ephemeris seconds past J2000 (ET).

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/scs2e_c.html

Parameters

- **sc** (`int`) – NAIF integer code for a spacecraft.
- **sclkch** (`str`) – An SCLK string.

Return type `float`

Returns Ephemeris time, seconds past J2000.

`spiceypy.spiceypy.sct2e(sc, sclkdp)`

Convert encoded spacecraft clock (“ticks”) to ephemeris seconds past J2000 (ET).

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sct2e_c.html

Parameters

- **sc** (`int`) – NAIF spacecraft ID code.
- **sclkd** (`Union[float, Iterable[float]]`) – SCLK, encoded as ticks since spacecraft clock start.

Return type `Union[float, ndarray]`

Returns Ephemeris time, seconds past J2000.

`spiceypy.spiceypy.sctiks(sc, clkstr)`

Convert a spacecraft clock format string to number of “ticks”.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sctiks_c.html

Parameters

- **sc** (`int`) – NAIF spacecraft identification code.
- **clkstr** (`str`) – Character representation of a spacecraft clock.

Return type `float`

Returns Number of ticks represented by the clock string.

`spiceypy.spiceypy.sdifff(a, b)`

Take the symmetric difference of two sets of any data type to form a third set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sdifff_c.html

Parameters

- **a** (`SpiceCell`) – First input set.
- **b** (`SpiceCell`) – Second input set.

Return type `SpiceCell`

Returns Symmetric difference of a and b.

`spiceypy.spiceypy.set_c(a, op, b)`

Given a relational operator, compare two sets of any data type.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/set_c.html

Parameters

- **a** (`SpiceCell`) – First set.

- **op** (*str*) – Comparison operator.
- **b** (*SpiceCell*) – Second set.

Return type `bool`

Returns The function returns the result of the comparison.

`spiceypy.spiceypy.setmsg` (*message*)

Set the value of the current long error message.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/setmsg_c.html

Parameters **message** (*str*) – A long error message.

Return type `None`

`spiceypy.spiceypy.shellc` (*ndim, lenvals, array*)

Sort an array of character strings according to the ASCII collating sequence using the Shell Sort algorithm.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/shellc_c.html

Parameters

- **ndim** (*int*) – Dimension of the array.
- **lenvals** (*int*) – String length.
- **array** (*Iterable[str]*) – The array to be sorted.

Return type `Iterable[str]`

Returns The sorted array.

`spiceypy.spiceypy.shelld` (*ndim, array*)

Sort a double precision array using the Shell Sort algorithm.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/shelld_c.html

Parameters

- **ndim** (*int*) – Dimension of the array.
- **array** (*Iterable[float]*) – The array to be sorted.

Return type `ndarray`

Returns The sorted array.

`spiceypy.spiceypy.shelli` (*ndim, array*)

Sort an integer array using the Shell Sort algorithm.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/shelli_c.html

Parameters

- **ndim** (*int*) – Dimension of the array.
- **array** (*Iterable[int]*) – The array to be sorted.

Return type `ndarray`

Returns The sorted array.

`spiceypy.spiceypy.sigerr` (*message*)

Inform the CSPICE error processing mechanism that an error has occurred, and specify the type of error.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sigerr_c.html

Parameters **message** (*str*) – A short error message.

Return type None

`spiceypy.spiceypy.sincpt (method, target, et, fixref, abcorr, obsrvr, dref, dvec)`

Given an observer and a direction vector defining a ray, compute the surface intercept of the ray on a target body at a specified epoch, optionally corrected for light time and stellar aberration.

This routine supersedes `srfxpt()`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sincpt_c.html

Parameters

- **method** (str) – Computation method.
- **target** (str) – Name of target body.
- **et** (float) – Epoch in ephemeris seconds past J2000 TDB.
- **fixref** (str) – Body-fixed, body-centered target body frame.
- **abcorr** (str) – Aberration correction.
- **obsrvr** (str) – Name of observing body.
- **dref** (str) – Reference frame of ray’s direction vector.
- **dvec** (ndarray) – Ray’s direction vector.

Return type Tuple[ndarray, float, ndarray, bool]

Returns Surface intercept point on the target body, Intercept epoch, Vector from observer to intercept point.

`spiceypy.spiceypy.size (cell)`

Return the size (maximum cardinality) of a SPICE cell of any data type.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/size_c.html

Parameters **cell** (*SpiceCell*) – Input cell.

Return type int

Returns The size of the input cell.

`spiceypy.spiceypy.spd()`

Return the number of seconds in a day.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spd_c.html

Return type float

Returns The number of seconds in a day.

`spiceypy.spiceypy.sphcyl (radius, colat, slon)`

This routine converts from spherical coordinates to cylindrical coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sphcyl_c.html

Parameters

- **radius** (float) – Distance of point from origin.
- **colat** (float) – Polar angle (co-latitude in radians) of point.
- **slon** (float) – Azimuthal angle (longitude) of point (radians).

Return type Tuple[float, float, float]

Returns Distance of point from z axis, angle (radians) of point from XZ plane, Height of point above XY plane.

`spiceypy.spiceypy.sphlat(r, colat, lons)`

Convert from spherical coordinates to latitudinal coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sphlat_c.html

Parameters

- **r** (float) – Distance of the point from the origin.
- **colat** (float) – Angle of the point from positive z axis (radians).
- **lons** (float) – Angle of the point from the XZ plane (radians).

Return type Tuple[float, float, float]

Returns Distance of a point from the origin, Angle of the point from the XZ plane in radians, Angle of the point from the XY plane in radians.

`spiceypy.spiceypy.sphrec(r, colat, lon)`

Convert from spherical coordinates to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sphrec_c.html

Parameters

- **r** (float) – Distance of a point from the origin.
- **colat** (float) – Angle of the point from the positive Z-axis.
- **lon** (float) – Angle of the point from the XZ plane in radians.

Return type ndarray

Returns Rectangular coordinates of the point.

`spiceypy.spiceypy.spice_error_check(f)`

Decorator for spiceypy hooking into spice error system. If an error is detected, an output similar to outmsg

Returns

`spiceypy.spiceypy.spice_found_exception_thrower(f)`

Decorator for wrapping functions that use status codes

Return type Callable

`spiceypy.spiceypy.spk14a(handle, ncsets, coeffs, epochs)`

Add data to a type 14 SPK segment associated with handle. See also `spk14b()` and `spk14e()`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spk14a_c.html

Parameters

- **handle** (int) – The handle of an SPK file open for writing.
- **ncsets** (int) – The number of coefficient sets and epochs.
- **coeffs** (Iterable[float]) – The collection of coefficient sets.
- **epochs** (Iterable[float]) – The epochs associated with the coefficient sets.

Return type None

`spiceypy.spiceypy.spk14b(handle, segid, body, center, framename, first, last, chbdeg)`

Begin a type 14 SPK segment in the SPK file associated with handle. See also `spk14a()` and `spk14e()`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spk14b_c.html

Parameters

- **handle** (int) – The handle of an SPK file open for writing.
- **segid** (str) – The string to use for segment identifier.
- **body** (int) – The NAIF ID code for the body of the segment.
- **center** (int) – The center of motion for body.
- **framename** (str) – The reference frame for this segment.
- **first** (float) – The first epoch for which the segment is valid.
- **last** (float) – The last epoch for which the segment is valid.
- **chbdeg** (int) – The degree of the Chebyshev Polynomial used.

Return type None

`spiceypy.spiceypy.spk14e(handle)`

End the type 14 SPK segment currently being written to the SPK file associated with handle. See also `spk14a()` and `spk14b()`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spk14e_c.html

Parameters **handle** (int) – The handle of an SPK file open for writing.

Return type None

`spiceypy.spiceypy.spkacs(targ, et, ref, abcorr, obs)`

Return the state (position and velocity) of a target body relative to an observer, optionally corrected for light time and stellar aberration, expressed relative to an inertial reference frame.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkacs_c.html

Parameters

- **targ** (int) – Target body.
- **et** (float) – Observer epoch.
- **ref** (str) – Inertial reference frame of output state.
- **abcorr** (str) – Aberration correction flag.
- **obs** (int) – Observer.

Return type Tuple[ndarray, float, float]

Returns State of target, One way light time between observer and target, Derivative of light time with respect to time.

`spiceypy.spiceypy.spkapo(targ, et, ref, sobs, abcorr)`

Return the position of a target body relative to an observer, optionally corrected for light time and stellar aberration.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkapo_c.html

Parameters

- **targ** (int) – Target body.
- **et** (float) – Observer epoch.
- **ref** (str) – Inertial reference frame of observer's state.
- **sobs** (ndarray) – State of observer wrt. solar system barycenter.

- **abcorr** (str) – Aberration correction flag.

Return type Tuple[ndarray, float]

Returns Position of target, One way light time between observer and target.

`spiceypy.spiceypy.spkapp(targ, et, ref, sobs, abcorr)`

Deprecated: This routine has been superseded by `spkaps()`. This routine is supported for purposes of backward compatibility only.

Return the state (position and velocity) of a target body relative to an observer, optionally corrected for light time and stellar aberration.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkapp_c.html

Parameters

- **targ** (int) – Target body.
- **et** (float) – Observer epoch.
- **ref** (str) – Inertial reference frame of observer’s state.
- **sobs** (ndarray) – State of observer wrt. solar system barycenter.
- **abcorr** (str) – Aberration correction flag.

Return type Tuple[ndarray, float]

Returns State of target, One way light time between observer and target.

`spiceypy.spiceypy.spkaps(targ, et, ref, abcorr, stobs, accobs)`

Given the state and acceleration of an observer relative to the solar system barycenter, return the state (position and velocity) of a target body relative to the observer, optionally corrected for light time and stellar aberration. All input and output vectors are expressed relative to an inertial reference frame.

This routine supersedes `spkapp()`.

SPICE users normally should call the high-level API routines `spkezr()` or `spkez()` rather than this routine. http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkaps_c.html

Parameters

- **targ** (int) – Target body.
- **et** (float) – Observer epoch.
- **ref** (str) – Inertial reference frame of output state.
- **abcorr** (str) – Aberration correction flag.
- **stobs** (ndarray) – State of the observer relative to the SSB.
- **accobs** (Iterable[float]) – Acceleration of the observer relative to the SSB.

Return type Tuple[ndarray, float, float]

Returns State of target, One way light time between observer and target, Derivative of light time with respect to time.

`spiceypy.spiceypy.spkcls(handle)`

Close an open SPK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkcls_c.html

Parameters **handle** (int) – Handle of the SPK file to be closed.

Return type None

`spiceypy.spiceypy.spkcov` (*spk, idcode, cover=None*)

Find the coverage window for a specified ephemeris object in a specified SPK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkcov_c.html

Parameters

- **spk** (*str*) – Name of SPK file.
- **idcode** (*int*) – ID code of ephemeris object.
- **cover** (Optional[*SpiceCell*]) – Optional SPICE Window giving coverage in “spk” for “idcode”.

Return type *SpiceCell*

`spiceypy.spiceypy.spkcpo` (*target, et, outref, refloc, abcorr, obspos, obsctr, obsref*)

Return the state of a specified target relative to an “observer,” where the observer has constant position in a specified reference frame. The observer’s position is provided by the calling program rather than by loaded SPK files.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkcpo_c.html

Parameters

- **target** (*str*) – Name of target ephemeris object.
- **et** (*float*) – Observation epoch.
- **outref** (*str*) – Reference frame of output state.
- **refloc** (*str*) – Output reference frame evaluation locus.
- **abcorr** (*str*) – Aberration correction.
- **obspos** (Iterable[*float*]) – Observer position relative to center of motion.
- **obsctr** (*str*) – Center of motion of observer.
- **obsref** (*str*) – Frame of observer position.

Return type *Tuple[ndarray, float]*

Returns State of target with respect to observer, One way light time between target and observer.

`spiceypy.spiceypy.spkcpt` (*trgpos, trgctr, trgref, et, outref, refloc, abcorr, obsrvr*)

Return the state, relative to a specified observer, of a target having constant position in a specified reference frame. The target’s position is provided by the calling program rather than by loaded SPK files.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkcpt_c.html

Parameters

- **trgpos** (Iterable[*float*]) – Target position relative to center of motion.
- **trgctr** (*str*) – Center of motion of target.
- **trgref** (*str*) – Observation epoch.
- **et** (*float*) – Observation epoch.
- **outref** (*str*) – Reference frame of output state.
- **refloc** (*str*) – Output reference frame evaluation locus.
- **abcorr** (*str*) – Aberration correction.
- **obsrvr** (*str*) – Name of observing ephemeris object.

Return type Tuple[ndarray, float]

Returns State of target with respect to observer, One way light time between target and observer.

`spiceypy.spiceypy.spkcv` (*target, et, outref, refloc, abcorr, obssta, obsepc, obsctr, obsref*)

Return the state of a specified target relative to an “observer,” where the observer has constant velocity in a specified reference frame. The observer’s state is provided by the calling program rather than by loaded SPK files.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkcv_c.html

Parameters

- **target** (str) – Name of target ephemeris object.
- **et** (float) – Observation epoch.
- **outref** (str) – Reference frame of output state.
- **refloc** (str) – Output reference frame evaluation locus.
- **abcorr** (str) – Aberration correction.
- **obssta** (Iterable[float]) – Observer state relative to center of motion.
- **obsepc** (float) – Epoch of observer state.
- **obsctr** (str) – Center of motion of observer.
- **obsref** (str) – Frame of observer state.

Return type Tuple[ndarray, float]

Returns State of target with respect to observer, One way light time between target and observer.

`spiceypy.spiceypy.spkcv` (*trgsta, trgepc, trgctr, trgref, et, outref, refloc, abcorr, obsrvr*)

Return the state, relative to a specified observer, of a target having constant velocity in a specified reference frame. The target’s state is provided by the calling program rather than by loaded SPK files.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkcv_c.html

Parameters

- **trgsta** (Iterable[float]) – Target state relative to center of motion.
- **trgepc** (float) – Epoch of target state.
- **trgctr** (str) – Center of motion of target.
- **trgref** (str) – Frame of target state.
- **et** (float) – Observation epoch.
- **outref** (str) – Reference frame of output state.
- **refloc** (str) – Output reference frame evaluation locus.
- **abcorr** (str) – Aberration correction.
- **obsrvr** (str) – Name of observing ephemeris object.

Return type Tuple[ndarray, float]

Returns State of target with respect to observer, One way light time between target and observer.

`spiceypy.spiceypy.spkez` (*targ, et, ref, abcorr, obs*)

Return the state (position and velocity) of a target body relative to an observing body, optionally corrected for light time (planetary aberration) and stellar aberration.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkez_c.html

Parameters

- **targ** (int) – Target body.
- **et** (float) – Observer epoch.
- **ref** (str) – Reference frame of output state vector.
- **abcorr** (str) – Aberration correction flag.
- **obs** (int) – Observing body.

Return type Tuple[ndarray, float]

Returns State of target, One way light time between observer and target.

`spiceypy.spiceypy.spkezp(targ, et, ref, abcorr, obs)`

Return the position of a target body relative to an observing body, optionally corrected for light time (planetary aberration) and stellar aberration.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkezp_c.html

Parameters

- **targ** (int) – Target body NAIF ID code.
- **et** (float) – Observer epoch.
- **ref** (str) – Reference frame of output position vector.
- **abcorr** (str) – Aberration correction flag.
- **obs** (int) – Observing body NAIF ID code.

Return type Tuple[ndarray, float]

Returns Position of target, One way light time between observer and target.

`spiceypy.spiceypy.spkezr(targ, et, ref, abcorr, obs)`

Return the state (position and velocity) of a target body relative to an observing body, optionally corrected for light time (planetary aberration) and stellar aberration.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkezr_c.html

Parameters

- **targ** (str) – Target body name.
- **et** (Union[ndarray, float]) – Observer epoch.
- **ref** (str) – Reference frame of output state vector.
- **abcorr** (str) – Aberration correction flag.
- **obs** (str) – Observing body name.

Return type Union[Tuple[ndarray, float], Tuple[Iterable[ndarray], Iterable[float]]]

Returns State of target, One way light time between observer and target.

`spiceypy.spiceypy.spkgeo(targ, et, ref, obs)`

Compute the geometric state (position and velocity) of a target body relative to an observing body.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkgeo_c.html

Parameters

- **targ** (int) – Target body.
- **et** (float) – Target epoch.
- **ref** (str) – Target reference frame.
- **obs** (int) – Observing body.

Return type Tuple[ndarray, float]

Returns State of target, Light time.

`spiceypy.spiceypy.spkggps (targ, et, ref, obs)`

Compute the geometric position of a target body relative to an observing body.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkggps_c.html

Parameters

- **targ** (int) – Target body.
- **et** (float) – Target epoch.
- **ref** (str) – Target reference frame.
- **obs** (int) – Observing body.

Return type Tuple[ndarray, float]

Returns Position of target, Light time.

`spiceypy.spiceypy.spklef (filename)`

Load an ephemeris file for use by the readers. Return that file's handle, to be used by other SPK routines to refer to the file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spklef_c.html

Parameters **filename** (str) – Name of the file to be loaded.

Return type int

Returns Loaded file's handle.

`spiceypy.spiceypy.spkltc (targ, et, ref, abcorr, stobs)`

Return the state (position and velocity) of a target body relative to an observer, optionally corrected for light time, expressed relative to an inertial reference frame.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkltc_c.html

Parameters

- **targ** (int) – Target body.
- **et** (float) – Observer epoch.
- **ref** (str) – Inertial reference frame of output state.
- **abcorr** (str) – Aberration correction flag.
- **stobs** (ndarray) – State of the observer relative to the SSB.

Return type Tuple[ndarray, float, float]

Returns One way light time between observer and target, Derivative of light time with respect to time

`spiceypy.spiceypy.spkobj` (*spk*, *out_cell=None*)

Find the set of ID codes of all objects in a specified SPK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkobj_c.html

Parameters

- **spk** (*str*) – Name of SPK file.
- **out_cell** (Optional[*SpiceCell*]) – Optional Spice Int Cell.

Return type *SpiceCell*

`spiceypy.spiceypy.spkopa` (*filename*)

Open an existing SPK file for subsequent write.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkopa_c.html

Parameters **filename** (*str*) – The name of an existing SPK file.

Return type *int*

Returns A handle attached to the SPK file opened to append.

`spiceypy.spiceypy.spkopn` (*filename*, *ifname*, *ncomch*)

Create a new SPK file, returning the handle of the opened file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkopn_c.html

Parameters

- **filename** (*str*) – The name of the new SPK file to be created.
- **ifname** (*str*) – The internal filename for the SPK file.
- **ncomch** (*int*) – The number of characters to reserve for comments.

Return type *int*

Returns The handle of the opened SPK file.

`spiceypy.spiceypy.spkpds` (*body*, *center*, *framestr*, *typenum*, *first*, *last*)

Perform routine error checks and if all check pass, pack the descriptor for an SPK segment

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkpds_c.html

Parameters

- **body** (*int*) – The NAIF ID code for the body of the segment.
- **center** (*int*) – The center of motion for body.
- **framestr** (*str*) – The frame for this segment.
- **typenum** (*int*) – The type of SPK segment to create.
- **first** (*float*) – The first epoch for which the segment is valid.
- **last** (*float*) – The last epoch for which the segment is valid.

Return type *ndarray*

Returns An SPK segment descriptor.

`spiceypy.spiceypy.spkpos` (*targ*, *et*, *ref*, *abcorr*, *obs*)

Return the position of a target body relative to an observing body, optionally corrected for light time (planetary aberration) and stellar aberration.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkpos_c.html

Parameters

- **targ** (*str*) – Target body name.
- **et** (*Union[float, ndarray]*) – Observer epoch.
- **ref** (*str*) – Reference frame of output position vector.
- **abcorr** (*str*) – Aberration correction flag.
- **obs** (*str*) – Observing body name.

Return type *Union[Tuple[ndarray, float], Tuple[ndarray, ndarray]]*

Returns Position of target, One way light time between observer and target.

`spiceypy.spiceypy.spkpv` (*handle, descr, et*)

For a specified SPK segment and time, return the state (position and velocity) of the segment's target body relative to its center of motion.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkpv_c.html

Parameters

- **handle** (*int*) – File handle.
- **descr** (*ndarray*) – Segment descriptor.
- **et** (*float*) – Evaluation epoch.

Return type *Tuple[int, ndarray, int]*

Returns Segment reference frame ID code, Output state vector, Center of state.

`spiceypy.spiceypy.spksfs` (*body, et, idlen*)

Search through loaded SPK files to find the highest-priority segment applicable to the body and time specified.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spksfs_c.html

Parameters

- **body** (*int*) – Body ID.
- **et** (*float*) – Ephemeris time.
- **idlen** (*int*) – Length of output segment ID string.

Return type *Tuple[int, ndarray, str, bool]*

Returns Handle of file containing the applicable segment, Descriptor of the applicable segment, Identifier of the applicable segment.

`spiceypy.spiceypy.spkssb` (*targ, et, ref*)

Return the state (position and velocity) of a target body relative to the solar system barycenter.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkssb_c.html

Parameters

- **targ** (*int*) – Target body.
- **et** (*float*) – Target epoch.
- **ref** (*str*) – Target reference frame.

Return type *ndarray*

Returns State of target.

`spiceypy.spiceypy.spksub` (*handle, descr, identin, begin, end, newh*)
 Extract a subset of the data in an SPK segment into a separate segment.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spksub_c.html

Parameters

- **handle** (*int*) – Handle of source segment.
- **descr** (*ndarray*) – Descriptor of source segment.
- **identin** (*str*) – Identifier of source segment.
- **begin** (*float*) – Beginning (initial epoch) of subset.
- **end** (*float*) – End (final epoch) of subset.
- **newh** (*int*) – Handle of new segment.

Return type `None`

`spiceypy.spiceypy.spkuds` (*descr*)
 Unpack the contents of an SPK segment descriptor.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkuds_c.html

Parameters **descr** (*ndarray*) – An SPK segment descriptor.

Return type `Tuple[int, int, int, int, float, float, int, int]`

Returns The NAIF ID code for the body of the segment, The center of motion for body, The ID code for the frame of this segment, The type of SPK segment, The first epoch for which the segment is valid, The last epoch for which the segment is valid, Beginning DAF address of the segment, Ending DAF address of the segment.

`spiceypy.spiceypy.spkuef` (*handle*)
 Unload an ephemeris file so that it will no longer be searched by the readers.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkuef_c.html

Parameters **handle** (*int*) – Handle of file to be unloaded

Return type `None`

`spiceypy.spiceypy.spkw02` (*handle, body, center, inframe, first, last, segid, intlen, n, polydg, cdata, btime*)
 Write a type 2 segment to an SPK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw02_c.html

Parameters

- **handle** (*int*) – Handle of an SPK file open for writing.
- **body** (*int*) – Body code for ephemeris object.
- **center** (*int*) – Body code for the center of motion of the body.
- **inframe** (*str*) – The reference frame of the states.
- **first** (*float*) – First valid time for which states can be computed.
- **last** (*float*) – Last valid time for which states can be computed.
- **segid** (*str*) – Segment identifier.
- **intlen** (*float*) – Length of time covered by logical record.
- **n** (*int*) – Number of coefficient sets.

- **polydg** (int) – Chebyshev polynomial degree.
- **cdata** (Iterable[float]) – Array of Chebyshev coefficients.
- **btime** (float) – Begin time of first logical record.

Return type None

`spiceypy.spiceypy.spkw03` (*handle, body, center, inframe, first, last, segid, intlen, n, polydg, cdata, btime*)

Write a type 3 segment to an SPK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw03_c.html

Parameters

- **handle** (int) – Handle of SPK file open for writing.
- **body** (int) – NAIF code for ephemeris object.
- **center** (int) – NAIF code for the center of motion of the body.
- **inframe** (str) – Reference frame name.
- **first** (float) – Start time of interval covered by segment.
- **last** (float) – End time of interval covered by segment.
- **segid** (str) – Segment identifier.
- **intlen** (float) – Length of time covered by record.
- **n** (int) – Number of records in segment.
- **polydg** (int) – Chebyshev polynomial degree.
- **cdata** (Iterable[float]) – Array of Chebyshev coefficients.
- **btime** (float) – Begin time of first record.

Return type None

`spiceypy.spiceypy.spkw05` (*handle, body, center, inframe, first, last, segid, gm, n, states, epochs*)

Write an SPK segment of type 5 given a time-ordered set of discrete states and epochs, and the gravitational parameter of a central body.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw05_c.html

Parameters

- **handle** (int) – Handle of an SPK file open for writing.
- **body** (int) – Body code for ephemeris object.
- **center** (int) – Body code for the center of motion of the body.
- **inframe** (str) – The reference frame of the states.
- **first** (float) – First valid time for which states can be computed.
- **last** (float) – Last valid time for which states can be computed.
- **segid** (str) – Segment identifier.
- **gm** (float) – Gravitational parameter of central body.
- **n** (int) – Number of states and epochs.
- **states** (Iterable[Iterable[float]]) – States.
- **epochs** (Iterable[float]) – Epochs.

Return type None

`spiceypy.spiceypy.spkw08` (*handle, body, center, inframe, first, last, segid, degree, n, states, epoch1, step*)

Write a type 8 segment to an SPK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw08_c.html

Parameters

- **handle** (`int`) – Handle of an SPK file open for writing.
- **body** (`int`) – NAIF code for an ephemeris object.
- **center** (`int`) – NAIF code for center of motion of “body”.
- **inframe** (`str`) – Reference frame name.
- **first** (`float`) – Start time of interval covered by segment.
- **last** (`float`) – End time of interval covered by segment.
- **segid** (`str`) – Segment identifier.
- **degree** (`int`) – Degree of interpolating polynomials.
- **n** (`int`) – Number of states.
- **states** (`Iterable[Iterable[float]]`) – Array of states.
- **epoch1** (`float`) – Epoch of first state in states array.
- **step** (`float`) – Time step separating epochs of states.

Return type None

`spiceypy.spiceypy.spkw09` (*handle, body, center, inframe, first, last, segid, degree, n, states, epochs*)

Write a type 9 segment to an SPK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw09_c.html

Parameters

- **handle** (`int`) – Handle of an SPK file open for writing.
- **body** (`int`) – NAIF code for an ephemeris object.
- **center** (`int`) – NAIF code for center of motion of “body”.
- **inframe** (`str`) – Reference frame name.
- **first** (`float`) – Start time of interval covered by segment.
- **last** (`float`) – End time of interval covered by segment.
- **segid** (`str`) – Segment identifier.
- **degree** (`int`) – Degree of interpolating polynomials.
- **n** (`int`) – Number of states.
- **states** (`Iterable[Iterable[float]]`) – Array of states.
- **epochs** (`Iterable[float]`) – Array of epochs corresponding to states.

Return type None

`spiceypy.spiceypy.spkw10` (*handle, body, center, inframe, first, last, segid, consts, n, elems, epochs*)

Write an SPK type 10 segment to the DAF open and attached to the input handle.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw10_c.html

Parameters

- **handle** (int) – The handle of a DAF file open for writing.
- **body** (int) – The NAIF ID code for the body of the segment.
- **center** (int) – The center of motion for body.
- **inframe** (str) – The reference frame for this segment.
- **first** (float) – The first epoch for which the segment is valid.
- **last** (float) – The last epoch for which the segment is valid.
- **segid** (str) – The string to use for segment identifier.
- **consts** (Iterable[float]) – The array of geophysical constants for the segment.
- **n** (int) – The number of element/epoch pairs to be stored.
- **elems** (Iterable[float]) – The collection of “two-line” element sets.
- **epochs** (Iterable[float]) – The epochs associated with the element sets.

Return type None

`spiceypy.spiceypy.spkw12(handle, body, center, inframe, first, last, segid, degree, n, states, epoch0, step)`

Write a type 12 segment to an SPK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw12_c.html

Parameters

- **handle** (int) – Handle of an SPK file open for writing.
- **body** (int) – NAIF code for an ephemeris object.
- **center** (int) – NAIF code for center of motion of body.
- **inframe** (str) – Reference frame name.
- **first** (float) – Start time of interval covered by segment.
- **last** (float) – End time of interval covered by segment.
- **segid** (str) – Segment identifier.
- **degree** (int) – Degree of interpolating polynomials.
- **n** (int) – Number of states.
- **states** (Iterable[Iterable[float]]) – Array of states.
- **epoch0** (float) – Epoch of first state in states array.
- **step** (float) – Time step separating epochs of states.

Return type None

`spiceypy.spiceypy.spkw13(handle, body, center, inframe, first, last, segid, degree, n, states, epochs)`

Write a type 13 segment to an SPK file.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw13_c.html

Parameters

- **handle** (int) – Handle of an SPK file open for writing.
- **body** (int) – NAIF code for an ephemeris object.

- **center** (int) – NAIF code for center of motion of body.
- **inframe** (str) – Reference frame name.
- **first** (float) – Start time of interval covered by segment.
- **last** (float) – End time of interval covered by segment.
- **segid** (str) – Segment identifier.
- **degree** (int) – Degree of interpolating polynomials.
- **n** (int) – Number of states.
- **states** (Iterable[Iterable[float]]) – Array of states.
- **epochs** (Iterable[float]) – Array of epochs corresponding to states.

Return type None

`spiceypy.spiceypy.spkw15` (*handle, body, center, inframe, first, last, segid, epoch, tp, pa, p, ecc, j2flg, pv, gm, j2, radius*)

Write an SPK segment of type 15 given a type 15 data record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw15_c.html

Parameters

- **handle** (int) – Handle of an SPK file open for writing.
- **body** (int) – Body code for ephemeris object.
- **center** (int) – Body code for the center of motion of the body.
- **inframe** (str) – The reference frame of the states.
- **first** (float) – First valid time for which states can be computed.
- **last** (float) – Last valid time for which states can be computed.
- **segid** (str) – Segment identifier.
- **epoch** (float) – Epoch of the periapse.
- **tp** (ndarray) – Trajectory pole vector.
- **pa** (ndarray) – Periapsis vector.
- **p** (float) – Semi-latus rectum.
- **ecc** (float) – Eccentricity.
- **j2flg** (float) – J2 processing flag.
- **pv** (Iterable[float]) – Central body pole vector.
- **gm** (float) – Central body GM.
- **j2** (float) – Central body J2.
- **radius** (float) – Equatorial radius of central body.

Return type None

`spiceypy.spiceypy.spkw17` (*handle, body, center, inframe, first, last, segid, epoch, eqel, rapol, decpol*)

Write an SPK segment of type 17 given a type 17 data record.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw17_c.html

Parameters

- **handle** (int) – Handle of an SPK file open for writing.
- **body** (int) – Body code for ephemeris object.
- **center** (int) – Body code for the center of motion of the body.
- **inframe** (str) – The reference frame of the states.
- **first** (float) – First valid time for which states can be computed.
- **last** (float) – Last valid time for which states can be computed.
- **segid** (str) – Segment identifier.
- **epoch** (float) – Epoch of elements in seconds past J2000.
- **eqel** (Iterable[float]) – Array of equinoctial elements.
- **rapol** (float) – Right Ascension of the pole of the reference plane.
- **decpol** (float) – Declination of the pole of the reference plane.

Return type None

`spiceypy.spiceypy.spkw18` (*handle, subtype, body, center, inframe, first, last, segid, degree, packets, epochs*)

Write a type 18 segment to an SPK file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw18_c.html

Parameters

- **handle** (int) – Handle of an SPK file open for writing.
- **subtype** (int) – SPK type 18 subtype code.
- **body** (int) – Body code for ephemeris object.
- **center** (int) – Body code for the center of motion of the body.
- **inframe** (str) – The reference frame of the states.
- **first** (float) – First valid time for which states can be computed.
- **last** (float) – Last valid time for which states can be computed.
- **segid** (str) – Segment identifier.
- **degree** (int) – Degree of interpolating polynomials.
- **packets** (Sequence[Iterable[float]]) – data packets
- **epochs** (Sequence[float]) – Array of epochs corresponding to states.

Return type None

`spiceypy.spiceypy.spkw20` (*handle, body, center, inframe, first, last, segid, intlen, n, polydg, cdata, dscale, tscale, initjd, initfr*)

Write a type 20 segment to an SPK file.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/spkw20_c.html

Parameters

- **handle** (int) – Handle of an SPK file open for writing.
- **body** (int) – Body code for ephemeris object.
- **center** (int) – Body code for the center of motion of the body.
- **inframe** (str) – The reference frame of the states.

- **first** (float) – First valid time for which states can be computed.
- **last** (float) – Last valid time for which states can be computed.
- **segid** (str) – Segment identifier.
- **intlen** (float) – Length of time covered by logical record (days).
- **n** (int) – Number of logical records in segment.
- **polydg** (int) – Chebyshev polynomial degree.
- **cdata** (ndarray) – Array of Chebyshev coefficients and positions.
- **dscale** (float) – Distance scale of data.
- **tscale** (float) – Time scale of data.
- **initjd** (float) – Integer part of begin time (TDB Julian date) of first record.
- **initfr** (float) – Fractional part of begin time (TDB Julian date) of first record.

Return type None

`spiceypy.spiceypy.srfc2s (code, bodyid, srflen=256)`

Translate a surface ID code, together with a body ID code, to the corresponding surface name. If no such name exists, return a string representation of the surface ID code.

note: from NAIF if isname is false, this case is not treated as an error.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/srfc2s_c.html

Parameters

- **code** (int) – Integer surface ID code to translate to a string.
- **bodyid** (int) – ID code of body associated with surface.
- **srflen** (int) – Available space in output string.
- **srflen** – int

Return type Tuple[str, bool]

Returns String corresponding to surface ID code.

`spiceypy.spiceypy.srfcss (code, bodstr, srflen=256)`

Translate a surface ID code, together with a body string, to the corresponding surface name. If no such surface name exists, return a string representation of the surface ID code.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/srfcss_c.html

Parameters

- **code** (int) – Integer surface ID code to translate to a string.
- **bodstr** (str) – Name or ID of body associated with surface.
- **srflen** (int) – Available space in output string.
- **srflen** – int

Return type Tuple[str, bool]

Returns String corresponding to surface ID code.

`spiceypy.spiceypy.srfnrm (method, target, et, fixref, srfts)`

Map array of surface points on a specified target body to the corresponding unit length outward surface normal vectors.

The surface of the target body may be represented by a triaxial ellipsoid or by topographic data provided by DSK files.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/srfnmr_c.html

Parameters

- **method** (*str*) – Computation method.
- **target** (*str*) – Name of target body.
- **et** (*float*) – Epoch in TDB seconds past J2000 TDB.
- **fixref** (*str*) – Body-fixed, body-centered target body frame.
- **srfpts** (*ndarray*) – Array of surface points.

Return type *ndarray*

Returns Array of outward, unit length normal vectors.

`spiceypy.spiceypy.srfrec` (*body, longitude, latitude*)

Convert planetocentric latitude and longitude of a surface point on a specified body to rectangular coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/srfrec_c.html

Parameters

- **body** (*int*) – NAIF integer code of an extended body.
- **longitude** (*float*) – Longitude of point in radians.
- **latitude** (*float*) – Latitude of point in radians.

Return type *ndarray*

Returns Rectangular coordinates of the point.

`spiceypy.spiceypy.srfs2c` (*srfstr, bodstr*)

Translate a surface string, together with a body string, to the corresponding surface ID code. The input strings may contain names or integer ID codes.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/srfs2c_c.html

Parameters

- **srfstr** (*str*) – Surface name or ID string.
- **bodstr** (*str*) – Body name or ID string.

Return type *Tuple[int, bool]*

Returns Integer surface ID code.

`spiceypy.spiceypy.srfsc` (*srfstr, bodyid*)

Translate a surface string, together with a body ID code, to the corresponding surface ID code. The input surface string may contain a name or an integer ID code.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/srfsc_c.html

Parameters

- **srfstr** (*str*) – Surface name or ID string.
- **bodyid** (*int*) – ID code of body associated with surface.

Return type *Tuple[int, bool]*

Returns Integer surface ID code.

`spiceypy.spiceypy.srfxpt` (*method, target, et, abcorr, obsrvr, dref, dvec*)

Deprecated: This routine has been superseded by the CSPICE routine `sincpt()`. This routine is supported for purposes of backward compatibility only.

Given an observer and a direction vector defining a ray, compute the surface intercept point of the ray on a target body at a specified epoch, optionally corrected for light time and stellar aberration.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/srfxpt_c.html

Parameters

- **method** (`str`) – Computation method.
- **target** (`str`) – Name of target body.
- **et** (`Union[float, Iterable[float]]`) – Epoch in ephemeris seconds past J2000 TDB.
- **abcorr** (`str`) – Aberration correction.
- **obsrvr** (`str`) – Name of observing body.
- **dref** (`str`) – Reference frame of input direction vector.
- **dvec** (`ndarray`) – Ray’s direction vector.

Return type `Union[Tuple[ndarray, float, float, ndarray, bool], Tuple[ndarray, ndarray, ndarray, ndarray, ndarray]]`

Returns Surface intercept point on the target body, Distance from the observer to the intercept point, Intercept epoch, Observer position relative to target center.

`spiceypy.spiceypy.ssize` (*newsize, cell*)

Set the size (maximum cardinality) of a CSPICE cell of any data type.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ssize_c.html

Parameters

- **newsize** (`int`) – Size (maximum cardinality) of the cell.
- **cell** (`SpiceCell`) – The cell.

Return type `SpiceCell`

Returns The updated cell.

`spiceypy.spiceypy.stelab` (*pobj, vobs*)

Correct the apparent position of an object for stellar aberration.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/stelab_c.html

Parameters

- **pobj** (`ndarray`) – Position of an object with respect to the observer.
- **vobs** (`ndarray`) – Velocity of the observer with respect to the Solar System barycenter.

Return type `ndarray`

Returns Apparent position of the object with respect to the observer, corrected for stellar aberration.

`spiceypy.spiceypy.stpool` (*item, nth, contin, lenout=256*)

Retrieve the *nth* string from the kernel pool variable, where the string may be continued across several components of the kernel pool variable.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/stpool_c.html

Parameters

- **item** (*str*) – Name of the kernel pool variable.
- **nth** (*int*) – Index of the full string to retrieve.
- **contin** (*str*) – Character sequence used to indicate continuation.
- **lenout** (*int*) – Available space in output string.

Return type `Tuple[str, int, bool]`

Returns A full string concatenated across continuations, The number of characters in the full string value.

`spiceypy.spiceypy.str2et` (*time*)

Convert a string representing an epoch to a double precision value representing the number of TDB seconds past the J2000 epoch corresponding to the input epoch.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/str2et_c.html

Parameters **time** (`Union[str, Iterable[str]]`) – A string representing an epoch.

Return type `Union[float, ndarray]`

Returns The equivalent value in seconds past J2000, TDB.

`spiceypy.spiceypy.subpnt` (*method, target, et, fixref, abcorr, obsrvr*)

Compute the rectangular coordinates of the sub-observer point on a target body at a specified epoch, optionally corrected for light time and stellar aberration.

This routine supersedes `subpt()`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/subpnt_c.html

Parameters

- **method** (*str*) – Computation method.
- **target** (*str*) – Name of target body.
- **et** (*float*) – Epoch in ephemeris seconds past J2000 TDB.
- **fixref** (*str*) – Body-fixed, body-centered target body frame.
- **abcorr** (*str*) – Aberration correction.
- **obsrvr** (*str*) – Name of observing body.

Return type `Tuple[ndarray, float, ndarray]`

Returns Sub-observer point on the target body, Sub-observer point epoch, Vector from observer to sub-observer point.

`spiceypy.spiceypy.subpt` (*method, target, et, abcorr, obsrvr*)

Deprecated: This routine has been superseded by the CSPICE routine `subpnt()`. This routine is supported for purposes of backward compatibility only.

Compute the rectangular coordinates of the sub-observer point on a target body at a particular epoch, optionally corrected for planetary (light time) and stellar aberration. Return these coordinates expressed in the body-fixed frame associated with the target body. Also, return the observer's altitude above the target body.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/subpt_c.html

Parameters

- **method** (*str*) – Computation method.

- **target** (*str*) – Name of target body.
- **et** (*Union[float, Iterable[float]]*) – Epoch in ephemeris seconds past J2000 TDB.
- **abcorr** (*str*) – Aberration correction.
- **obsrvr** (*str*) – Name of observing body.

Return type *Union[Tuple[ndarray, ndarray], Tuple[ndarray, float]]*

Returns Sub-observer point on the target body, Altitude of the observer above the target body.

`spiceypy.spiceypy.subslr` (*method, target, et, fixref, abcorr, obsrvr*)

Compute the rectangular coordinates of the sub-solar point on a target body at a specified epoch, optionally corrected for light time and stellar aberration.

This routine supersedes `subsol_c`.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/subslr_c.html

Parameters

- **method** (*str*) – Computation method.
- **target** (*str*) – Name of target body.
- **et** (*float*) – Epoch in ephemeris seconds past J2000 TDB.
- **fixref** (*str*) – Body-fixed, body-centered target body frame.
- **abcorr** (*str*) – Aberration correction.
- **obsrvr** (*str*) – Name of observing body.

Return type *Tuple[ndarray, float, ndarray]*

Returns Sub-solar point on the target body, Sub-solar point epoch, Vector from observer to sub-solar point.

`spiceypy.spiceypy.subsol` (*method, target, et, abcorr, obsrvr*)

Deprecated: This routine has been superseded by the CSPICE routine `subslr()`. This routine is supported for purposes of backward compatibility only.

Determine the coordinates of the sub-solar point on a target body as seen by a specified observer at a specified epoch, optionally corrected for planetary (light time) and stellar aberration.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/subsol_c.html

Parameters

- **method** (*str*) – Computation method.
- **target** (*str*) – Name of target body.
- **et** (*float*) – Epoch in ephemeris seconds past J2000 TDB.
- **abcorr** (*str*) – Aberration correction.
- **obsrvr** (*str*) – Name of observing body.

Return type *ndarray*

Returns Sub-solar point on the target body.

`spiceypy.spiceypy.sumad` (*array*)

Return the sum of the elements of a double precision array.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sumad_c.html

Parameters **array** (Sequence[float]) – Input Array.

Return type float

Returns The sum of the array.

`spiceypy.spiceypy.sumai(array)`

Return the sum of the elements of an integer array.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sumai_c.html

Parameters **array** (Sequence[int]) – Input Array.

Return type int

Returns The sum of the array.

`spiceypy.spiceypy.surfnm(a, b, c, point)`

This routine computes the outward-pointing, unit normal vector from a point on the surface of an ellipsoid.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/surfnm_c.html

Parameters

- **a** (float) – Length of the ellipsoid semi-axis along the x-axis.
- **b** (float) – Length of the ellipsoid semi-axis along the y-axis.
- **c** (float) – Length of the ellipsoid semi-axis along the z-axis.
- **point** (Iterable[float]) – Body-fixed coordinates of a point on the ellipsoid'

Return type ndarray

Returns Outward pointing unit normal to ellipsoid at point.

`spiceypy.spiceypy.surfpt(positn, u, a, b, c)`

Determine the intersection of a line-of-sight vector with the surface of an ellipsoid.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/surfpt_c.html

Parameters

- **positn** (Union[ndarray, Iterable[float]]) – Position of the observer in body-fixed frame.
- **u** (Union[ndarray, Iterable[float]]) – Vector from the observer in some direction.
- **a** (float) – Length of the ellipsoid semi-axis along the x-axis.
- **b** (float) – Length of the ellipsoid semi-axis along the y-axis.
- **c** (float) – Length of the ellipsoid semi-axis along the z-axis.

Return type Tuple[ndarray, bool]

Returns Point on the ellipsoid pointed to by u.

`spiceypy.spiceypy.surfvp(stvrtx, stdir, a, b, c)`

Find the state (position and velocity) of the surface intercept defined by a specified ray, ray velocity, and ellipsoid.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/surfvp_c.html

Parameters

- **stvrtx** (Iterable[float]) – State of ray's vertex.
- **stdir** (Iterable[float]) – State of ray's direction vector.

- **a** (float) – Length of the ellipsoid semi-axis along the x-axis.
- **b** (float) – Length of the ellipsoid semi-axis along the y-axis.
- **c** (float) – Length of the ellipsoid semi-axis along the z-axis.

Return type Tuple[ndarray, bool]

Returns State of surface intercept.

`spiceypy.spiceypy.swpool` (*agent, nnames, lenvals, names*)

Add a name to the list of agents to notify whenever a member of a list of kernel variables is updated.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/swpool_c.html

Parameters

- **agent** (str) – The name of an agent to be notified after updates.
- **nnames** (int) – The number of variables to associate with agent.
- **lenvals** (int) – Length of strings in the names array.
- **names** (Iterable[str]) – Variable names whose update causes the notice.

Return type None

`spiceypy.spiceypy.sxform` (*instring, tostring, et*)

Return the state transformation matrix from one frame to another at a specified epoch.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sxform_c.html

Parameters

- **instring** (str) – Name of the frame to transform from.
- **tostring** (str) – Name of the frame to transform to.
- **et** (Union[float, ndarray]) – Epoch of the state transformation matrix.

Return type ndarray

Returns A state transformation matrix.

`spiceypy.spiceypy.szpool` (*name*)

Return the kernel pool size limitations.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/szpool_c.html

Parameters **name** (str) – Name of the parameter to be returned.

Return type Tuple[int, bool]

Returns Value of parameter specified by name,

`spiceypy.spiceypy.termpt` (*method, ilusrc, target, et, fixref, abcorr, corloc, obsrvr, refvec, rolstp, ncuts, schstp, soltol, maxn*)

Find terminator points on a target body. The caller specifies half-planes, bounded by the illumination source center-target center vector, in which to search for terminator points.

The terminator can be either umbral or penumbral. The umbral terminator is the boundary of the region on the target surface where no light from the source is visible. The penumbral terminator is the boundary of the region on the target surface where none of the light from the source is blocked by the target itself.

The surface of the target body may be represented either by a triaxial ellipsoid or by topographic data.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/termpt_c.html

Parameters

- **method** (*str*) – Computation method.
- **ilusrc** (*str*) – Illumination source.
- **target** (*str*) – Name of target body.
- **et** (*float*) – Epoch in ephemeris seconds past J2000 TDB.
- **fixref** (*str*) – Body-fixed, body-centered target body frame.
- **abcorr** (*str*) – Aberration correction.
- **corloc** (*str*) – Aberration correction locus.
- **obsrvr** (*str*) – Name of observing body.
- **refvec** (*Iterable[float]*) – Reference vector for cutting half-planes.
- **rolstp** (*float*) – Roll angular step for cutting half-planes.
- **ncuts** (*int*) – Number of cutting half-planes.
- **schstp** (*float*) – Angular step size for searching.
- **soltol** (*float*) – Solution convergence tolerance.
- **maxn** (*int*) – Maximum number of entries in output arrays.

Return type `Tuple[ndarray, ndarray, ndarray, ndarray]`

Returns Counts of terminator points corresponding to cuts, Terminator points, Times associated with terminator points, Terminator vectors emanating from the observer

`spiceypy.spiceypy.timdef (action, item, lenout, value=None)`

Set and retrieve the defaults associated with calendar input strings.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/timdef_c.html

Parameters

- **action** (*str*) – the kind of action to take “SET” or “GET”.
- **item** (*str*) – the default item of interest.
- **lenout** (*int*) – the length of list for output.
- **value** (*Optional[str]*) – the optional string used if action is “SET”

Return type `str`

Returns the value associated with the default item.

`spiceypy.spiceypy.timeout (et, pictur, lenout=256)`

This vectorized routine converts an input epoch represented in TDB seconds past the TDB epoch of J2000 to a character string formatted to the specifications of a user’s format picture.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/timeout_c.html

Parameters

- **et** (*Union[ndarray, float]*) – An epoch in seconds past the ephemeris epoch J2000.
- **pictur** (*str*) – A format specification for the output string.
- **lenout** (*int*) – The length of the output string plus 1.

Return type `Union[ndarray, str]`

Returns A string representation of the input epoch.

`spiceypy.spiceypy.tipbod(ref, body, et)`

Return a 3x3 matrix that transforms positions in inertial coordinates to positions in body-equator-and-prime-meridian coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/tipbod_c.html

Parameters

- **ref** (str) – ID of inertial reference frame to transform from.
- **body** (int) – ID code of body.
- **et** (float) – Epoch of transformation.

Return type ndarray

Returns Transformation (position), inertial to prime meridian.

`spiceypy.spiceypy.tisbod(ref, body, et)`

Return a 6x6 matrix that transforms states in inertial coordinates to states in body-equator-and-prime-meridian coordinates.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/tisbod_c.html

Parameters

- **ref** (str) – ID of inertial reference frame to transform from.
- **body** (int) – ID code of body.
- **et** (float) – Epoch of transformation.

Return type ndarray

Returns Transformation (state), inertial to prime meridian.

`spiceypy.spiceypy.tkfram(typid)`

This routine returns the rotation from the input frame specified by ID to the associated frame given by FRAME.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/spicelib/tkfram.html

Parameters **typid** (int) – Class identification code for the instrument

Return type Tuple[ndarray, int, bool]

Returns Rotation matrix from the input frame to the returned reference frame, id for the reference frame

`spiceypy.spiceypy.tkvrnsn(item)`

Given an item such as the Toolkit or an entry point name, return the latest version string.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/tkvrnsn_c.html

Parameters **item** (str) – Item for which a version string is desired.

Return type str

Returns the latest version string.

`spiceypy.spiceypy.tparse(instr, lenout=256)`

Parse a time string and return seconds past the J2000 epoch on a formal calendar.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/tparse_c.html

Parameters

- **instr** (str) – Input time string, UTC.
- **lenout** (int) – Available space in output error message string.

Return type Tuple[float, str]

Returns Equivalent UTC seconds past J2000, Descriptive error message.

`spiceypy.spiceypy.tpicttr (sample, lenout=256, lenerr=256)`

Given a sample time string, create a time format picture suitable for use by the routine timout.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/tpicttr_c.html

Parameters

- **sample** (str) – A sample time string.
- **lenout** (int) – The length for the output picture string.
- **lenerr** (int) – The length for the output error string.

Return type Tuple[str, int, str]

Returns A format picture that describes sample, Flag indicating whether sample parsed successfully, Diagnostic returned if sample cannot be parsed

`spiceypy.spiceypy.trace (matrix)`

Return the trace of a 3x3 matrix.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/trace_c.html

Parameters **matrix** (Iterable[Iterable[float]]) – 3x3 matrix of double precision numbers.

Return type float

Returns The trace of matrix.

`spiceypy.spiceypy.trcdep ()`

Return the number of modules in the traceback representation.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/trcdep_c.html

Return type int

Returns The number of modules in the traceback.

`spiceypy.spiceypy.trcnam (index, namlen=256)`

Return the name of the module having the specified position in the trace representation. The first module to check in is at index 0.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/trcnam_c.html

Parameters

- **index** (int) – The position of the requested module name.
- **namlen** (int) – Available space in output name string.

Return type str

Returns The name at position index in the traceback.

`spiceypy.spiceypy.trcoff ()`

Disable tracing.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/trcoff_c.html

`spiceypy.spiceypy.tsetyr (year)`

Set the lower bound on the 100 year range.

Default value is 1969

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/tsetyr_c.html

Parameters **year** (*int*) – Lower bound on the 100 year interval of expansion

Return type *None*

`spiceypy.spiceypy.twopi()`

Return twice the value of pi (the ratio of the circumference of a circle to its diameter).

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/twopi_c.html

Return type *float*

Returns Twice the value of pi.

`spiceypy.spiceypy.twovec(axdef, indexa, plndef, indexp)`

Find the transformation to the right-handed frame having a given vector as a specified axis and having a second given vector lying in a specified coordinate plane.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/twovec_c.html

Parameters

- **axdef** (*Iterable[float]*) – Vector defining a principal axis.
- **indexa** (*int*) – Principal axis number of axdef (X=1, Y=2, Z=3).
- **plndef** (*Iterable[float]*) – Vector defining (with axdef) a principal plane.
- **indexp** (*int*) – Second axis number (with indexa) of principal plane.

Return type *ndarray*

Returns Output rotation matrix.

`spiceypy.spiceypy.txtopn(fname)`

Internal undocumented command for opening a new text file for subsequent write access.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ftncls_c.html#Files https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ftncls_c.html#Examples

Parameters **fname** (*str*) – name of the new text file to be opened.

Return type *int*

Returns FORTRAN logical unit of opened file

`spiceypy.spiceypy.tyear()`

Return the number of seconds in a tropical year.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/tyear_c.html

Return type *float*

Returns The number of seconds in a tropical year.

`spiceypy.spiceypy.ucase(inchar, lenout=None)`

Convert the characters in a string to uppercase.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ucase_c.html

Parameters

- **inchar** (*str*) – Input string.
- **lenout** (*Optional[int]*) – Optional Maximum length of output string.

Return type *str*

Returns Output string, all uppercase.

`spiceypy.spiceypy.ucrss(v1, v2)`

Compute the normalized cross product of two 3-vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/ucrss_c.html

Parameters

- **v1** (ndarray) – Left vector for cross product.
- **v2** (ndarray) – Right vector for cross product.

Return type ndarray

Returns Normalized cross product $v1 \times v2 / \text{abs}(v1 \times v2)$.

`spiceypy.spiceypy.uddc(udfunc, x, dx)`

SPICE private routine intended solely for the support of SPICE routines. Users should not call this routine directly due to the volatile nature of this routine.

This routine calculates the derivative of ‘udfunc’ with respect to time for ‘et’, then determines if the derivative has a negative value.

Use the `@spiceypy.utils.callbacks.SpiceUDFUNS` decorator to wrap a given python function that takes one parameter (float) and returns a float. For example:

```
@spiceypy.utils.callbacks.SpiceUDFUNS
def udfunc(et_in):
    pos, new_et = spice.spkpos("MERCURY", et_in, "J2000", "LT+S", "MOON")
    return new_et

deriv = spice.uddf(udfunc, et, 1.0)
```

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/uddc_c.html

Parameters

- **udfunc** – Name of the routine that computes the scalar value of interest.
- **x** – Independent variable of ‘udfunc’.
- **dx** – Interval from ‘x’ for derivative calculation.

Returns Boolean indicating if the derivative is negative.

`spiceypy.spiceypy.uddf(udfunc, x, dx)`

Routine to calculate the first derivative of a caller-specified function using a three-point estimation.

Use the `@spiceypy.utils.callbacks.SpiceUDFUNS` decorator to wrap a given python function that takes one parameter (float) and returns a float. For example:

```
@spiceypy.utils.callbacks.SpiceUDFUNS
def udfunc(et_in):
    pos, new_et = spice.spkpos("MERCURY", et_in, "J2000", "LT+S", "MOON")
    return new_et

deriv = spice.uddf(udfunc, et, 1.0)
```

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/uddf_c.html

Parameters

- **udfunc** – Name of the routine that computes the scalar value of interest.

- **x** – Independent variable of ‘udfunc’.
- **dx** – Interval from ‘x’ for derivative calculation.

Returns Approximate derivative of ‘udfunc’ at ‘x’

`spiceypy.spiceypy.udf(x)`

No-op routine for with an argument signature matching udfuns. Always returns 0.0 .

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/udf_c.html

Parameters **x** (float) – Double precision value, unused.

Return type float

Returns Double precision value, unused.

`spiceypy.spiceypy.union(a, b)`

Compute the union of two sets of any data type to form a third set.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/union_c.html

Parameters

- **a** (*SpiceCell*) – First input set.
- **b** (*SpiceCell*) – Second input set.

Return type *SpiceCell*

Returns Union of a and b.

`spiceypy.spiceypy.unitim(epoch, insys, outsys)`

Transform time from one uniform scale to another. The uniform time scales are TAI, TDT, TDB, ET, JED, JDTDB, JDTDT.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/unitim_c.html

Parameters

- **epoch** (float) – An epoch to be converted.
- **insys** (str) – The time scale associated with the input epoch.
- **outsys** (str) – The time scale associated with the function value.

Return type float

Returns The float in outsys that is equivalent to the epoch on the insys time scale.

`spiceypy.spiceypy.unload(filename)`

Unload a SPICE kernel.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/unload_c.html

Parameters **filename** (Union[str, Iterable[str]]) – The name of a kernel to unload.

Return type None

`spiceypy.spiceypy.unorm(v1)`

Normalize a double precision 3-vector and return its magnitude.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/unorm_c.html

Parameters **v1** (ndarray) – Vector to be normalized.

Return type Tuple[ndarray, float]

Returns Unit vector of v1, Magnitude of v1.

`spiceypy.spiceypy.unormg(v1, ndim)`

Normalize a double precision vector of arbitrary dimension and return its magnitude.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/unormg_c.html

Parameters

- **v1** (ndarray) – Vector to be normalized.
- **ndim** (int) – This is the dimension of v1 and vout.

Return type Tuple[ndarray, float]

Returns Unit vector of v1, Magnitude of v1.

`spiceypy.spiceypy.utc2et(utcstr)`

Convert an input time from Calendar or Julian Date format, UTC, to ephemeris seconds past J2000.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/utc2et_c.html

Parameters **utcstr** (str) – Input time string, UTC.

Return type float

Returns Output epoch, ephemeris seconds past J2000.

`spiceypy.spiceypy.vadd(v1, v2)`

Add two 3 dimensional vectors. http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vadd_c.html

Parameters

- **v1** (Iterable[float]) – First vector to be added.
- **v2** (Iterable[float]) – Second vector to be added.

Return type ndarray

Returns v1+v2

`spiceypy.spiceypy.vaddg(v1, v2, ndim)`

Add two n-dimensional vectors http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vaddg_c.html

Parameters

- **v1** (Iterable[float]) – First vector to be added.
- **v2** (Iterable[float]) – Second vector to be added.
- **ndim** (int) – Dimension of v1 and v2.

Return type ndarray

Returns v1+v2

`spiceypy.spiceypy.valid(insize, n, inset)`

Create a valid CSPICE set from a CSPICE Cell of any data type.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/valid_c.html

Parameters

- **insize** (int) – Size (maximum cardinality) of the set.
- **n** (int) – Initial no. of (possibly non-distinct) elements.
- **inset** (*SpiceCell*) – Set to be validated.

Return type *SpiceCell*

Returns validated set

`spiceypy.spiceypy.vcrss(v1, v2)`

Compute the cross product of two 3-dimensional vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vcrss_c.html

Parameters

- **v1** (ndarray) – Left hand vector for cross product.
- **v2** (ndarray) – Right hand vector for cross product.

Return type ndarray

Returns Cross product $v1 \times v2$.

`spiceypy.spiceypy.vdist(v1, v2)`

Return the distance between two three-dimensional vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vdist_c.html

Parameters

- **v1** (ndarray) – First vector in the dot product.
- **v2** (ndarray) – Second vector in the dot product.

Return type float

Returns the distance between $v1$ and $v2$

`spiceypy.spiceypy.vdistg(v1, v2, ndim)`

Return the distance between two vectors of arbitrary dimension.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vdistg_c.html

Parameters

- **v1** (ndarray) – ndim-dimensional double precision vector.
- **v2** (ndarray) – ndim-dimensional double precision vector.
- **ndim** (int) – Dimension of $v1$ and $v2$.

Return type float

Returns the distance between $v1$ and $v2$

`spiceypy.spiceypy.vdot(v1, v2)`

Compute the dot product of two double precision, 3-dimensional vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vdot_c.html

Parameters

- **v1** (ndarray) – First vector in the dot product.
- **v2** (ndarray) – Second vector in the dot product.

Return type float

Returns dot product of $v1$ and $v2$.

`spiceypy.spiceypy.vdotg(v1, v2, ndim)`

Compute the dot product of two double precision vectors of arbitrary dimension.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vdotg_c.html

Parameters

- **v1** (ndarray) – First vector in the dot product.

- **v2** (ndarray) – Second vector in the dot product.
- **ndim** (int) – Dimension of v1 and v2.

Return type float

Returns dot product of v1 and v2.

`spiceypy.spiceypy.vequ(v1)`

Make one double precision 3-dimensional vector equal to another.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vequ_c.html

Parameters **v1** (ndarray) – 3-dimensional double precision vector.

Return type ndarray

Returns 3-dimensional double precision vector set equal to vin.

`spiceypy.spiceypy.vequg(v1, ndim)`

Make one double precision vector of arbitrary dimension equal to another.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vequg_c.html

Parameters

- **v1** (ndarray) – ndim-dimensional double precision vector.
- **ndim** (int) – Dimension of vin (and also vout).

Return type ndarray

Returns ndim-dimensional double precision vector set equal to vin.

`spiceypy.spiceypy.vhat(v1)`

Find the unit vector along a double precision 3-dimensional vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vhat_c.html

Parameters **v1** (ndarray) – Vector to be unitized.

Return type ndarray

Returns Unit vector $v / \text{abs}(v)$.

`spiceypy.spiceypy.vhatg(v1, ndim)`

Find the unit vector along a double precision vector of arbitrary dimension.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vhatg_c.html

Parameters

- **v1** (ndarray) – Vector to be normalized.
- **ndim** (int) – Dimension of v1 (and also vout).

Return type ndarray

Returns Unit vector $v / \text{abs}(v)$.

`spiceypy.spiceypy.vlcom(a, v1, b, v2)`

Compute a vector linear combination of two double precision, 3-dimensional vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vlcom_c.html

Parameters

- **a** (float) – Coefficient of v1
- **v1** (Union[ndarray, Iterable[float]]) – Vector in 3-space

- **b** (float) – Coefficient of **v2**
- **v2** (Union[ndarray, Iterable[float]]) – Vector in 3-space

Return type ndarray

Returns Linear Vector Combination $a*v1 + b*v2$.

`spiceypy.spiceypy.vlcom3(a, v1, b, v2, c, v3)`

This subroutine computes the vector linear combination $a*v1 + b*v2 + c*v3$ of double precision, 3-dimensional vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vlcom3_c.html

Parameters

- **a** (float) – Coefficient of **v1**
- **v1** (Iterable[float]) – Vector in 3-space
- **b** (float) – Coefficient of **v2**
- **v2** (Iterable[float]) – Vector in 3-space
- **c** (float) – Coefficient of **v3**
- **v3** (Iterable[float]) – Vector in 3-space

Return type ndarray

Returns Linear Vector Combination $a*v1 + b*v2 + c*v3$

`spiceypy.spiceypy.vlcomg(n, a, v1, b, v2)`

Compute a vector linear combination of two double precision vectors of arbitrary dimension.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vlcomg_c.html

Parameters

- **n** (int) – Dimension of vector space
- **a** (float) – Coefficient of **v1**
- **v1** (Union[ndarray, Iterable[float]]) – Vector in n-space
- **b** (float) – Coefficient of **v2**
- **v2** (Union[ndarray, Iterable[float]]) – Vector in n-space

Return type ndarray

Returns Linear Vector Combination $a*v1 + b*v2$

`spiceypy.spiceypy.vminug(vin, ndim)`

Negate a double precision vector of arbitrary dimension.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vminug_c.html

Parameters

- **vin** (ndarray) – ndim-dimensional double precision vector to be negated.
- **ndim** (int) – Dimension of **vin**.

Return type ndarray

Returns ndim-dimensional double precision vector equal to -vin.

`spiceypy.spiceypy.vminus(vin)`

Negate a double precision 3-dimensional vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vminus_c.html

Parameters `vin` (ndarray) – Vector to be negated.

Return type ndarray

Returns Negated vector -v1.

`spiceypy.spiceypy.vnorm(v)`

Compute the magnitude of a double precision, 3-dimensional vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vnorm_c.html

Parameters `v` (ndarray) – Vector whose magnitude is to be found.

Return type float

Returns magnitude of `v` calculated in a numerically stable way

`spiceypy.spiceypy.vnormg(v, ndim)`

Compute the magnitude of a double precision vector of arbitrary dimension.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vnormg_c.html

Parameters

- `v` (ndarray) – Vector whose magnitude is to be found.
- `ndim` (int) – Dimension of `v`

Return type float

Returns magnitude of `v` calculated in a numerically stable way

`spiceypy.spiceypy.vpack(x, y, z)`

Pack three scalar components into a vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vpack_c.html

Parameters

- `x` (float) – first scalar component
- `y` (float) – second scalar component
- `z` (float) – third scalar component

Return type ndarray

Returns Equivalent 3-vector.

`spiceypy.spiceypy.vperp(a, b)`

Find the component of a vector that is perpendicular to a second vector. All vectors are 3-dimensional.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vperp_c.html

Parameters

- `a` (ndarray) – The vector whose orthogonal component is sought.
- `b` (ndarray) – The vector used as the orthogonal reference.

Return type ndarray

Returns The component of `a` orthogonal to `b`.

`spiceypy.spiceypy.vprjp(vin, plane)`

Project a vector onto a specified plane, orthogonally.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vprjp_c.html

Parameters

- **vin** (Iterable[float]) – The projected vector.
- **plane** (*Plane*) – Plane containing vin.

Return type ndarray

Returns Vector resulting from projection.

`spiceypy.spiceypy.vprjpi(vin, projpl, invpl)`

Find the vector in a specified plane that maps to a specified vector in another plane under orthogonal projection.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vprjpi_c.html

Parameters

- **vin** (Iterable[float]) – The projected vector.
- **projpl** (*Plane*) – Plane containing vin.
- **invpl** (*Plane*) – Plane containing inverse image of vin.

Return type Tuple[ndarray, bool]

Returns Inverse projection of vin.

`spiceypy.spiceypy.vproj(a, b)`

Find the projection of one vector onto another vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vproj_c.html

Parameters

- **a** (ndarray) – The vector to be projected.
- **b** (ndarray) – The vector onto which a is to be projected.

Return type ndarray

Returns The projection of a onto b.

`spiceypy.spiceypy.vrel(v1, v2)`

Return the relative difference between two 3-dimensional vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vrel_c.html

Parameters

- **v1** (Iterable[float]) – First vector
- **v2** (Iterable[float]) – Second vector

Return type float

Returns the relative difference between v1 and v2.

`spiceypy.spiceypy.vrelg(v1, v2, ndim)`

Return the relative difference between two vectors of general dimension.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vrelg_c.html

Parameters

- **v1** (Iterable[float]) – First vector

- **v2** (Iterable[float]) – Second vector
- **ndim** (int) – Dimension of v1 and v2.

Return type float

Returns the relative difference between v1 and v2.

`spiceypy.spiceypy.vrotr(v, axis, theta)`

Rotate a vector about a specified axis vector by a specified angle and return the rotated vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vrotr_c.html

Parameters

- **v** (ndarray) – Vector to be rotated.
- **axis** (ndarray) – Axis of the rotation.
- **theta** (float) – Angle of rotation (radians).

Return type ndarray

Returns Result of rotating v about axis by theta

`spiceypy.spiceypy.vsc1(s, v1)`

Multiply a scalar and a 3-dimensional double precision vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vsc1_c.html

Parameters

- **s** (float) – Scalar to multiply a vector
- **v1** (ndarray) – Vector to be multiplied

Return type ndarray

Returns Product vector, s*v1.

`spiceypy.spiceypy.vsc1g(s, v1, ndim)`

Multiply a scalar and a double precision vector of arbitrary dimension.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vsc1g_c.html

Parameters

- **s** (float) – Scalar to multiply a vector
- **v1** (ndarray) – Vector to be multiplied
- **ndim** (int) – Dimension of v1

Return type ndarray

Returns Product vector, s*v1.

`spiceypy.spiceypy.vsep(v1, v2)`

Find the separation angle in radians between two double precision, 3-dimensional vectors. This angle is defined as zero if either vector is zero.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vsep_c.html

Parameters

- **v1** (ndarray) – First vector
- **v2** (ndarray) – Second vector

Return type float

Returns separation angle in radians

`spiceypy.spiceypy.vsepg(v1, v2, ndim)`

Find the separation angle in radians between two double precision vectors of arbitrary dimension. This angle is defined as zero if either vector is zero.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vsepg_c.html

Parameters

- **v1** (ndarray) – First vector
- **v2** (ndarray) – Second vector
- **ndim** (int) – The number of elements in v1 and v2.

Return type float

Returns separation angle in radians

`spiceypy.spiceypy.vsub(v1, v2)`

Compute the difference between two 3-dimensional, double precision vectors.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vsub_c.html

Parameters

- **v1** (ndarray) – First vector (minuend).
- **v2** (ndarray) – Second vector (subtrahend).

Return type ndarray

Returns Difference vector, $v1 - v2$.

`spiceypy.spiceypy.vsubg(v1, v2, ndim)`

Compute the difference between two double precision vectors of arbitrary dimension.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vsubg_c.html

Parameters

- **v1** (ndarray) – First vector (minuend).
- **v2** (ndarray) – Second vector (subtrahend).
- **ndim** (int) – Dimension of v1, v2, and vout.

Return type ndarray

Returns Difference vector, $v1 - v2$.

`spiceypy.spiceypy.vtmv(v1, matrix, v2)`

Multiply the transpose of a 3-dimensional column vector a 3x3 matrix, and a 3-dimensional column vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vtmv_c.html

Parameters

- **v1** (ndarray) – 3 dimensional double precision column vector.
- **matrix** (ndarray) – 3x3 double precision matrix.
- **v2** (ndarray) – 3 dimensional double precision column vector.

Return type float

Returns the result of $(v1^{**t} * matrix * v2)$.

`spiceypy.spiceypy.vtmvg(v1, matrix, v2, nrow, ncol)`

Multiply the transpose of a n-dimensional column vector a nxm matrix, and a m-dimensional column vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vtmvg_c.html

Parameters

- **v1** (ndarray) – n-dimensional double precision column vector.
- **matrix** (ndarray) – nxm double precision matrix.
- **v2** (ndarray) – m-dimensional double precision column vector.
- **nrow** (int) – Number of rows in matrix (number of rows in v1.)
- **ncol** (int) – Number of columns in matrix (number of rows in v2.)

Return type float

Returns the result of $(v1^{**t} * matrix * v2)$

`spiceypy.spiceypy.vupack(v)`

Unpack three scalar components from a vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vupack_c.html

Parameters **v** (ndarray) – Vector

Return type Tuple[float, float, float]

Returns (x, y, z)

`spiceypy.spiceypy.vzero(v)`

Indicate whether a 3-vector is the zero vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vzero_c.html

Parameters **v** (ndarray) – Vector to be tested

Return type bool

Returns true if and only if v is the zero vector

`spiceypy.spiceypy.vzerog(v, ndim)`

Indicate whether a general-dimensional vector is the zero vector.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/vzerog_c.html

Parameters

- **v** (ndarray) – Vector to be tested
- **ndim** (int) – Dimension of v

Return type bool

Returns true if and only if v is the zero vector

`spiceypy.spiceypy.wncard(window)`

Return the cardinality (number of intervals) of a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wncard_c.html

Parameters **window** (*SpiceCell*) – Input window

Return type int

Returns the cardinality of the input window.

`spiceypy.spiceypy.wncomd(left, right, window)`

Determine the complement of a double precision window with respect to a specified interval.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wncomd_c.html

Parameters

- **left** (float) – left endpoints of complement interval.
- **right** (float) – right endpoints of complement interval.
- **window** (*SpiceCell*) – Input window

Return type *SpiceCell*

Returns Complement of window with respect to left and right.

`spiceypy.spiceypy.wncond(left, right, window)`

Contract each of the intervals of a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wncond_c.html

Parameters

- **left** (float) – Amount added to each left endpoint.
- **right** (float) – Amount subtracted from each right endpoint.
- **window** (*SpiceCell*) – Window to be contracted

Return type *SpiceCell*

Returns Contracted Window.

`spiceypy.spiceypy.wndifd(a, b)`

Place the difference of two double precision windows into a third window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wndifd_c.html

Parameters

- **a** (*SpiceCell*) – Input window A.
- **b** (*SpiceCell*) – Input window B.

Return type *SpiceCell*

Returns Difference of a and b.

`spiceypy.spiceypy.wnelmd(point, window)`

Determine whether a point is an element of a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnelmd_c.html

Parameters

- **point** (float) – Input point.
- **window** (*SpiceCell*) – Input window

Return type bool

Returns returns True if point is an element of window.

`spiceypy.spiceypy.wnexpd(left, right, window)`

Expand each of the intervals of a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnexpd_c.html

Parameters

- **left** (float) – Amount subtracted from each left endpoint.
- **right** (float) – Amount added to each right endpoint.
- **window** (*SpiceCell*) – Window to be expanded.

Return type *SpiceCell*

Returns Expanded Window.

`spiceypy.spiceypy.wnextd(side, window)`

Extract the left or right endpoints from a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnextd_c.html

Parameters

- **side** (str) – Extract left “L” or right “R” endpoints.
- **window** (*SpiceCell*) – Window to be extracted.

Return type *SpiceCell*

Returns Extracted Window.

`spiceypy.spiceypy.wnfetd(window, n)`

Fetch a particular interval from a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnfetd_c.html

Parameters

- **window** (*SpiceCell*) – Input window
- **n** (int) – Index of interval to be fetched.

Return type Tuple[float, float]

Returns Left, right endpoints of the nth interval.

`spiceypy.spiceypy.wnfield(small, window)`

Fill small gaps between adjacent intervals of a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnfield_c.html

Parameters

- **small** (float) – Limiting measure of small gaps.
- **window** (*SpiceCell*) – Window to be filled

Return type *SpiceCell*

Returns Filled Window.

`spiceypy.spiceypy.wnfltd(small, window)`

Filter (remove) small intervals from a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnfltd_c.html

Parameters

- **small** (float) – Limiting measure of small intervals.
- **window** (*SpiceCell*) – Window to be filtered.

Return type *SpiceCell*

Returns Filtered Window.

`spiceypy.spiceypy.wnincd(left, right, window)`

Determine whether an interval is included in a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnincd_c.html

Parameters

- **left** (float) – Left interval
- **right** (float) – Right interval
- **window** (*SpiceCell*) – Input window

Return type bool

Returns Returns True if the input interval is included in window.

`spiceypy.spiceypy.wninsd(left, right, window)`

Insert an interval into a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wninsd_c.html

Parameters

- **left** (float) – Left endpoints of new interval.
- **right** (float) – Right endpoints of new interval.
- **window** (*SpiceCell*) – Input window.

Return type None

`spiceypy.spiceypy.wnintd(a, b)`

Place the intersection of two double precision windows into a third window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnintd_c.html

Parameters

- **a** (*SpiceCell*) – Input window A.
- **b** (*SpiceCell*) – Input window B.

Return type *SpiceCell*

Returns Intersection of a and b.

`spiceypy.spiceypy.wnreld(a, op, b)`

Compare two double precision windows.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnreld_c.html

Parameters

- **a** (*SpiceCell*) – First window.
- **op** (str) – Comparison operator.
- **b** (*SpiceCell*) – Second window.

Return type bool

Returns The result of comparison: a (op) b.

`spiceypy.spiceypy.wnsumd(window)`

Summarize the contents of a double precision window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnsumd_c.html

Parameters **window** (*SpiceCell*) – Window to be summarized.

Return type Tuple[float, float, float, int, int]

Returns Total measure of intervals in window, Average measure, Standard deviation, Location of shortest interval, Location of longest interval.

`spiceypy.spiceypy.wnunid(a, b)`

Place the union of two double precision windows into a third window.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnunid_c.html

Parameters

- **a** (*SpiceCell*) – Input window A.
- **b** (*SpiceCell*) – Input window B.

Return type *SpiceCell*

Returns Union of a and b.

`spiceypy.spiceypy.wnvald(insize, n, window)`

Form a valid double precision window from the contents of a window array.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/wnvald_c.html

Parameters

- **insize** (int) – Size of window.
- **n** (int) – Original number of endpoints.
- **window** (*SpiceCell*) – Input window.

Return type *SpiceCell*

Returns The union of the intervals in the input cell.

`spiceypy.spiceypy.writln(line, unit)`

Internal undocumented command for writing a text line to a logical unit

No URL available; relevant lines from SPICE source:

FORTRAN SPICE, writln.f:

```
C$Procedure      WRITLN ( Write a text line to a logical unit )
  SUBROUTINE WRITLN ( LINE, UNIT )
    CHARACTER*(*)    LINE
    INTEGER           UNIT

C   Variable  I/O  Description
C   -----  ---  -----
C   LINE      I   The line which is to be written to UNIT.
C   UNIT      I   The Fortran unit number to use for output.
```

CSPICE, writln.c:

```
/* $Procedure      WRITLN ( Write a text line to a logical unit ) */
/* Subroutine */ int writln_(char *line, integer *unit, ftnlen line_len)
```

Parameters

- **line** (str) – The line which is to be written to UNIT.
- **unit** (int) – The Fortran unit number to use for output.

Return type None

`spiceypy.spiceypy.xf2eul` (*xform*, *axisa*, *axisb*, *axisc*)
http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/xf2eul_c.html

Parameters

- **xform** (ndarray) – state transformation matrix
- **axisa** (int) – Axis A of the Euler angle factorization.
- **axisb** (int) – Axis B of the Euler angle factorization.
- **axisc** (int) – Axis C of the Euler angle factorization.

Return type Tuple[ndarray, int]

Returns (eulang, unique)

`spiceypy.spiceypy.xf2rav` (*xform*)

This routine determines the rotation matrix and angular velocity of the rotation from a state transformation matrix. http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/xf2rav_c.html

Parameters **xform** (ndarray) – state transformation matrix

Return type Tuple[ndarray, ndarray]

Returns rotation associated with xform, angular velocity associated with xform.

`spiceypy.spiceypy.xfmsta` (*input_state*, *input_coord_sys*, *output_coord_sys*, *body*)

Transform a state between coordinate systems.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/xfmsta_c.html

Parameters

- **input_state** (ndarray) – Input state.
- **input_coord_sys** (str) – Current (input) coordinate system.
- **output_coord_sys** (str) – Desired (output) coordinate system.
- **body** (str) – Name or NAIF ID of body with which coordinates are associated (if applicable).

Return type ndarray

Returns Converted output state

`spiceypy.spiceypy.xpose` (*m*)

Transpose a 3x3 matrix

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/xpose_c.html

Parameters **m** (Union[ndarray, Iterable[Iterable[float]]]) – Matrix to be transposed

Return type ndarray

Returns Transposed matrix

`spiceypy.spiceypy.xpose6` (*m*)

Transpose a 6x6 matrix

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/xpose6_c.html

Parameters **m** (Iterable[Iterable[float]]) – Matrix to be transposed

Return type ndarray

Returns Transposed matrix

`spiceypy.spiceypy.xposeg(matrix, nrow, ncol)`

Transpose a matrix of arbitrary size in place, the matrix need not be square.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/xposeg_c.html

Parameters

- **matrix** (Union[ndarray, Iterable[Iterable[float]]]) – Matrix to be transposed
- **nrow** (int) – Number of rows of input matrix.
- **ncol** (int) – Number of columns of input matrix

Return type ndarray

Returns Transposed matrix

`spiceypy.spiceypy.zzdynrot(typid, center, et)`

Find the rotation from a dynamic frame ID to the associated frame at the time requested

Parameters

- **typid** (int) – ID code for the dynamic frame
- **center** (int) – the ID for the center of the frame
- **et** (float) – Epoch measured in seconds past J2000

Return type Tuple[ndarray, int]

Returns Rotation matrix from the input frame to the returned associated frame, id for the associated frame

7.2 spiceypy.utils.support_types module

The MIT License (MIT)

Copyright (c) [2015-2019] [Andrew Annex]

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The MIT License (MIT)

Copyright (c) 2013 Philipp Rasch

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
class spiceypy.utils.support_types.Cell_Bool (size)
    Bases: spiceypy.utils.support_types.SpiceCell

class spiceypy.utils.support_types.Cell_Char (size, length)
    Bases: spiceypy.utils.support_types.SpiceCell

class spiceypy.utils.support_types.Cell_Double (size)
    Bases: spiceypy.utils.support_types.SpiceCell

class spiceypy.utils.support_types.Cell_Int (size)
    Bases: spiceypy.utils.support_types.SpiceCell

class spiceypy.utils.support_types.Cell_Time (size)
    Bases: spiceypy.utils.support_types.SpiceCell

class spiceypy.utils.support_types.DataType
    Bases: object

    BOOL = 4

    CHR = 0

    DP = 1

    INT = 2

    SPICE_BOOL = 4

    SPICE_CHR = 0

    SPICE_DP = 1

    SPICE_INT = 2

    SPICE_TIME = 3

    TIME = 3

class spiceypy.utils.support_types.DoubleArrayType
    Bases: object

    from_array (param)

    from_list (param)

    from_ndarray (param)

    from_param (param)

    from_tuple (param)

class spiceypy.utils.support_types.DoubleMatrixType
    Bases: object

    from_list (param)

    from_matrix (param)
```

```

    from_ndarray (param)
    from_param (param)
    from_tuple (param)
class spiceypy.utils.support_types.Ellipse
    Bases: ctypes.Structure
    property center
    property semi_major
    property semi_minor
class spiceypy.utils.support_types.IntArrayType
    Bases: object
    from_array (param)
    from_list (param)
    from_ndarray (param)
    from_param (param)
    from_tuple (param)
class spiceypy.utils.support_types.IntMatrixType
    Bases: object
    from_list (param)
    from_matrix (param)
    from_ndarray (param)
    from_param (param)
    from_tuple (param)
class spiceypy.utils.support_types.Plane
    Bases: ctypes.Structure
    property constant
    property normal
spiceypy.utils.support_types.SPICEBOOL_CELL (size)
    Returns a Bool Spice Cell with a given size :param size: number of elements :type size: int :return: empty Spice
    Cell :rtype: spiceypy.utils.support_types.SpiceCell
spiceypy.utils.support_types.SPICECHAR_CELL (size, length)
    Returns a Char Spice Cell with a given size :param size: number of elements :type size: int :param length: width
    of elements :type length: int :return: empty Spice Cell :rtype: spiceypy.utils.support_types.SpiceCell
spiceypy.utils.support_types.SPICEDOUBLE_CELL (size)
    Returns a Double Spice Cell with a given size :param size: number of elements :type size: int :return: empty
    Spice Cell :rtype: spiceypy.utils.support_types.SpiceCell
spiceypy.utils.support_types.SPICEINT_CELL (size)
    Returns a Int Spice Cell with a given size :param size: number of elements :type size: int :return: empty Spice
    Cell :rtype: spiceypy.utils.support_types.SpiceCell
spiceypy.utils.support_types.SPICETIME_CELL (size)
    Returns a Time Spice Cell with a given size :param size: number of elements :type size: int :return: empty Spice
    Cell :rtype: spiceypy.utils.support_types.SpiceCell

```

```

class spiceypy.utils.support_types.SpiceCell (dtype=None,  length=None,  size=None,
                                              card=None,  isSet=None,  base=None,
                                              data=None)

    Bases: ctypes.Structure

    CTRLBLOCK = 6

    DATATYPES_ENUM = {'bool': 4, 'char': 0, 'double': 1, 'int': 2, 'time': 3}

    DATATYPES_GET = [<function _char_getter>, <function _double_getter>, <function _int_ge
    adjust
        Structure/Union member

    base
        Structure/Union member

    baseSize = 6

    classmethod bool (size)

    card
        Structure/Union member

    classmethod character (size, length)

    data
        Structure/Union member

    classmethod double (size)

    dtype
        Structure/Union member

    init
        Structure/Union member

    classmethod integer (size)

    isSet
        Structure/Union member

    is_bool ()

    is_char ()

    is_double ()

    is_int ()

    is_set ()

    is_time ()

    length
        Structure/Union member

    minCharLen = 6

    reset ()

    size
        Structure/Union member

    classmethod time (size)

class spiceypy.utils.support_types.SpiceDLADescr
    Bases: ctypes.Structure

```

```
    property bwdptr
    property cbase
    property csize
    property dbase
    property dsize
    property fwdptr
    property ibase
    property isize

class spiceypy.utils.support_types.SpiceDSKDescr
    Bases: _ctypes.Structure

    property center
    property colmax
    property colmin
    property co2max
    property co2min
    property co3max
    property co3min
    property corpar
    property corsys
    property dclass
    property dtype
    property frmcode
    property start
    property stop
    property surface

class spiceypy.utils.support_types.SpiceEKAttDsc
    Bases: _ctypes.Structure

    property cclass
    property dtype
    property indexd
    property nullok
    property size
    property strlen

class spiceypy.utils.support_types.SpiceEKDataType
    Bases: ctypes.c_int

    SPICE_BOOL = 4
    SPICE_CHR = 0
```

```

    SPICE_DP = 1
    SPICE_INT = 2
    SPICE_TIME = 3

class spiceypy.utils.support_types.SpiceEKExprClass
    Bases: ctypes.c_int

    SPICE_EK_EXP_COL = 0
    SPICE_EK_EXP_EXPR = 2
    SPICE_EK_EXP_FUNC = 1

class spiceypy.utils.support_types.SpiceEKSegSum
    Bases: _ctypes.Structure

    property cdescrs
    property cnames
    property ncols
    property nrows
    property tabnam

class spiceypy.utils.support_types.SpiceSPK18Subtype
    Bases: ctypes.c_int

    S18TP0 = 0
    S18TP1 = 1

exception spiceypy.utils.support_types.SpiceyError (value, found=None)
    Bases: Exception

    SpiceyError wraps CSPICE errors. :type value: str

spiceypy.utils.support_types.c_int_vector_to_bool_python(x)

spiceypy.utils.support_types.c_matrix_to_numpy(x)
    Convert a ctypes 2d array (or matrix) into a numpy array for python use :param x: thing to convert :return:
    numpy.ndarray

spiceypy.utils.support_types.c_vector_to_python(x)
    Convert the c vector data into the correct python data type (numpy arrays or strings) :param x: :return:

spiceypy.utils.support_types.empty_char_array(x_len=None, y_len=None)

spiceypy.utils.support_types.empty_double_matrix(x=3, y=3)

spiceypy.utils.support_types.empty_double_vector(n)

spiceypy.utils.support_types.empty_int_matrix(x=3, y=3)

spiceypy.utils.support_types.empty_int_vector(n)

spiceypy.utils.support_types.empty_spice_ek_data_type_vector(n)

spiceypy.utils.support_types.empty_spice_ek_expr_class_vector(n)

spiceypy.utils.support_types.is_iterable(i)
    From stackoverflow https://stackoverflow.com/questions/1055360/how-to-tell-a-variable-is-iterable-but-not-a-string/44328500#44328500 :param i: input collection :return:

spiceypy.utils.support_types.list_to_char_array(arg, x_len=None, y_len=None)

```

```
spiceypy.utils.support_types.list_to_char_array_ptr(input, x_len=None,
                                                    y_len=None)
spiceypy.utils.support_types.string_to_char_p(inobject, inlen=None)
```

Parameters

- **inobject** – input string, int for getting null string of length of int
- **inlen** – optional parameter, length of a given string can be specified

Returns

```
spiceypy.utils.support_types.to_double_matrix(x)
spiceypy.utils.support_types.to_double_vector(x)
spiceypy.utils.support_types.to_int_matrix(x)
spiceypy.utils.support_types.to_int_vector(x)
spiceypy.utils.support_types.to_python_string(in_string)
```

7.3 spiceypy.utils.libspice module

The MIT License (MIT)

Copyright (c) [2015-2019] [Andrew Annex]

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
spiceypy.utils.libspicehelper.s_dla_p
    alias of spiceypy.utils.libspicehelper.LP_SpiceDLADescr
spiceypy.utils.libspicehelper.s_dsk_p
    alias of spiceypy.utils.libspicehelper.LP_SpiceDSKDescr
spiceypy.utils.libspicehelper.s_eka_p
    alias of spiceypy.utils.libspicehelper.LP_SpiceEKAttDsc
spiceypy.utils.libspicehelper.s_eks_p
    alias of spiceypy.utils.libspicehelper.LP_SpiceEKSegSum
spiceypy.utils.libspicehelper.s_elip_p
    alias of spiceypy.utils.libspicehelper.LP_Ellipse
spiceypy.utils.libspicehelper.s_plan_p
    alias of spiceypy.utils.libspicehelper.LP_Plane
```


INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

S

`spiceypy.spiceypy`, [153](#)
`spiceypy.utils.libspicehelper`, [307](#)
`spiceypy.utils.support_types`, [301](#)

A

`adjust` (*spiceypy.utils.support_types.SpiceCell* attribute), 304
`appndc()` (*in module spiceypy.spiceypy*), 153
`appnnd()` (*in module spiceypy.spiceypy*), 153
`appndi()` (*in module spiceypy.spiceypy*), 153
`axisar()` (*in module spiceypy.spiceypy*), 154

B

`b1900()` (*in module spiceypy.spiceypy*), 154
`b1950()` (*in module spiceypy.spiceypy*), 154
`badkpvr()` (*in module spiceypy.spiceypy*), 154
`base` (*spiceypy.utils.support_types.SpiceCell* attribute), 304
`baseSize` (*spiceypy.utils.support_types.SpiceCell* attribute), 304
`bltfrm()` (*in module spiceypy.spiceypy*), 154
`bodc2n()` (*in module spiceypy.spiceypy*), 155
`bodc2s()` (*in module spiceypy.spiceypy*), 155
`boddef()` (*in module spiceypy.spiceypy*), 155
`bodfnd()` (*in module spiceypy.spiceypy*), 155
`bodn2c()` (*in module spiceypy.spiceypy*), 156
`bods2c()` (*in module spiceypy.spiceypy*), 156
`bodvar()` (*in module spiceypy.spiceypy*), 156
`bodvcd()` (*in module spiceypy.spiceypy*), 156
`bodvrd()` (*in module spiceypy.spiceypy*), 156
`BOOL` (*spiceypy.utils.support_types.DataType* attribute), 302
`bool()` (*spiceypy.utils.support_types.SpiceCell* class method), 304
`brcktd()` (*in module spiceypy.spiceypy*), 157
`brckti()` (*in module spiceypy.spiceypy*), 157
`bschoc()` (*in module spiceypy.spiceypy*), 157
`bschoi()` (*in module spiceypy.spiceypy*), 158
`bsrchc()` (*in module spiceypy.spiceypy*), 158
`bsrchr()` (*in module spiceypy.spiceypy*), 158
`bsrchi()` (*in module spiceypy.spiceypy*), 158
`bwdptr()` (*spiceypy.utils.support_types.SpiceDLADescr* property), 304

C

`c_int_vector_to_bool_python()` (*in module*

spiceypy.utils.support_types), 306
`c_matrix_to_numpy()` (*in module*
spiceypy.utils.support_types), 306
`c_vector_to_python()` (*in module*
spiceypy.utils.support_types), 306
`card` (*spiceypy.utils.support_types.SpiceCell* attribute), 304
`card()` (*in module spiceypy.spiceypy*), 159
`cbase()` (*spiceypy.utils.support_types.SpiceDLADescr* property), 305
`ccifrm()` (*in module spiceypy.spiceypy*), 159
`cclass()` (*spiceypy.utils.support_types.SpiceEKAttDsc* property), 305
`cdescrs()` (*spiceypy.utils.support_types.SpiceEKSegSum* property), 306
`Cell_Bool` (*class in spiceypy.utils.support_types*), 302
`cell_bool()` (*in module spiceypy.spiceypy*), 159
`Cell_Char` (*class in spiceypy.utils.support_types*), 302
`cell_char()` (*in module spiceypy.spiceypy*), 159
`Cell_Double` (*class in spiceypy.utils.support_types*), 302
`cell_double()` (*in module spiceypy.spiceypy*), 159
`Cell_Int` (*class in spiceypy.utils.support_types*), 302
`cell_int()` (*in module spiceypy.spiceypy*), 159
`Cell_Time` (*class in spiceypy.utils.support_types*), 302
`cell_time()` (*in module spiceypy.spiceypy*), 159
`center()` (*spiceypy.utils.support_types.Ellipse* property), 303
`center()` (*spiceypy.utils.support_types.SpiceDSKDescr* property), 305
`cgv2el()` (*in module spiceypy.spiceypy*), 159
`character()` (*spiceypy.utils.support_types.SpiceCell* class method), 304
`chbder()` (*in module spiceypy.spiceypy*), 160
`check_for_spice_error()` (*in module*
spiceypy.spiceypy), 160
`chkin()` (*in module spiceypy.spiceypy*), 160
`chkout()` (*in module spiceypy.spiceypy*), 160
`CHR` (*spiceypy.utils.support_types.DataType* attribute), 302
`cidfrm()` (*in module spiceypy.spiceypy*), 160
`ckcls()` (*in module spiceypy.spiceypy*), 160

- `ckcov()` (in module `spiceypy.spiceypy`), 161
- `ckfrot()` (in module `spiceypy.spiceypy`), 161
- `ckgp()` (in module `spiceypy.spiceypy`), 161
- `ckgpav()` (in module `spiceypy.spiceypy`), 161
- `cklpf()` (in module `spiceypy.spiceypy`), 162
- `ckobj()` (in module `spiceypy.spiceypy`), 162
- `ckopn()` (in module `spiceypy.spiceypy`), 162
- `ckupf()` (in module `spiceypy.spiceypy`), 162
- `ckw01()` (in module `spiceypy.spiceypy`), 163
- `ckw02()` (in module `spiceypy.spiceypy`), 163
- `ckw03()` (in module `spiceypy.spiceypy`), 163
- `ckw05()` (in module `spiceypy.spiceypy`), 164
- `cleard()` (in module `spiceypy.spiceypy`), 165
- `clight()` (in module `spiceypy.spiceypy`), 165
- `clpool()` (in module `spiceypy.spiceypy`), 165
- `cltext()` (in module `spiceypy.spiceypy`), 165
- `cmprss()` (in module `spiceypy.spiceypy`), 165
- `cnames()` (`spiceypy.utils.support_types.SpiceEKSegSum` property), 306
- `cnmfrm()` (in module `spiceypy.spiceypy`), 165
- `colmax()` (`spiceypy.utils.support_types.SpiceDSKDescr` property), 305
- `colmin()` (`spiceypy.utils.support_types.SpiceDSKDescr` property), 305
- `co2max()` (`spiceypy.utils.support_types.SpiceDSKDescr` property), 305
- `co2min()` (`spiceypy.utils.support_types.SpiceDSKDescr` property), 305
- `co3max()` (`spiceypy.utils.support_types.SpiceDSKDescr` property), 305
- `co3min()` (`spiceypy.utils.support_types.SpiceDSKDescr` property), 305
- `conics()` (in module `spiceypy.spiceypy`), 166
- `constant()` (`spiceypy.utils.support_types.Plane` property), 303
- `convrt()` (in module `spiceypy.spiceypy`), 166
- `copy()` (in module `spiceypy.spiceypy`), 166
- `corpar()` (`spiceypy.utils.support_types.SpiceDSKDescr` property), 305
- `corsys()` (`spiceypy.utils.support_types.SpiceDSKDescr` property), 305
- `cpos()` (in module `spiceypy.spiceypy`), 166
- `cpostr()` (in module `spiceypy.spiceypy`), 167
- `csize()` (`spiceypy.utils.support_types.SpiceDLADescr` property), 305
- `CTRLBLOCK` (`spiceypy.utils.support_types.SpiceCell` attribute), 304
- `cvpool()` (in module `spiceypy.spiceypy`), 167
- `cyllat()` (in module `spiceypy.spiceypy`), 167
- `cylrec()` (in module `spiceypy.spiceypy`), 167
- `cylsph()` (in module `spiceypy.spiceypy`), 168
- D**
- `dafac()` (in module `spiceypy.spiceypy`), 168
- `dafbbs()` (in module `spiceypy.spiceypy`), 168
- `dafbfs()` (in module `spiceypy.spiceypy`), 168
- `dafcfs()` (in module `spiceypy.spiceypy`), 168
- `dafcs()` (in module `spiceypy.spiceypy`), 168
- `dafdc()` (in module `spiceypy.spiceypy`), 169
- `dafec()` (in module `spiceypy.spiceypy`), 169
- `daffna()` (in module `spiceypy.spiceypy`), 169
- `daffpa()` (in module `spiceypy.spiceypy`), 169
- `dafgda()` (in module `spiceypy.spiceypy`), 169
- `dafgh()` (in module `spiceypy.spiceypy`), 169
- `dafgn()` (in module `spiceypy.spiceypy`), 170
- `dafgs()` (in module `spiceypy.spiceypy`), 170
- `dafgsr()` (in module `spiceypy.spiceypy`), 170
- `dafopr()` (in module `spiceypy.spiceypy`), 170
- `dafopw()` (in module `spiceypy.spiceypy`), 170
- `dafps()` (in module `spiceypy.spiceypy`), 171
- `dafnda()` (in module `spiceypy.spiceypy`), 171
- `dafndr()` (in module `spiceypy.spiceypy`), 171
- `dafns()` (in module `spiceypy.spiceypy`), 171
- `dafus()` (in module `spiceypy.spiceypy`), 171
- `dasac()` (in module `spiceypy.spiceypy`), 172
- `dascls()` (in module `spiceypy.spiceypy`), 172
- `dasdc()` (in module `spiceypy.spiceypy`), 172
- `dasec()` (in module `spiceypy.spiceypy`), 172
- `dashfn()` (in module `spiceypy.spiceypy`), 172
- `dasonw()` (in module `spiceypy.spiceypy`), 173
- `dasopr()` (in module `spiceypy.spiceypy`), 173
- `dasopw()` (in module `spiceypy.spiceypy`), 173
- `dasrfr()` (in module `spiceypy.spiceypy`), 173
- `data` (`spiceypy.utils.support_types.SpiceCell` attribute), 304
- `DataType` (class in `spiceypy.utils.support_types`), 302
- `DATATYPES_ENUM` (`spiceypy.utils.support_types.SpiceCell` attribute), 304
- `DATATYPES_GET` (`spiceypy.utils.support_types.SpiceCell` attribute), 304
- `datetime2et()` (in module `spiceypy.spiceypy`), 173
- `dbase()` (`spiceypy.utils.support_types.SpiceDLADescr` property), 305
- `dclass()` (`spiceypy.utils.support_types.SpiceDSKDescr` property), 305
- `dcylldr()` (in module `spiceypy.spiceypy`), 174
- `deltet()` (in module `spiceypy.spiceypy`), 174
- `det()` (in module `spiceypy.spiceypy`), 174
- `dgeodr()` (in module `spiceypy.spiceypy`), 174
- `diags2()` (in module `spiceypy.spiceypy`), 175
- `diff()` (in module `spiceypy.spiceypy`), 175
- `dlabbs()` (in module `spiceypy.spiceypy`), 175
- `dlabfs()` (in module `spiceypy.spiceypy`), 175
- `dlafns()` (in module `spiceypy.spiceypy`), 175
- `dlafps()` (in module `spiceypy.spiceypy`), 175
- `dlatdr()` (in module `spiceypy.spiceypy`), 176
- `double()` (`spiceypy.utils.support_types.SpiceCell` class method), 304

- DoubleArrayType (class *spiceypy.utils.support_types*), 302
 DoubleMatrixType (class *spiceypy.utils.support_types*), 302
 DP (*spiceypy.utils.support_types.DataType* attribute), 302
 dp2hx () (in module *spiceypy.spiceypy*), 176
 dpgrdr () (in module *spiceypy.spiceypy*), 176
 dpmax () (in module *spiceypy.spiceypy*), 176
 dpmin () (in module *spiceypy.spiceypy*), 177
 dpr () (in module *spiceypy.spiceypy*), 177
 drdcyl () (in module *spiceypy.spiceypy*), 177
 drdgeo () (in module *spiceypy.spiceypy*), 177
 drdlat () (in module *spiceypy.spiceypy*), 177
 drdpgr () (in module *spiceypy.spiceypy*), 178
 drdsph () (in module *spiceypy.spiceypy*), 178
 dsize () (*spiceypy.utils.support_types.SpiceDLADescr* property), 305
 dskb02 () (in module *spiceypy.spiceypy*), 178
 dskcls () (in module *spiceypy.spiceypy*), 178
 dskd02 () (in module *spiceypy.spiceypy*), 179
 dskgd () (in module *spiceypy.spiceypy*), 179
 dskgtl () (in module *spiceypy.spiceypy*), 179
 dski02 () (in module *spiceypy.spiceypy*), 179
 dskmi2 () (in module *spiceypy.spiceypy*), 180
 dskn02 () (in module *spiceypy.spiceypy*), 180
 dskobj () (in module *spiceypy.spiceypy*), 180
 dskopn () (in module *spiceypy.spiceypy*), 180
 dskp02 () (in module *spiceypy.spiceypy*), 181
 dskrb2 () (in module *spiceypy.spiceypy*), 181
 dsksrfl () (in module *spiceypy.spiceypy*), 181
 dskstl () (in module *spiceypy.spiceypy*), 181
 dskv02 () (in module *spiceypy.spiceypy*), 182
 dskw02 () (in module *spiceypy.spiceypy*), 182
 dskx02 () (in module *spiceypy.spiceypy*), 183
 dskxsi () (in module *spiceypy.spiceypy*), 183
 dskxv () (in module *spiceypy.spiceypy*), 183
 dskz02 () (in module *spiceypy.spiceypy*), 184
 dsphdr () (in module *spiceypy.spiceypy*), 184
 dtpool () (in module *spiceypy.spiceypy*), 184
 dtype (*spiceypy.utils.support_types.SpiceCell* attribute), 304
 dtype () (*spiceypy.utils.support_types.SpiceDSKDescr* property), 305
 dtype () (*spiceypy.utils.support_types.SpiceEKAttDsc* property), 305
 ducrssl () (in module *spiceypy.spiceypy*), 184
 dvcrssl () (in module *spiceypy.spiceypy*), 185
 dvdot () (in module *spiceypy.spiceypy*), 185
 dvhat () (in module *spiceypy.spiceypy*), 185
 dvnorm () (in module *spiceypy.spiceypy*), 185
 dvpool () (in module *spiceypy.spiceypy*), 185
 dvsep () (in module *spiceypy.spiceypy*), 185
- E**
 edlimb () (in module *spiceypy.spiceypy*), 186
 edterm () (in module *spiceypy.spiceypy*), 186
 ekacec () (in module *spiceypy.spiceypy*), 186
 ekaced () (in module *spiceypy.spiceypy*), 187
 ekacei () (in module *spiceypy.spiceypy*), 187
 ekacalc () (in module *spiceypy.spiceypy*), 187
 ekacld () (in module *spiceypy.spiceypy*), 188
 ekaccli () (in module *spiceypy.spiceypy*), 188
 ekappr () (in module *spiceypy.spiceypy*), 188
 ekbseg () (in module *spiceypy.spiceypy*), 189
 ekccnt () (in module *spiceypy.spiceypy*), 189
 ekcii () (in module *spiceypy.spiceypy*), 189
 ekcls () (in module *spiceypy.spiceypy*), 189
 ekdelr () (in module *spiceypy.spiceypy*), 190
 ekffld () (in module *spiceypy.spiceypy*), 190
 ekfind () (in module *spiceypy.spiceypy*), 190
 ekgc () (in module *spiceypy.spiceypy*), 190
 ekgd () (in module *spiceypy.spiceypy*), 190
 ekgi () (in module *spiceypy.spiceypy*), 191
 ekifld () (in module *spiceypy.spiceypy*), 191
 ekinsr () (in module *spiceypy.spiceypy*), 191
 eklef () (in module *spiceypy.spiceypy*), 192
 eknelt () (in module *spiceypy.spiceypy*), 192
 eknseg () (in module *spiceypy.spiceypy*), 192
 ekntab () (in module *spiceypy.spiceypy*), 192
 ekopn () (in module *spiceypy.spiceypy*), 192
 ekopr () (in module *spiceypy.spiceypy*), 192
 ekops () (in module *spiceypy.spiceypy*), 193
 ekopw () (in module *spiceypy.spiceypy*), 193
 ekpsel () (in module *spiceypy.spiceypy*), 193
 ekrcec () (in module *spiceypy.spiceypy*), 193
 ekrced () (in module *spiceypy.spiceypy*), 194
 ekrcei () (in module *spiceypy.spiceypy*), 194
 ekssum () (in module *spiceypy.spiceypy*), 194
 ektnam () (in module *spiceypy.spiceypy*), 194
 ekucec () (in module *spiceypy.spiceypy*), 195
 ekuced () (in module *spiceypy.spiceypy*), 195
 ekucei () (in module *spiceypy.spiceypy*), 195
 ekuef () (in module *spiceypy.spiceypy*), 196
 el2cgv () (in module *spiceypy.spiceypy*), 196
 elemc () (in module *spiceypy.spiceypy*), 196
 elemd () (in module *spiceypy.spiceypy*), 196
 elemi () (in module *spiceypy.spiceypy*), 196
 Ellipse (class in *spiceypy.utils.support_types*), 303
 empty_char_array () (in module *spiceypy.utils.support_types*), 306
 empty_double_matrix () (in module *spiceypy.utils.support_types*), 306
 empty_double_vector () (in module *spiceypy.utils.support_types*), 306
 empty_int_matrix () (in module *spiceypy.utils.support_types*), 306

`empty_int_vector()` (in module `spiceypy.support_types`), 306

`empty_spice_ek_data_type_vector()` (in module `spiceypy.support_types`), 306

`empty_spice_ek_expr_class_vector()` (in module `spiceypy.support_types`), 306

`eqncpv()` (in module `spiceypy.spiceypy`), 197

`eqstr()` (in module `spiceypy.spiceypy`), 197

`erract()` (in module `spiceypy.spiceypy`), 197

`errch()` (in module `spiceypy.spiceypy`), 197

`errdev()` (in module `spiceypy.spiceypy`), 198

`errdp()` (in module `spiceypy.spiceypy`), 198

`errint()` (in module `spiceypy.spiceypy`), 198

`errprt()` (in module `spiceypy.spiceypy`), 198

`esrchc()` (in module `spiceypy.spiceypy`), 199

`et2lst()` (in module `spiceypy.spiceypy`), 199

`et2utc()` (in module `spiceypy.spiceypy`), 199

`etcal()` (in module `spiceypy.spiceypy`), 199

`eul2m()` (in module `spiceypy.spiceypy`), 200

`eul2xf()` (in module `spiceypy.spiceypy`), 200

`ev2lin()` (in module `spiceypy.spiceypy`), 200

`exists()` (in module `spiceypy.spiceypy`), 201

`expool()` (in module `spiceypy.spiceypy`), 201

F

`failed()` (in module `spiceypy.spiceypy`), 201

`fn2lun()` (in module `spiceypy.spiceypy`), 201

`found_check()` (in module `spiceypy.spiceypy`), 201

`found_check_off()` (in module `spiceypy.spiceypy`), 201

`found_check_on()` (in module `spiceypy.spiceypy`), 202

`fovray()` (in module `spiceypy.spiceypy`), 202

`fovtrg()` (in module `spiceypy.spiceypy`), 202

`frame()` (in module `spiceypy.spiceypy`), 202

`frinfo()` (in module `spiceypy.spiceypy`), 202

`frmcde()` (`spiceypy.utils.support_types.SpiceDSKDescr` property), 305

`frmnam()` (in module `spiceypy.spiceypy`), 203

`from_array()` (`spiceypy.utils.support_types.DoubleArrayType` method), 302

`from_array()` (`spiceypy.utils.support_types.IntArrayType` method), 303

`from_list()` (`spiceypy.utils.support_types.DoubleArrayType` method), 302

`from_list()` (`spiceypy.utils.support_types.DoubleMatrixType` method), 302

`from_list()` (`spiceypy.utils.support_types.IntArrayType` method), 303

`from_list()` (`spiceypy.utils.support_types.IntMatrixType` method), 303

`from_matrix()` (`spiceypy.utils.support_types.DoubleMatrixType` method), 302

`from_matrix()` (`spiceypy.utils.support_types.IntMatrixType` method), 303

`from_ndarray()` (`spiceypy.utils.support_types.DoubleArrayType` method), 302

`from_ndarray()` (`spiceypy.utils.support_types.DoubleMatrixType` method), 302

`from_ndarray()` (`spiceypy.utils.support_types.IntArrayType` method), 303

`from_ndarray()` (`spiceypy.utils.support_types.IntMatrixType` method), 303

`from_param()` (`spiceypy.utils.support_types.DoubleArrayType` method), 302

`from_param()` (`spiceypy.utils.support_types.DoubleMatrixType` method), 303

`from_param()` (`spiceypy.utils.support_types.IntArrayType` method), 303

`from_param()` (`spiceypy.utils.support_types.IntMatrixType` method), 303

`from_tuple()` (`spiceypy.utils.support_types.DoubleArrayType` method), 302

`from_tuple()` (`spiceypy.utils.support_types.DoubleMatrixType` method), 303

`from_tuple()` (`spiceypy.utils.support_types.IntArrayType` method), 303

`from_tuple()` (`spiceypy.utils.support_types.IntMatrixType` method), 303

`ftncls()` (in module `spiceypy.spiceypy`), 203

`furnsh()` (in module `spiceypy.spiceypy`), 203

`fdwptr()` (`spiceypy.utils.support_types.SpiceDLADescr` property), 305

G

`gcpool()` (in module `spiceypy.spiceypy`), 203

`gdpool()` (in module `spiceypy.spiceypy`), 203

`georec()` (in module `spiceypy.spiceypy`), 204

`get_found_catch_state()` (in module `spiceypy.spiceypy`), 204

`getelm()` (in module `spiceypy.spiceypy`), 204

`getfat()` (in module `spiceypy.spiceypy`), 204

`getfov()` (in module `spiceypy.spiceypy`), 204

`getmsg()` (in module `spiceypy.spiceypy`), 205

`gfbail()` (in module `spiceypy.spiceypy`), 205

`gfclrh()` (in module `spiceypy.spiceypy`), 205

`gfclst()` (in module `spiceypy.spiceypy`), 205

`gfevnt()` (in module `spiceypy.spiceypy`), 206

`gfmove()` (in module `spiceypy.spiceypy`), 206

`gfilum()` (in module `spiceypy.spiceypy`), 207

`gfinth()` (in module `spiceypy.spiceypy`), 208

`gfocce()` (in module `spiceypy.spiceypy`), 208

`gfoclt()` (in module `spiceypy.spiceypy`), 209

`gfpa()` (in module `spiceypy.spiceypy`), 209

`gfprtc()` (in module `spiceypy.spiceypy`), 209

`gfrefn()` (in module `spiceypy.spiceypy`), 210

`gfrepf()` (in module `spiceypy.spiceypy`), 210

[gfrepi\(\)](#) (in module *spiceypy.spiceypy*), 210
[gfrepu\(\)](#) (in module *spiceypy.spiceypy*), 211
[gfrfov\(\)](#) (in module *spiceypy.spiceypy*), 211
[gfrfrr\(\)](#) (in module *spiceypy.spiceypy*), 211
[gfsep\(\)](#) (in module *spiceypy.spiceypy*), 212
[gfsntc\(\)](#) (in module *spiceypy.spiceypy*), 212
[gfsstp\(\)](#) (in module *spiceypy.spiceypy*), 213
[gfstep\(\)](#) (in module *spiceypy.spiceypy*), 213
[gfstol\(\)](#) (in module *spiceypy.spiceypy*), 213
[gfsuubc\(\)](#) (in module *spiceypy.spiceypy*), 213
[gftfov\(\)](#) (in module *spiceypy.spiceypy*), 214
[gfudb\(\)](#) (in module *spiceypy.spiceypy*), 214
[gfuds\(\)](#) (in module *spiceypy.spiceypy*), 214
[gipool\(\)](#) (in module *spiceypy.spiceypy*), 215
[gnpool\(\)](#) (in module *spiceypy.spiceypy*), 215

H

[halfpi\(\)](#) (in module *spiceypy.spiceypy*), 215
[hrmint\(\)](#) (in module *spiceypy.spiceypy*), 216
[hx2dp\(\)](#) (in module *spiceypy.spiceypy*), 216

I

[ibase\(\)](#) (*spiceypy.utils.support_types.SpiceDLADescr* property), 305
[ident\(\)](#) (in module *spiceypy.spiceypy*), 216
[illum\(\)](#) (in module *spiceypy.spiceypy*), 216
[illumf\(\)](#) (in module *spiceypy.spiceypy*), 216
[illumg\(\)](#) (in module *spiceypy.spiceypy*), 217
[ilumin\(\)](#) (in module *spiceypy.spiceypy*), 217
[indexd\(\)](#) (*spiceypy.utils.support_types.SpiceEKAttDsc* property), 305
[inedpl\(\)](#) (in module *spiceypy.spiceypy*), 218
[inelpl\(\)](#) (in module *spiceypy.spiceypy*), 218
[init\(\)](#) (*spiceypy.utils.support_types.SpiceCell* attribute), 304
[inrypl\(\)](#) (in module *spiceypy.spiceypy*), 218
[insrtc\(\)](#) (in module *spiceypy.spiceypy*), 219
[insrtd\(\)](#) (in module *spiceypy.spiceypy*), 219
[insrti\(\)](#) (in module *spiceypy.spiceypy*), 219
[INT](#) (*spiceypy.utils.support_types.DataType* attribute), 302
[IntArrayType](#) (class in *spiceypy.utils.support_types*), 303
[integer\(\)](#) (*spiceypy.utils.support_types.SpiceCell* class method), 304
[inter\(\)](#) (in module *spiceypy.spiceypy*), 219
[IntMatrixType](#) (class in *spiceypy.utils.support_types*), 303
[intmax\(\)](#) (in module *spiceypy.spiceypy*), 219
[intmin\(\)](#) (in module *spiceypy.spiceypy*), 219
[invert\(\)](#) (in module *spiceypy.spiceypy*), 220
[invort\(\)](#) (in module *spiceypy.spiceypy*), 220
[is_bool\(\)](#) (*spiceypy.utils.support_types.SpiceCell* method), 304

[is_char\(\)](#) (*spiceypy.utils.support_types.SpiceCell* method), 304
[is_double\(\)](#) (*spiceypy.utils.support_types.SpiceCell* method), 304
[is_int\(\)](#) (*spiceypy.utils.support_types.SpiceCell* method), 304
[is_iterable\(\)](#) (in module *spiceypy.utils.support_types*), 306
[is_set\(\)](#) (*spiceypy.utils.support_types.SpiceCell* method), 304
[is_time\(\)](#) (*spiceypy.utils.support_types.SpiceCell* method), 304
[isize\(\)](#) (*spiceypy.utils.support_types.SpiceDLADescr* property), 305
[isordv\(\)](#) (in module *spiceypy.spiceypy*), 220
[isrchc\(\)](#) (in module *spiceypy.spiceypy*), 220
[isrchd\(\)](#) (in module *spiceypy.spiceypy*), 220
[isrchi\(\)](#) (in module *spiceypy.spiceypy*), 221
[isrot\(\)](#) (in module *spiceypy.spiceypy*), 221
[isSet](#) (*spiceypy.utils.support_types.SpiceCell* attribute), 304
[iswhsp\(\)](#) (in module *spiceypy.spiceypy*), 221

J

[j1900\(\)](#) (in module *spiceypy.spiceypy*), 221
[j1950\(\)](#) (in module *spiceypy.spiceypy*), 222
[j2000\(\)](#) (in module *spiceypy.spiceypy*), 222
[j2100\(\)](#) (in module *spiceypy.spiceypy*), 222
[jyear\(\)](#) (in module *spiceypy.spiceypy*), 222

K

[kclear\(\)](#) (in module *spiceypy.spiceypy*), 222
[kdata\(\)](#) (in module *spiceypy.spiceypy*), 222
[kinfo\(\)](#) (in module *spiceypy.spiceypy*), 222
[kplfrm\(\)](#) (in module *spiceypy.spiceypy*), 223
[ktotal\(\)](#) (in module *spiceypy.spiceypy*), 223
[kxtrct\(\)](#) (in module *spiceypy.spiceypy*), 223

L

[lastnb\(\)](#) (in module *spiceypy.spiceypy*), 223
[latcyl\(\)](#) (in module *spiceypy.spiceypy*), 224
[latrec\(\)](#) (in module *spiceypy.spiceypy*), 224
[latsph\(\)](#) (in module *spiceypy.spiceypy*), 224
[latsrf\(\)](#) (in module *spiceypy.spiceypy*), 224
[lcase\(\)](#) (in module *spiceypy.spiceypy*), 225
[ldpool\(\)](#) (in module *spiceypy.spiceypy*), 225
[length](#) (*spiceypy.utils.support_types.SpiceCell* attribute), 304
[lgrind\(\)](#) (in module *spiceypy.spiceypy*), 225
[limbpt\(\)](#) (in module *spiceypy.spiceypy*), 225
[list_to_char_array\(\)](#) (in module *spiceypy.utils.support_types*), 306
[list_to_char_array_ptr\(\)](#) (in module *spiceypy.utils.support_types*), 306

lmpool () (in module *spiceypy.spiceypy*), 226
 lparse () (in module *spiceypy.spiceypy*), 226
 lparsm () (in module *spiceypy.spiceypy*), 226
 lparss () (in module *spiceypy.spiceypy*), 227
 lspcn () (in module *spiceypy.spiceypy*), 227
 lstlec () (in module *spiceypy.spiceypy*), 227
 lstled () (in module *spiceypy.spiceypy*), 227
 lstlei () (in module *spiceypy.spiceypy*), 228
 lstltc () (in module *spiceypy.spiceypy*), 228
 lstltd () (in module *spiceypy.spiceypy*), 228
 lstlti () (in module *spiceypy.spiceypy*), 228
 ltime () (in module *spiceypy.spiceypy*), 229
 lx4dec () (in module *spiceypy.spiceypy*), 229
 lx4num () (in module *spiceypy.spiceypy*), 229
 lx4sgn () (in module *spiceypy.spiceypy*), 229
 lx4uns () (in module *spiceypy.spiceypy*), 230
 lxqstr () (in module *spiceypy.spiceypy*), 230

M

m2eul () (in module *spiceypy.spiceypy*), 230
 m2q () (in module *spiceypy.spiceypy*), 230
 matchi () (in module *spiceypy.spiceypy*), 231
 matchw () (in module *spiceypy.spiceypy*), 231
 mequ () (in module *spiceypy.spiceypy*), 231
 mequg () (in module *spiceypy.spiceypy*), 231
 minCharLen (*spiceypy.utils.support_types.SpiceCell*
 attribute), 304
 mtxm () (in module *spiceypy.spiceypy*), 232
 mtxmg () (in module *spiceypy.spiceypy*), 232
 mtxv () (in module *spiceypy.spiceypy*), 232
 mtxvg () (in module *spiceypy.spiceypy*), 232
 mxm () (in module *spiceypy.spiceypy*), 233
 mxmg () (in module *spiceypy.spiceypy*), 233
 mxmt () (in module *spiceypy.spiceypy*), 233
 mxmtg () (in module *spiceypy.spiceypy*), 233
 mxv () (in module *spiceypy.spiceypy*), 234
 mxvg () (in module *spiceypy.spiceypy*), 234

N

namfrm () (in module *spiceypy.spiceypy*), 234
 ncols () (*spiceypy.utils.support_types.SpiceEKSegSum*
 property), 306
 ncpos () (in module *spiceypy.spiceypy*), 234
 ncposr () (in module *spiceypy.spiceypy*), 234
 nearpt () (in module *spiceypy.spiceypy*), 235
 no_found_check () (in module *spiceypy.spiceypy*),
 235
 normal () (*spiceypy.utils.support_types.Plane* *prop-*
 erty), 303
 npedln () (in module *spiceypy.spiceypy*), 235
 npelpt () (in module *spiceypy.spiceypy*), 236
 nplnpt () (in module *spiceypy.spiceypy*), 236
 nrows () (*spiceypy.utils.support_types.SpiceEKSegSum*
 property), 306

nullok () (*spiceypy.utils.support_types.SpiceEKAttDsc*
 property), 305
 nvc2pl () (in module *spiceypy.spiceypy*), 236
 nvp2pl () (in module *spiceypy.spiceypy*), 236

O

occult () (in module *spiceypy.spiceypy*), 237
 ordc () (in module *spiceypy.spiceypy*), 237
 ordd () (in module *spiceypy.spiceypy*), 237
 orderc () (in module *spiceypy.spiceypy*), 237
 orderd () (in module *spiceypy.spiceypy*), 238
 orderi () (in module *spiceypy.spiceypy*), 238
 ordi () (in module *spiceypy.spiceypy*), 238
 oscelt () (in module *spiceypy.spiceypy*), 238
 oscltx () (in module *spiceypy.spiceypy*), 239

P

pckcls () (in module *spiceypy.spiceypy*), 239
 pckcov () (in module *spiceypy.spiceypy*), 239
 pckfrm () (in module *spiceypy.spiceypy*), 239
 pcklof () (in module *spiceypy.spiceypy*), 239
 pckopn () (in module *spiceypy.spiceypy*), 240
 pckuof () (in module *spiceypy.spiceypy*), 240
 pckw02 () (in module *spiceypy.spiceypy*), 240
 pcpool () (in module *spiceypy.spiceypy*), 240
 pdpool () (in module *spiceypy.spiceypy*), 241
 pgrrec () (in module *spiceypy.spiceypy*), 241
 phaseq () (in module *spiceypy.spiceypy*), 241
 pi () (in module *spiceypy.spiceypy*), 242
 pipool () (in module *spiceypy.spiceypy*), 242
 pjelp1 () (in module *spiceypy.spiceypy*), 242
 pl2nvc () (in module *spiceypy.spiceypy*), 242
 pl2nvp () (in module *spiceypy.spiceypy*), 242
 pl2psv () (in module *spiceypy.spiceypy*), 242
 Plane (*class in spiceypy.utils.support_types*), 303
 pltar () (in module *spiceypy.spiceypy*), 243
 pltxp () (in module *spiceypy.spiceypy*), 243
 pltnp () (in module *spiceypy.spiceypy*), 243
 pltnrm () (in module *spiceypy.spiceypy*), 243
 pltvol () (in module *spiceypy.spiceypy*), 244
 polyds () (in module *spiceypy.spiceypy*), 244
 pos () (in module *spiceypy.spiceypy*), 244
 posr () (in module *spiceypy.spiceypy*), 244
 prop2b () (in module *spiceypy.spiceypy*), 244
 prsdp () (in module *spiceypy.spiceypy*), 245
 prsint () (in module *spiceypy.spiceypy*), 245
 psv2pl () (in module *spiceypy.spiceypy*), 245
 pxform () (in module *spiceypy.spiceypy*), 245
 pxfrm2 () (in module *spiceypy.spiceypy*), 246

Q

q2m () (in module *spiceypy.spiceypy*), 246
 qcktrc () (in module *spiceypy.spiceypy*), 246
 qdq2av () (in module *spiceypy.spiceypy*), 246

`qxq()` (in module `spicepy.spicepy`), 246

R

`radrec()` (in module `spicepy.spicepy`), 247
`rav2xf()` (in module `spicepy.spicepy`), 247
`raxisa()` (in module `spicepy.spicepy`), 247
`rdtext()` (in module `spicepy.spicepy`), 247
`reccyl()` (in module `spicepy.spicepy`), 247
`recgeo()` (in module `spicepy.spicepy`), 248
`reclat()` (in module `spicepy.spicepy`), 248
`recpgr()` (in module `spicepy.spicepy`), 248
`recrad()` (in module `spicepy.spicepy`), 248
`recsph()` (in module `spicepy.spicepy`), 248
`removc()` (in module `spicepy.spicepy`), 249
`removd()` (in module `spicepy.spicepy`), 249
`removi()` (in module `spicepy.spicepy`), 249
`reordc()` (in module `spicepy.spicepy`), 249
`reordd()` (in module `spicepy.spicepy`), 249
`reordi()` (in module `spicepy.spicepy`), 250
`reordl()` (in module `spicepy.spicepy`), 250
`repmc()` (in module `spicepy.spicepy`), 250
`repmct()` (in module `spicepy.spicepy`), 250
`repmf()` (in module `spicepy.spicepy`), 251
`repmi()` (in module `spicepy.spicepy`), 251
`repmot()` (in module `spicepy.spicepy`), 252
`reset()` (in module `spicepy.spicepy`), 252
`reset()` (`spicepy.utils.support_types.SpiceCell` method), 304
`return_c()` (in module `spicepy.spicepy`), 252
`rotate()` (in module `spicepy.spicepy`), 252
`rotmat()` (in module `spicepy.spicepy`), 252
`rotvec()` (in module `spicepy.spicepy`), 253
`rpd()` (in module `spicepy.spicepy`), 253
`rquad()` (in module `spicepy.spicepy`), 253

S

`S18TP0` (`spicepy.utils.support_types.SpiceSPK18Subtype` attribute), 306
`S18TP1` (`spicepy.utils.support_types.SpiceSPK18Subtype` attribute), 306
`s_dla_p` (in module `spicepy.utils.libspicehelper`), 307
`s_dsk_p` (in module `spicepy.utils.libspicehelper`), 307
`s_eka_p` (in module `spicepy.utils.libspicehelper`), 307
`s_eks_p` (in module `spicepy.utils.libspicehelper`), 307
`s_elip_p` (in module `spicepy.utils.libspicehelper`), 307
`s_plan_p` (in module `spicepy.utils.libspicehelper`), 307
`saelv()` (in module `spicepy.spicepy`), 253
`scard()` (in module `spicepy.spicepy`), 254
`scdec()` (in module `spicepy.spicepy`), 254
`sce2c()` (in module `spicepy.spicepy`), 254
`sce2s()` (in module `spicepy.spicepy`), 254
`sce2t()` (in module `spicepy.spicepy`), 255
`scncd()` (in module `spicepy.spicepy`), 255
`scfmt()` (in module `spicepy.spicepy`), 255
`scpart()` (in module `spicepy.spicepy`), 255
`scs2e()` (in module `spicepy.spicepy`), 255
`sct2e()` (in module `spicepy.spicepy`), 256
`sctiks()` (in module `spicepy.spicepy`), 256
`sdiff()` (in module `spicepy.spicepy`), 256
`semi_major()` (`spicepy.utils.support_types.Ellipse` property), 303
`semi_minor()` (`spicepy.utils.support_types.Ellipse` property), 303
`set_c()` (in module `spicepy.spicepy`), 256
`setmsg()` (in module `spicepy.spicepy`), 257
`shellc()` (in module `spicepy.spicepy`), 257
`shelld()` (in module `spicepy.spicepy`), 257
`shelli()` (in module `spicepy.spicepy`), 257
`sigerr()` (in module `spicepy.spicepy`), 257
`sincpt()` (in module `spicepy.spicepy`), 258
`size` (`spicepy.utils.support_types.SpiceCell` attribute), 304
`size()` (in module `spicepy.spicepy`), 258
`size()` (`spicepy.utils.support_types.SpiceEKAttDsc` property), 305
`spd()` (in module `spicepy.spicepy`), 258
`sphcyl()` (in module `spicepy.spicepy`), 258
`sphlat()` (in module `spicepy.spicepy`), 259
`sphrec()` (in module `spicepy.spicepy`), 259
`SPICE_BOOL` (`spicepy.utils.support_types.DataType` attribute), 302
`SPICE_BOOL` (`spicepy.utils.support_types.SpiceEKDataType` attribute), 305
`SPICE_CHR` (`spicepy.utils.support_types.DataType` attribute), 302
`SPICE_CHR` (`spicepy.utils.support_types.SpiceEKDataType` attribute), 305
`SPICE_DP` (`spicepy.utils.support_types.DataType` attribute), 302
`SPICE_DP` (`spicepy.utils.support_types.SpiceEKDataType` attribute), 305
`SPICE_EK_EXP_COL` (`spicepy.utils.support_types.SpiceEKExprClass` attribute), 306
`SPICE_EK_EXP_EXPR` (`spicepy.utils.support_types.SpiceEKExprClass` attribute), 306
`SPICE_EK_EXP_FUNC` (`spicepy.utils.support_types.SpiceEKExprClass` attribute), 306
`spice_error_check()` (in module `spicepy.spicepy`), 259
`spice_found_exception_thrower()` (in module `spicepy.spicepy`), 259
`SPICE_INT` (`spicepy.utils.support_types.DataType` attribute), 302
`SPICE_INT` (`spicepy.utils.support_types.SpiceEKDataType` attribute), 306

SPICE_TIME (*spiceypy.utils.support_types.DataType*
 attribute), 302
 SPICE_TIME (*spiceypy.utils.support_types.SpiceEKDataType*
 attribute), 306
 SPICEBOOL_CELL () (in module
 spiceypy.utils.support_types), 303
 SpiceCell (*class* in *spiceypy.utils.support_types*), 303
 SPICECHAR_CELL () (in module
 spiceypy.utils.support_types), 303
 SpiceDLADescr (*class* in
 spiceypy.utils.support_types), 304
 SPICEDOUBLE_CELL () (in module
 spiceypy.utils.support_types), 303
 SpiceDSKDescr (*class* in
 spiceypy.utils.support_types), 305
 SpiceEKAttDsc (*class* in
 spiceypy.utils.support_types), 305
 SpiceEKDataType (*class* in
 spiceypy.utils.support_types), 305
 SpiceEKExprClass (*class* in
 spiceypy.utils.support_types), 306
 SpiceEKSegSum (*class* in
 spiceypy.utils.support_types), 306
 SPICEINT_CELL () (in module
 spiceypy.utils.support_types), 303
 SpiceSPK18Subtype (*class* in
 spiceypy.utils.support_types), 306
 SPICETIME_CELL () (in module
 spiceypy.utils.support_types), 303
 SpiceyError, 306
 spiceypy.spiceypy (*module*), 153
 spiceypy.utils.libspicehelper (*module*),
 307
 spiceypy.utils.support_types (*module*), 301
 spk14a () (in module *spiceypy.spiceypy*), 259
 spk14b () (in module *spiceypy.spiceypy*), 259
 spk14e () (in module *spiceypy.spiceypy*), 260
 spkacs () (in module *spiceypy.spiceypy*), 260
 spkapa () (in module *spiceypy.spiceypy*), 260
 spkapp () (in module *spiceypy.spiceypy*), 261
 spkaps () (in module *spiceypy.spiceypy*), 261
 spkcls () (in module *spiceypy.spiceypy*), 261
 spkcov () (in module *spiceypy.spiceypy*), 261
 spkcpo () (in module *spiceypy.spiceypy*), 262
 spkcpt () (in module *spiceypy.spiceypy*), 262
 spkcvo () (in module *spiceypy.spiceypy*), 263
 spkcvt () (in module *spiceypy.spiceypy*), 263
 spkez () (in module *spiceypy.spiceypy*), 263
 spkezp () (in module *spiceypy.spiceypy*), 264
 spkezz () (in module *spiceypy.spiceypy*), 264
 spkgeo () (in module *spiceypy.spiceypy*), 264
 spkgps () (in module *spiceypy.spiceypy*), 265
 spklaf () (in module *spiceypy.spiceypy*), 265
 spkltc () (in module *spiceypy.spiceypy*), 265
 spkobj () (in module *spiceypy.spiceypy*), 265
 spkopa () (in module *spiceypy.spiceypy*), 266
 spkopn () (in module *spiceypy.spiceypy*), 266
 spkpds () (in module *spiceypy.spiceypy*), 266
 spkpos () (in module *spiceypy.spiceypy*), 266
 spkpvn () (in module *spiceypy.spiceypy*), 267
 spksfs () (in module *spiceypy.spiceypy*), 267
 spkssb () (in module *spiceypy.spiceypy*), 267
 spksub () (in module *spiceypy.spiceypy*), 267
 spkuds () (in module *spiceypy.spiceypy*), 268
 spkuef () (in module *spiceypy.spiceypy*), 268
 spkw02 () (in module *spiceypy.spiceypy*), 268
 spkw03 () (in module *spiceypy.spiceypy*), 269
 spkw05 () (in module *spiceypy.spiceypy*), 269
 spkw08 () (in module *spiceypy.spiceypy*), 270
 spkw09 () (in module *spiceypy.spiceypy*), 270
 spkw10 () (in module *spiceypy.spiceypy*), 270
 spkw12 () (in module *spiceypy.spiceypy*), 271
 spkw13 () (in module *spiceypy.spiceypy*), 271
 spkw15 () (in module *spiceypy.spiceypy*), 272
 spkw17 () (in module *spiceypy.spiceypy*), 272
 spkw18 () (in module *spiceypy.spiceypy*), 273
 spkw20 () (in module *spiceypy.spiceypy*), 273
 srfc2s () (in module *spiceypy.spiceypy*), 274
 srfcss () (in module *spiceypy.spiceypy*), 274
 srfrnm () (in module *spiceypy.spiceypy*), 274
 srfrec () (in module *spiceypy.spiceypy*), 275
 srfs2c () (in module *spiceypy.spiceypy*), 275
 srfsc () (in module *spiceypy.spiceypy*), 275
 srfxpt () (in module *spiceypy.spiceypy*), 275
 ssize () (in module *spiceypy.spiceypy*), 276
 start () (*spiceypy.utils.support_types.SpiceDSKDescr*
 property), 305
 stelab () (in module *spiceypy.spiceypy*), 276
 stop () (*spiceypy.utils.support_types.SpiceDSKDescr*
 property), 305
 stpool () (in module *spiceypy.spiceypy*), 276
 str2et () (in module *spiceypy.spiceypy*), 277
 string_to_char_p () (in module
 spiceypy.utils.support_types), 307
 strlen () (*spiceypy.utils.support_types.SpiceEKAttDsc*
 property), 305
 subpnt () (in module *spiceypy.spiceypy*), 277
 subpt () (in module *spiceypy.spiceypy*), 277
 subslr () (in module *spiceypy.spiceypy*), 278
 subsol () (in module *spiceypy.spiceypy*), 278
 sumad () (in module *spiceypy.spiceypy*), 278
 sumai () (in module *spiceypy.spiceypy*), 279
 surface () (*spiceypy.utils.support_types.SpiceDSKDescr*
 property), 305
 surfnm () (in module *spiceypy.spiceypy*), 279
 surfpt () (in module *spiceypy.spiceypy*), 279
 surfpv () (in module *spiceypy.spiceypy*), 279
 swpool () (in module *spiceypy.spiceypy*), 280

[sxform\(\)](#) (in module *spiceypy.spiceypy*), 280
[szpool\(\)](#) (in module *spiceypy.spiceypy*), 280

T

[tabnam\(\)](#) (*spiceypy.utils.support_types.SpiceEKSegSum* property), 306
[termpt\(\)](#) (in module *spiceypy.spiceypy*), 280
[timdef\(\)](#) (in module *spiceypy.spiceypy*), 281
[TIME](#) (*spiceypy.utils.support_types.DataType* attribute), 302
[time\(\)](#) (*spiceypy.utils.support_types.SpiceCell* class method), 304
[timeout\(\)](#) (in module *spiceypy.spiceypy*), 281
[tipbod\(\)](#) (in module *spiceypy.spiceypy*), 281
[tisbod\(\)](#) (in module *spiceypy.spiceypy*), 282
[tkfram\(\)](#) (in module *spiceypy.spiceypy*), 282
[tkvrns\(\)](#) (in module *spiceypy.spiceypy*), 282
[to_double_matrix\(\)](#) (in module *spiceypy.utils.support_types*), 307
[to_double_vector\(\)](#) (in module *spiceypy.utils.support_types*), 307
[to_int_matrix\(\)](#) (in module *spiceypy.utils.support_types*), 307
[to_int_vector\(\)](#) (in module *spiceypy.utils.support_types*), 307
[to_python_string\(\)](#) (in module *spiceypy.utils.support_types*), 307
[tparse\(\)](#) (in module *spiceypy.spiceypy*), 282
[tpictr\(\)](#) (in module *spiceypy.spiceypy*), 283
[trace\(\)](#) (in module *spiceypy.spiceypy*), 283
[trcdep\(\)](#) (in module *spiceypy.spiceypy*), 283
[trcnam\(\)](#) (in module *spiceypy.spiceypy*), 283
[trcoff\(\)](#) (in module *spiceypy.spiceypy*), 283
[tsetyr\(\)](#) (in module *spiceypy.spiceypy*), 283
[twopi\(\)](#) (in module *spiceypy.spiceypy*), 284
[twovec\(\)](#) (in module *spiceypy.spiceypy*), 284
[txtopn\(\)](#) (in module *spiceypy.spiceypy*), 284
[tyear\(\)](#) (in module *spiceypy.spiceypy*), 284

U

[ucase\(\)](#) (in module *spiceypy.spiceypy*), 284
[ucrss\(\)](#) (in module *spiceypy.spiceypy*), 285
[uddc\(\)](#) (in module *spiceypy.spiceypy*), 285
[uddf\(\)](#) (in module *spiceypy.spiceypy*), 285
[udf\(\)](#) (in module *spiceypy.spiceypy*), 286
[union\(\)](#) (in module *spiceypy.spiceypy*), 286
[unitim\(\)](#) (in module *spiceypy.spiceypy*), 286
[unload\(\)](#) (in module *spiceypy.spiceypy*), 286
[unorm\(\)](#) (in module *spiceypy.spiceypy*), 286
[unormg\(\)](#) (in module *spiceypy.spiceypy*), 286
[utc2et\(\)](#) (in module *spiceypy.spiceypy*), 287

V

[vadd\(\)](#) (in module *spiceypy.spiceypy*), 287

[vaddg\(\)](#) (in module *spiceypy.spiceypy*), 287
[valid\(\)](#) (in module *spiceypy.spiceypy*), 287
[vcrss\(\)](#) (in module *spiceypy.spiceypy*), 287
[vdist\(\)](#) (in module *spiceypy.spiceypy*), 288
[vdistg\(\)](#) (in module *spiceypy.spiceypy*), 288
[vdot\(\)](#) (in module *spiceypy.spiceypy*), 288
[vdotg\(\)](#) (in module *spiceypy.spiceypy*), 288
[vequ\(\)](#) (in module *spiceypy.spiceypy*), 289
[veqg\(\)](#) (in module *spiceypy.spiceypy*), 289
[vhat\(\)](#) (in module *spiceypy.spiceypy*), 289
[vhatg\(\)](#) (in module *spiceypy.spiceypy*), 289
[vlcom\(\)](#) (in module *spiceypy.spiceypy*), 289
[vlcom3\(\)](#) (in module *spiceypy.spiceypy*), 290
[vlcomg\(\)](#) (in module *spiceypy.spiceypy*), 290
[vminug\(\)](#) (in module *spiceypy.spiceypy*), 290
[vminus\(\)](#) (in module *spiceypy.spiceypy*), 290
[vnorm\(\)](#) (in module *spiceypy.spiceypy*), 291
[vnormg\(\)](#) (in module *spiceypy.spiceypy*), 291
[vpack\(\)](#) (in module *spiceypy.spiceypy*), 291
[vperp\(\)](#) (in module *spiceypy.spiceypy*), 291
[vprjp\(\)](#) (in module *spiceypy.spiceypy*), 291
[vprjpi\(\)](#) (in module *spiceypy.spiceypy*), 292
[vproj\(\)](#) (in module *spiceypy.spiceypy*), 292
[vrel\(\)](#) (in module *spiceypy.spiceypy*), 292
[vrelg\(\)](#) (in module *spiceypy.spiceypy*), 292
[vrotrv\(\)](#) (in module *spiceypy.spiceypy*), 293
[vscl\(\)](#) (in module *spiceypy.spiceypy*), 293
[vsclg\(\)](#) (in module *spiceypy.spiceypy*), 293
[vsep\(\)](#) (in module *spiceypy.spiceypy*), 293
[vsepg\(\)](#) (in module *spiceypy.spiceypy*), 294
[vsub\(\)](#) (in module *spiceypy.spiceypy*), 294
[vsubg\(\)](#) (in module *spiceypy.spiceypy*), 294
[vtmv\(\)](#) (in module *spiceypy.spiceypy*), 294
[vtmvg\(\)](#) (in module *spiceypy.spiceypy*), 294
[vupack\(\)](#) (in module *spiceypy.spiceypy*), 295
[vzero\(\)](#) (in module *spiceypy.spiceypy*), 295
[vzerog\(\)](#) (in module *spiceypy.spiceypy*), 295

W

[wncard\(\)](#) (in module *spiceypy.spiceypy*), 295
[wncomd\(\)](#) (in module *spiceypy.spiceypy*), 295
[wncond\(\)](#) (in module *spiceypy.spiceypy*), 296
[wndifd\(\)](#) (in module *spiceypy.spiceypy*), 296
[wnelmd\(\)](#) (in module *spiceypy.spiceypy*), 296
[wnexpd\(\)](#) (in module *spiceypy.spiceypy*), 296
[wnextd\(\)](#) (in module *spiceypy.spiceypy*), 297
[wnfetd\(\)](#) (in module *spiceypy.spiceypy*), 297
[wnfild\(\)](#) (in module *spiceypy.spiceypy*), 297
[wnfltd\(\)](#) (in module *spiceypy.spiceypy*), 297
[wnincd\(\)](#) (in module *spiceypy.spiceypy*), 297
[wninsd\(\)](#) (in module *spiceypy.spiceypy*), 298
[wnintd\(\)](#) (in module *spiceypy.spiceypy*), 298
[wnreld\(\)](#) (in module *spiceypy.spiceypy*), 298
[wnsumd\(\)](#) (in module *spiceypy.spiceypy*), 298

`wnunid()` (*in module `spiceypy.spiceypy`*), 299
`wnvald()` (*in module `spiceypy.spiceypy`*), 299
`writln()` (*in module `spiceypy.spiceypy`*), 299

X

`xf2eul()` (*in module `spiceypy.spiceypy`*), 300
`xf2rav()` (*in module `spiceypy.spiceypy`*), 300
`xfmsta()` (*in module `spiceypy.spiceypy`*), 300
`xpose()` (*in module `spiceypy.spiceypy`*), 300
`xpose6()` (*in module `spiceypy.spiceypy`*), 300
`xposeg()` (*in module `spiceypy.spiceypy`*), 300

Z

`zzdynrot()` (*in module `spiceypy.spiceypy`*), 301