IRENE: AE9/AP9/SPM Radiation Environment Model

Python Application Programming Interface

Version 1.58.001

The IRENE (International Radiation Environment Near Earth): (AE9/AP9/SPM) model was developed by the Air Force Research Laboratory in partnership with MIT Lincoln Laboratory, Aerospace Corporation, Atmospheric and Environmental Research, Incorporated, Los Alamos National Laboratory and Boston College Institute for Scientific Research.

IRENE (AE9/AP9/SPM) development team: Wm. Robert Johnston[1] (PI), T. Paul O'Brien[2] (PI), Gregory Ginet[3] (PI), Stuart Huston[4] Tim Guild[2], Yi-Jiun Su[1], Christopher Roth[5], Rick Quinn[5], Michael Starks[1], Paul Whelan[5], Reiner Friedel[6], Chad Lindstrom[1], Steve Morley[6], and Dan Madden[7].

To contact the IRENE (AE9/AP9/SPM) development team, email ae9ap9@vdl.afrl.af.mil .

The IRENE (AE9/AP9/SPM) model and related information can be obtained from AFRL's Virtual Distributed Laboratory (VDL) website: https://www.vdl.afrl.af.mil/programs/ae9ap9

V1.00.002 release: 05 September 2012

V1.03.001 release: 26 September 2012

V1.04.001 release: 20 March 2013

V1.04.002 release: 20 June 2013

V1.05.001 release: 06 September 2013

V1.20.001 release: 31 July 2014

V1.20.002 release: 13 March 2015

V1.20.003 release: 15 April 2015

V1.20.004 release: 28 September 2015

V1.30.001 release: 25 January 2016

V1.35.001 release: 03 January 2017

V1.50.001 release: 01 December 2017

V1.57.004 release: 21 July 2022

V1.58.001 release: 04 March 2024

[1] Air Force Research Laboratory, Space Vehicles Directorate
[2] Aerospace Corporation
[3] MIT Lincoln Laboratory
[4] Confluence Analytics, Incorporated
[5] Atmospheric and Environmental Research, Incorporated
[6] Los Alamos National Laboratory
[7] Boston College Institute for Scientific Research

IRENE: AE9/AP9/SPM Model
Python Application Programming Interface
Version 1.58.001

Table of Contents

Overview

The IRENE (AE9/AP9/SPM) radiation environment model is distributed with a GUI client application and a command-line driven utility application that can be used to run the model either interactively or through batch-driven processes. For situations in which it is more appropriate to integrate the calls to the environment models and/or data usage directly into a new or existing application, an Application Programming Interface (API) is available.

The IRENE model package supports programmatic access through a suite of APIs accessible from the C++, C and Python programming languages. The main software is written in C++; the C and Python interfaces use "wrappers" for accessing the underlying C++ libraries. The base distribution provides all C and C++ header files, pre-compiled 64-bit Windows executables and library files, and Python modules. The source distribution also contains the complete set of source code files for building the executables and libraries on a Linux system; refer to the 'Build Instructions' document for more details. Either distribution can be used for the development of user applications employing the API libraries.

The IRENE API may be accessed at two different levels: the 'Application-Level' API provides higher-level programmatic access to most of the processing features and options available from the 'CmdLineIrene' application (or its GUI), including parallelization; the 'Model-Level' API provides lower-level programmatic access to each of the underlying models and components that comprise the IRENE model package. This gives the client application developer a great deal of freedom in determining at what level and granularity to integrate with the model. The methods of the 'Application' and each of the individual model classes are described in detail in the remainder of this document.

Important: The execution of any Python script using these IRENE Python modules requires that the top-level installation directory (ie '/home/*username*/Irene') be specified in the environment variable ***PYTHONPATH***; alternatively, that directory may be included in a call to `sys.path.append(<`*install dir*`>)` at the top of the script. The script is also required to include this line:

```
import irene_defs
```

prior to the 'import' of any other IRENE Python module; this defines several other environment variables that are needed by these modules for their proper operation.

Please note that this collection of Python API methods is not fully "Pythonic", in the sense that when an error occurs, an error code is *returned*, instead of throwing an exception. This is largely due to the very nature of these Python methods, simply wrapping the base-level C++ methods and their error handling.

DemoApp and DemoModel Programs

Included in the distribution of IRENE are two demonstration programs for showing how the IRENE API may be used in user applications. They are located in the 'Irene/api_demos' folder of the distribution, written in the three supported languages: C++, C, and Python. The *DemoApp* program demonstrates the use of the Application-level API, specifying an orbit ephemeris and performing several flux and fluence calculations, and then accessing the various forms of the results. The *DemoModel* program demonstrates the use of the IRENE API for several variations of calls to the Ephemeris, AE9, AP9, Adiabatic and Dose individual model components.

As noted above, the execution of the '*demoApp.py*' and '*demoModel.py*' scripts requires that either the *PYTHONPATH* environment variable be defined, or a call to '`sys.path.append(`*<install dir>*' be at the top of these scripts.

These scripts both perform a variety of model calculations and associated operations. When the above requirement is in place, execute the *demoApp* script with this command:

```
python demoApp.py
```

Other script options are available; enter '`python demoApp.py -h`' for the full list.

Execute the *demoModel* script with this command:

```
python demoModel.py
```

It is hoped that these two demonstration scripts will be helpful in building your own custom scripts. Study of the various API methods being called in these scripts against the API reference documents may be helpful in better understanding their effective usage.

Further Details

The IRENE Python modules use 'ctypes' to load and interface with the C library routines, which in turn interface with their corresponding methods in the underlying IRENE C++ library. For the proper loading of these libraries ('*.dll' files on Windows, or '*.so' files on Linux), any script using the IRENE Python API is required to include '`import irene_defs`' *prior* to the 'import' of any other IRENE Python module. This module determines the system type (Linux vs Windows) and defines several (internally used) environment variables that are used for the proper loading of the relevant libraries. This 'irene_defs.py' file is located in the top-level 'Irene' installation directory (possibly renamed by the user). To find and import this file, the calling python script needs the installation directory to either be included in the user's *PYTHONPATH* environment variable, or be specified within the user's script via a call to '`sys.path.append(`*<install dir>*`)`'.

The environment variables defined in *irene_defs.py* are shown in the table below; they may be referenced within the user's script if desired. These variables do *not* persist after the script completes.

Variable	Value	Description
IRENE_HOME	<path>/Irene	Top-level Irene package installation directory, as determined by location of 'irene_def.py' file.
IRENE_SYS	win64 | linux	System type identification string
IRENE_BIN	***IRENE_HOME*/*IRENE_SYS***/bin	Directory of executable files (and Windows DLLs)
IRENE_LIB	***IRENE_HOME*/*IRENE_SYS***/lib[64]	Directory of module library files
IRENE_DB	***IRENE_HOME***/modelData	Directory of model database files
IRENE_XML	***IRENE_HOME***/kernelXml	Directory of kernel xml files

The specification of a directory path and/or filename may include an environment variable, ie `os.environ['IRENE_DB']+'/igrfDB.h5'`.

Application-Level Python API Reference

Application Class

Import file: application.py "`from application import Application`"

This class is the main entry point that provides methods to programmatically configure, execute, and then access the model results available from the CmdLineIrene application. This API does not directly access the models, but instead relies on the execution of the same set of supporting programs used by the CmdLineIrene application, and then queries the results following the completion of the model calculation. Client applications requiring direct or synchronous access to the model results should use the 'model-level' API methods instead.

Computer system environment variables may be used when specifying a directory path and/or filename. Please note that all time values, both input and output, are in Modified Julian Date (MJD) form. Utility methods for conversions to and from MJD times are included here. Position coordinates are always used in sets of three values, in the coordinate system and units that are specified. Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the order of the coordinate values for non-Cartesian coordinate systems.

General:

Application

Usage: Instantiates an instance of the Application class for single- or multi-threaded processing.
Return value:
'application' class object

The following methods are used for adjusting the internal behavior of the model runs.
The use of the *set_execDir*() method is required; all others in this section are optional.

set_execDir

(strExecDir)
Usage: (***Required***) Specifies the directory path to the executable programs that are needed for performing the prescribed model calculations.
Parameters:
strExecDir – path to installation executables, such as *CmdLineIrene(.exe)*
Return value:
iErr – 0 = success, otherwise error

set_workDir

(strWorkDir)
Usage: Specifies the directory path in which a temporary directory, used as a repository for the intermediate binary files generated during model execution, is created. When not specified, this defaults to the current working directory of the client application. Use of an alternate directory may improve model performance. If the work directory is located on a RAID-5 disk unit, use of the *set_numFileIo*() method may further improve performance.
Parameters:

strWorkDir – path to location for temporary directory creation; this working directory will be created if it does not exist.

Return value:

iErr – 0 = success, otherwise error

set_binDirName

(strBinDirName)

Usage: Specifies a non-default name (no path) for the temporary directory containing the intermediate binary files generated during model execution. This may be desired when retaining these files for use with external applications. When this directory name is not specified, a unique name is automatically generated by the 'Application' class.

Parameters:

strBinDirName – name of temporary directory to be created

Return value:

iErr – 0 = success, otherwise error

set_delBinDir

(iVerdict)

Usage: Specifies the disposition of the temporary directory and its intermediate binary files when the 'Application' class object is destroyed or another call is made to the *run_model*() method. When not specified, the default setting is 1 *(true)*. Does not apply to calls to the *reset_modelRun*() method.

Parameters:

iVerdict – set to 1 *(true)* to remove the directory and files, or 0 *(false)* to retain them.

Return value:

iErr – 0 = success, otherwise error

set_numProc

(iNumProc)

Usage: Specifies the total number of processors† to use for the execution of the model calculations. This number *includes* one processor for the 'controller' node. *Must be 3 or greater for parallel processing*. A system call is used to query the number of actual processors, and use this number as a limit. When this query fails or returns an incorrect number (such as on a cluster system), specify the number as a *negative* value to bypass the query. When not specified, model calculations will be performed using a single processor. *On Windows machines with active VPN connections, multi-threaded model runs will fail or stall unless the environment variable '*`FI_TCP_IFACE`*' is set to* '`lo`'.

†Use of Intel CPU's 'Hyper-threaded' threads as a processor may *degrade* performance (YMMV).

Parameters:

iNumProc – number of processors

Return value:

iErr – 0 = success, otherwise error

set_numFileIo

(iNumFileIo)

Usage: Specifies the number of threads to use for the file I/O steps that are performed as part of the normal model calculations. This specification should only be called when using a 'work' directory located on RAID-5 disk unit. RAID-5 disk units are able to efficiently handle concurrent file I/O requests,

while ‘typical’ disk drives cannot. Internally, the number specified is capped at |NumProc|-1. When not specified, these file I/O steps will be performed using a single thread.

Parameters:

iNumFileIo – number of processors

Return value:

iErr – 0 = success, otherwise error

set_windowsMpiMode

(strMode)

Usage: Specifies the MPI communication mode on 64-bit Windows platforms for multi-threaded model execution. This mode determines the additional argument to be supplied to the internal usage of the Intel MPI Library process launcher utility ‘mpiexec’. When not specified, the ‘Local’ mode is used.

Parameters:

strMode – MPI communication mode string (“Local”(default) or “Hydra”)

Local: for use on the local Windows machine with multiple processors

Hydra: for use on a Windows cluster, relies on external 'hydra_service' for MPI communication. The ‘hydra_service’ utility program is included in the Irene/bin/win64 directory. See https://software.intel.com/en-us/node/528873 for more information about this utility.

Return value:

iErr – 0 = success, otherwise error

set_chunkSize

(iChunkSize)

Usage: This specifies the number of ephemeris input entries to be processed during each call to the internal model calculation routines. When not specified, the chunk size is set to 960 by default. Use of this default value is recommended; for systems with limited available memory resources, a value of 120 is suggested to improve performance. However, regardless of ample memory resources, specifying values larger than 2400 will likely *degrade* processing performance.

This specification also governs the size of data ‘chunks’ that are returned from each call to the various *flyin[]*() and *get[]*() data access methods in the “Model Execution and Results” section of this class API. A change in the chunk size causes an implied ‘reset’ of these data access methods; subsequent calls to them will restart the data access from the beginning of the ephemeris input time and positions.

Parameters:

iChunkSize – number of entries in processing chunk; values lower than 60 are not recommended

Return value:

iErr – 0 = success, otherwise error

int set_taskDelay

(iTaskDelay)

Usage: This specifies the number seconds to delay between sets of multi-threaded processing ‘tasks’. Values larger than 1 (the default) are only needed when using the maximum number of processors *and* MPI management-related errors occur (ie ‘*not enough slots*’ or ‘*all nodes are already filled*’). A longer delay may be needed for the MPI “housekeeping” to be completed when executing on a busy system.

Parameters:

iTaskDelay – number of seconds to delay; valid values: 1 - 5.

Return value: int – 0 = success, otherwise error

get_execDir

Usage: Returns the directory path to the executable programs as specified in the *set_execDir*() method.

Return value:

strExecDir – string containing path to installation executables

get_workDir

Usage: Returns the directory path of the temporary directory, as specified in the *set_workDir*() method.

Return value:

strWorkDir – string containing path to location for the temporary directory

get_binDirName

Usage: Returns the name for the temporary directory for the intermediate binary files, as specified in the *set_binDirName*() method.

Return value:

strBinDir – name for the temporary directory for the intermediate binary files, or 'default'.

get_delBinDir

Usage: Returns the disposition of the temporary directory and its intermediate binary files, as specified in the *set_delBinDir*() method.

Return value:

iVerdict – 0 *(false)* or 1 *(true)*

get_numProc

Usage: Returns the number of processors to use, as specified in the *set_numProc*() method.

Return value:

iNumProc – number of processors

get_numFileIo

Usage: Returns the number of threads to use for the file I/O steps, as specified in the *set_numFileIo*() method.

Return value:

iNumFileIo – number of processors

get_windowsMpiMode

Usage: Returns the Windows MPI mode, as specified in the *set_windowsMpiMode*() method.

Return value:

strWinMpiMode – Windows MPI mode string

get_chunkSize

Usage: Returns the current value of the 'chunk' size, as specified in the *set_chunkSize*() method.

Return value:

iChunkSize – number of entries in processing chunk

int get_taskDelay

Usage: Returns the current value of the 'task delay', as specified in the *set_taskDelay*() method.

Return value:

iTaskDelay – number of seconds to delay between multi-threaded processing 'tasks'.

External Ephemeris Specification:

These methods are for explicitly specifying the input ephemeris (time and position coordinates) from an external source. For ephemeris generation, see the following 'Ephemeris Parameter Inputs' section.

set_inCoordSys

(strCoordSys,
strCoordUnits)

Usage: Specifies the coordinate system and units for the position values that are specified by the *set_inEphemeris*() method. When not specified, these settings default to 'GEI' and 'Re'. "Re" = radius of the Earth, defined as 6371.2 km.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL';
Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – coordinate units, 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value

Return value:

iErr – 0 = success, otherwise error

set_inEphemeris

(daTimes,
daCoords1,
daCoords2,
daCoords3,
iAppend = *0*)

Usage: Specifies the input ephemeris time and positions, such as the orbit of a satellite or a grid of positions, to be used for the model calculations.

Parameters:

daTimes – numpy array of time values, in Modified Julian Date form. May be identical times (for defining a *grid*) or times in chronological order, associated with position coordinates.

daCoords1, daCoords2, daCoords3 – numpy arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system and units specified by *set_inCoordSys*().

Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' order of the coordinate values for non-Cartesian coordinate systems.

iAppend – optional flag for appending this ephemeris information to any ephemeris information specified in previous call(s) to *set_ephemeris*(). If 0 *(false)* (or if parameter is omitted), this ephemeris information will replace any existing ephemeris information.

Return value:
iErr – 0 = success, otherwise error

clear_inEphemeris

Usage: Clears any existing input ephemeris information that was specified in previous calls to *set_inEphemeris*().
Return value:
iErr – 0 = success, otherwise error

get_inCoordSys

Usage: Returns the coordinate system, specified in the *set_inCoordSys*() method for the set of position values as specified in the *set_inEphemeris*() method.
Return value:
strInCoordSys – system string

get_inCoordSysUnits

Usage: Returns the coordinate system units, specified in the *set_inCoordSys*() method for the set of position values as specified in the *set_inEphemeris*() method.
Return value:
strInCoordSysUnits – coordinate system units string ('Re' or 'km')

get_inEphemeris

Usage: Returns the input ephemeris time and positions, as specified in the *set_inEphemeris*() method.
Return values:
iNumT – number of ephemeris entries
daTimes – numpy array of time values, in Modified Julian Date form.
daCoords1, daCoords2, daCoords3 – numpy arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system and units returned from the *get_inCoordSys*() and *get_inCoordSysUnits*() methods.

Ephemeris Parameter Inputs:

Ephemeris generation requires the selection of an orbit propagator (and its options), a time range and time step size, and the definition of the orbit characteristics. These orbit characteristics may be defined by either a Two-Line Element (TLE) file, or a set of orbital element values. See the User's Guide '*Orbit Propagation Inputs*' section for details about each of the available settings.

set_propagator

(strPropSpec)
Usage: Specifies the orbit propagator algorithm to use for the ephemeris generation.
Parameters:
strPropSpec – propagator model specification; valid values: 'SatEph', 'SGP4' or 'Kepler'.
Return value:
iErr – 0 = success, otherwise error

set_sgp4Param

(strMode,
strWGS)
Usage: Specifies the mode and WGS parameter for the SGP4 orbit propagator, if being used.
Parameters:
strMode – SGP4 propagation mode; valid values: 'Standard' or 'Improved'.
strWGS – World Geodetic System version; valid values: '72old', '72' or '84'.
Return value:
iErr – 0 = success, otherwise error

set_keplerUseJ2

(iUseJ2)
Usage: Specifies the use of the 'J2' perturbation for the Kepler orbit propagator, if being used.
Parameters:
iUseJ2 – flag for use of the J2 perturbation feature; 1 *(true)* or 0 *(false)*
Return value:
iErr – 0 = success, otherwise error

---

get_propagator

Usage: Returns the orbit propagator identifier, as specified in the *set_propagator*() method.
Return value:
strProp – orbit propagator name.

get_sgp4Mode

Usage: Returns the SGP4 propagator mode setting, as specified in the *set_sGP4Param*() method.
Return value:
strMode – SGP4 mode setting string.

set_sgp4Datum

Usage: Returns the SGP4 propagator WGS setting, as specified in the *set_sGP4Param*() method.
Return value:

strWgs – SGP4 WGS setting string.

get_keplerUseJ2

Usage: Returns the Kepler propagator 'J2' perturbation setting.
Return value:
iVerdict – 1 *(true)* or 0 *(false)*, as specified in the *set_keplerUseJ2*() method.

---

set_times

(dStartTime,
dEndTime,
dTimeStepSec)

Usage: Specifies the start and stop times (*inclusive*), and time step, of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file.
Parameters:
dStartTime, dEndTime – start and stop time values, in Modified Julian Date form
dTimeStepSec – time step size, in seconds
Return value:
iErr – 0 = success, otherwise error

get_times

Usage: Returns the start and stop times, and time step, for the ephemeris generation.
Return values:
dStartTime, dEndTime – start and stop time values, in Modified Julian Date form
dTimeStepSec – time step size, in seconds

set_varTimes

(dStartTime,
dEndTime,
dTimeMinStepSec,
dTimeMaxStepSec = 3600.0,
dTimeRoundSec = 5.0)

Usage: Specifies the start and stop times (*inclusive*), and variable time step limits, of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file. Variable time steps are calculated based on the orbital radial values, and are useful for the more elliptical orbits (ie eccentricity>0.25). See the User's Guide document "Orbit Ephemeris File Description" section for more information.
Parameters:
dStartTime, *dEndTime* - start and stop time values, in Modified Julian Date form
dTimeMinStepSec – lower limit of variable time steps, in seconds; must be ≥ 10 seconds
dTimeMaxStepSec – upper limit of variable time steps, in seconds; must be > min, ≤ 3600 seconds
dTimeRoundSec – rounding of variable time steps, in whole seconds; use 0 for no rounding; < min
Return value:
iErr – 0 = success, otherwise error

get_varTimes

Usage: Returns the start and stop times, and variable time step limits, for the ephemeris generation.
Return values:
dStartTime, *dEndTime* - start and stop time values, in Modified Julian Date form
dTimeMinStepSec – lower limit of variable time steps, in seconds
dTimeMaxStepSec – upper limit of variable time steps, in seconds
dTimeRoundSec – rounding of variable time steps, in seconds

set_timesList

(daTimes)

Usage: Specifies the set of times, in Modified Julian Date form, of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file.
Parameters:
daTimes – numpy array of chronologically ordered time values, in Modified Julian Date form
Return value:
iErr – 0 = success, otherwise error

get_numTimesList

Usage: Returns the number of time entries defined for the ephemeris generation, from the specifications in the *set_timesList*() method.
Return value:
iNumT – number of ephemeris times defined; error if negative

void getTimesList

Usage: Returns the set of time values, in Modified Julian Date form, for the ephemeris generation, from the specifications in the *set_timesList*() method.
Return values:
iNumT – number of ephemeris times defined; error if negative
*daTime*s – vector of time values, in Modified Julian Date form

clear_timesList

Usage: Clears the time entries defined for the ephemeris generation, from the specifications in the *set_timesList*() method.
Return value:
iErr – 0 = success, otherwise error

TLE files are required to be in the standard NORAD format (see User's Guide, Appendix F). The use of the 'Kepler' propagator requires that the TLE file contain only *one* entry. For the other propagators, the TLE may contain multiple entries (for the same satellite), which must be in chronological order.

set_tleFile

(strTLEFile)

Usage: Specifies the name of the Two-Line Element (TLE) file (including path) to use with the selected orbit propagator; this parameter is not needed if a set of orbital element values are being used instead.
Parameters:
strTLEFile – path and filename of TLE file

Return value:
iErr – 0 = success, otherwise error

clear_tleFile

Usage: Clears the specification of the TLE file.
Return value:
iErr – 0 = success, otherwise error

get_tleFile

Usage: Returns the name of the specified TLE file, if any.
Return value:
strTleFile – path and filename of the TLE file, as specified in the *set_tLEFile*() method.

The orbital element values to be specified depend on the type of orbit and/or available orbit definition references. Their use also requires an associated element time to be specified. See the User's Guide document '*Orbiter Propagation Inputs*' section for more details.

set_elementTime

(dElementTime)
Usage: Specifies the 'epoch' time associated with the set of orbital element values.
Parameters:
dElementTime – element 'epoch' time, in Modified Julian Date form
Return value:
iErr – 0 = success, otherwise error

set_inclination

(dInclination)
Usage: Specifies the orbital element 'Inclination' value.
Parameters:
dInclination – orbit inclination angle, in degrees (0-180)
Return value:
iErr – 0 = success, otherwise error

set_rightAscension

(dRtAscOfAscNode)
Usage: Specifies the orbital element 'Right Ascension of the Ascending Node' value.
Parameters:
dRtAscOfAscNode – orbit ascending node position, in degrees (0-360)
Return value:
iErr – 0 = success, otherwise error

set_eccentricity

(dEccentricity)
Usage: Specifies the orbital element 'Eccentricity' value.
Parameters:
dEccentricity – orbit eccentricity value, unitless (0 - <1)

Return value:
iErr – 0 = success, otherwise error

set_argOfPerigee

(dArgOfPerigee)
Usage: Specifies the orbital element 'Argument of Perigee' value.
Parameters:
dArgOfPerigee – orbit perigee position, in degrees (0-360)
Return value:
iErr – 0 = success, otherwise error

set_meanAnomaly

(dMeanAnomaly)
Usage: Specifies the orbital element 'Mean Anomaly' value.
Parameters:
dMeanAnomaly – orbit mean anomaly value, in degrees (0-360)
Return value:
iErr – 0 = success, otherwise error

set_meanMotion

(dMeanMotion)
Usage: Specifies the orbital element 'Mean Motion' value.
Parameters:
dMeanMotion – orbit mean motion value, in units of *revolutions per day* (must be >0)
Return value:
iErr – 0 = success, otherwise error

set_meanMotion1stDeriv

(dMeanMotion1stDeriv)
Usage: Specifies the orbital element 'First Time Derivative of the Mean Motion' value (this should NOT be divided by 2, as when specified in a TLE); this value is only used by the SatEph propagator.
Parameters:
dMeanMotion1stDeriv – first derivative of mean motion, in units of revs per day^2
Return value:
iErr – 0 = success, otherwise error

set_meanMotion2ndDeriv

(dMeanMotion2ndDeriv)
Usage: Specifies the orbital element 'Second Time Derivative of the Mean Motion' value (this should NOT be divided by 6, as when specified in a TLE); this value is only used by the SatEph propagator.
Parameters:
dMeanMotion2ndDeriv – second derivative of mean motion, in units of revs per day^3
Return value:
iErr – 0 = success, otherwise error

set_bStar

(dBStar)

Usage: Specifies the orbital element 'B*' value, for modelling satellite drag effects; this value is only used by the SGP4 propagator.

Parameters:

dBStar – ballistic coefficient value

Return value:

iErr – 0 = success, otherwise error

set_altitudeOfApogee

(dAltApogee)

Usage: Specifies the orbital element 'Apogee Altitude' value (furthest distance).

Parameters:

dAltApogee – altitude (in km) above the Earth's surface at the orbit's apogee (>0, but <~20Re)

Return value:

iErr – 0 = success, otherwise error

set_altitudeOfPerigee

(dAltPerigee)

Usage: Specifies the orbital element 'Perigee Altitude' value (closest distance).

Parameters:

dAltPerigee – altitude (in km) above the Earth's surface at the orbit's perigee (>0, but <~20Re)

Return value:

iErr – 0 = success, otherwise error

set_localTimeOfApogee

(dLocTimeApogee)

Usage: Specifies the local time of the orbit's apogee.

Parameters:

dLocTimeApogee – local time, in hours (0-24)

Return value:

iErr – 0 = success, otherwise error

set_localTimeMaxInclination

(dLocTimeMaxIncl)

Usage: Specifies the local time of the orbit's maximum inclination (ie max latitude).

Parameters:

dLocTimeMaxIncl – local time, in hours (0-24)

Return value:

iErr – 0 = success, otherwise error

set_timeOfPerigee

(dTimeOfPerigee)

Usage: Specifies the time of the orbit's perigee, as an alternative to the Mean Anomaly specification. *Any Mean Anomaly value also specified will be overridden by this value*.

Parameters:

dTimeOfPerigee – time, in Modified Julian Date form, for orbit perigee
Return value:
iErr – 0 = success, otherwise error

set_semiMajorAxis

(dSemiMajorAxis)
Usage: Specifies the orbit's semi-major axis length.
Parameters:
dSemiMajorAxis – semi-major axis length (1-75), in units of Re (radius of Earth = 6371.2 km)
Return value:
iErr – 0 = success, otherwise error

set_geosynchLon

(dGeosynchLon)
Usage: Specifies the geographic East longitude of satellite in a geosynchronous orbit.
Parameters:
dGeosynchLon – East longitude, in degrees (-180 – 360)
Return value:
iErr – 0 = success, otherwise error

set_stateVectors

(daPos,
daVel)
Usage: Specifies the satellite's position and velocity in the GEI coordinate system at the element's 'epoch' time. Alternatively, the *set_positionGEI*() and *set_velocityGEI*() method could be used instead.
Parameters:
daPos – numpy array containing the GEI coordinate system satellite position values (X,Y,Z), in km
daVel – numpy array containing the GEI coordinate system satellite velocity values (X,Y,Z), in km/sec
Return value:
iErr – 0 = success, otherwise error

set_positionGEI

(dPosX,
dPosY,
dPosZ)
Usage: Specifies the satellite's position in the GEI coordinate system at the element's 'epoch' time. This must be used in conjunction with the *set_velocityGEI*() method.
Parameters:
dPosX, dPosY, dPosZ – GEI coordinate system satellite position values, in km (>1Re, but <~75Re)
Return value:
iErr – 0 = success, otherwise error

set_velocityGEI

(dVelX,
dVelY,
dVelZ)

Usage: Specifies the satellite's velocity in GEI coordinate system at the element's 'epoch' time. This must be used in conjunction with the *set_positionGEI*() method.

Parameters:

dVelX, dVelY, dVelZ – GEI coordinate system satellite velocity values, in km/sec

Return value:

iErr – 0 = success, otherwise error

set_coordSys

(strCoordSys,
strCoordUnits)

Usage: Specifies the coordinate system and units for the position values that will be generated by the propagator specified by *set_propagator*(). These default to 'GEI' and 'Re' when not specified; if the *set_inCoordSys*() method was called, the coordinate system and units will set to match those specifications. "Re" = radius of the Earth, defined as 6371.2 km.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL';
Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

Return value:

iErr – 0 = success, otherwise error

---

get_elementTime

Usage: Returns the 'epoch' time associated with the set of orbital element values, as specified in the *set_elementTime*() method

Return value:

dElementTime – element 'epoch' time, in Modified Julian Date form..

get_inclination

Usage: Returns the orbital element 'Inclination' value, as specified in the *set_inclination*() method.

Return value:

dInclination – orbit inclination angle, in degrees.

get_rightAscension

Usage: Returns the orbital element 'Right Ascension of the Ascending Node' value, as specified in the set_rightAscension() method.

Return value:

dRightAsc – orbit ascending node position, in degrees.

get_eccentricity

Usage: Returns the orbital element 'Eccentricity' value, as specified in the *set_eccentricity*() method.

Return value:

dEccentricity – orbit eccentricity value (unitless).

get_argOfPerigee

Usage: Returns the orbital element 'Argument of Perigee' value, as specified in the *set_argOfPerigee*() method.

Return value:

dArgPerigee – orbit perigee position, in degrees.

get_meanAnomaly

Usage: Returns the orbital element 'Mean Anomaly' value, as specified in the *set_meanAnomaly*() method.

Return value:

double – orbit mean anomaly value, in degrees.

get_meanMotion

Usage: Returns the orbital element 'Mean Motion' value, as specified in the *set_meanMotion*() method.

Return value:

dMeanMotion – orbit mean motion value, in revolutions per day.

get_meanMotion1stDeriv

Usage: Returns the orbital element 'First Time Derivative of the Mean Motion' value, as specified in the *set_meanMotion1stDeriv*() method.

Return value:

dMm1stDer – first derivative of mean motion.

get_meanMotion2ndDeriv

Usage: Returns the orbital element 'Second Time Derivative of the Mean Motion' value, as specified in the *set_meanMotion2ndDeriv*() method

Return value:

dMm2ndDer – second derivative of mean motion.

get_bStar

Usage: Returns the orbital element 'B*' value, as specified in the *set_bStar*() method.

Return value:

dBStar – ballistic coefficient value.

get_altitudeOfApogee

Usage: Returns the orbital element 'Apogee Altitude' value, as specified in the *set_altitudeOfApogee*() method.

Return value:

dApogeeAlt – orbit apogee altitude, in km.

get_altitudeOfPerigee

Usage: Returns the orbital element 'Perigee Altitude' value, as specified in the *set_altitudeOfPerigee*() method.

Return value:

dPerigeeAlt – orbit perigee altitude, in km.

get_localTimeOfApogee

Usage: Returns the local time of the orbit's apogee, as specified in the *set_localTimeOfApogee*() method.

Return value:

double – local time of orbit perigee, in hours + fraction.

set_localTimeMaxInclination

Usage: Returns the local time of the orbit's maximum inclination, as specified in the *set_localTimeMaxInclination*() method.

Return value:

dLtMaxInclHr – local time of orbit maximum inclination, in hours + fraction.

get_timeOfPerigee

Usage: Returns the time of the orbit's perigee, as specified in the *set_timeOfPerigee*() method.

Return value:

dTimeOfPerigeeMjd – orbit perigee time value, in Modified Julian Date form.

get_semiMajorAxis

Usage: Returns the orbit's semi-major axis length, as specified in the *set_semiMajorAxis*() method.

Return value:

dTimeOfPerigeeMjd – orbit semi-major axis length, in units of Re (1 Re = 6371.2 km).

get_geosynchLon

Usage: Returns the geographic longitude of satellite in a geosynchronous orbit, as specified in the *set_geosynchLon*() method.

Return value:

dGeosynchLon – orbit geosynchronous (East) longitude, in degrees.

get_stateVectors

Usage: Returns the satellite's position and velocity array values, as specified in the *set_stateArrays*() method, or in the *set_positionGEI*() and *set_velocityGEI*() methods.

Return values:

daPos – numpy array containing the GEI coordinate system satellite position values (X,Y,Z), in km

daVel – numpy array containing the GEI coordinate system satellite velocity values (X,Y,Z), in km/sec

get_positionGEI

Usage: Returns the satellite's position array, as specified in either the *set_positionGEI*() or *set_stateArrays*() method.

Return values:

dPosX, dPosY, dPosZ – GEI coordinate system satellite position values, in km

get_velocityGEI

Usage: Returns the satellite's velocity array, as specified in either the *set_velocityGEI*() or *set_stateArrays*() method.

Return values:

dVelX, dVelY, dVelZ – GEI coordinate system satellite velocity values, in km/sec

get_coordSys

Usage: Returns the coordinate system name, as specified in the *set_coordSys*() method.
Return value:
strCoordSys – coordinate system name.

get_coordSysUnits

Usage: Returns the coordinate system units, as specified in the *set_coordSys*() method.
Return value:
strCoordSysUnits – coordinate system units (‘Re’ or ‘km’).

Model Parameter Inputs:

Methods for specifying the various options and settings for the radiation belt model flux calculations. Only a subset of these methods may be used with the available 'Legacy' models. See the User's Guide, Appendices A and B for more information.

set_model

(strModel)

Usage: Specifies the name of the flux model to be used in the calculations. Note the 'Plasma' model names now includes the species type.
See the following 'Legacy Model Parameter Inputs' section for Legacy model-specific options.

Parameters:

strModel – model name: 'AE9', 'AP9', 'PlasmaE', 'PlasmaH', 'PlasmaHe' or 'PlasmaO'
Legacy models: 'AE8', 'AP8', 'CRRESELE', 'CRRESPRO' or 'CAMMICE'

Return value:

iErr – 0 = success, otherwise error

set_modelDBDir

(strDataDir)

Usage: Specifies the directory that contains the collection IRENE model database files. The various database files required are automatically selected according to the model and parameters specified. The use of this method is highly recommended, as it *eliminates* the need for the other methods that specify the individual database files; those are only needed for using alternate or non-standard versions.

Parameters:

strDataDir – directory path for the IRENE database files.

Return value:

iErr – 0 = success, otherwise error

set_modelDBFile

(strDataSource)

Usage: Specifies the name of the database file (including path) for flux model calculations. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

Please consult the User's Guide for the exact database filename associated with each model.

Parameters:

strDataSource – model database filename, including path

Return value:

iErr – 0 = success, otherwise error

set_kPhiDBFile

(strDataSource)

Usage: Specifies the name of the file (including path) for the K/Phi database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/fastPhi_net.mat'.

Parameters:

strDataSource – database filename, including path

Return value:
iErr – 0 = success, otherwise error

set_kHMinDBFile

(strDataSource)

Usage: Specifies the name of the file (including path) for the K/Hmin database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/fast_hmin_net.mat'.

Parameters:
strDataSource – database filename, including path
Return value:
iErr – 0 = success, otherwise error

set_magfieldDBFile

(strDataSource)

Usage: Specifies the name of the file (including path) for the magnetic field model database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/igrfDB.h5'.

Parameters:
strDataSource – magnetic field model database filename, including path
Return value:
iErr – 0 = success, otherwise error

set_doseModelDBFile

(strDataSource)

Usage: Specifies the name of the file (including path) for the dose calculation model database. The use of this method is *not needed* when *set_modelDBDir*() and/or *set_magfieldDBFile*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/sd2DB.h5'.

Parameters:
strDataSource – dose calculation model database filename, including path
Return value:
iErr – 0 = success, otherwise error

set_adiabatic

(iVerdict = *1*)

Usage: Specifies if the full adiabatic invariant values are to be calculated and be available during model run. Default state is 'off'. Not applicable for Legacy model runs.

Parameters:
iVerdict – optional flag for activating(default) or deactivating the adiabatic invariant calculation.
Return value:
iErr – 0 = success, otherwise error

set_fluxType

(strFluxType)

Usage: Specifies the type of flux values to be calculated by the model. Note that use of '2PtDiff' type requires that the sets of lower and upper bound flux energies be defined using the *set_fluxEnergies*() [or *set_fluxEnergy*()] and *set_fluxEnergies2*() [or *set_fluxEnergy2*()] methods, respectively..

Parameters:

strFluxType – flux type identifier: '1PtDiff', '2PtDiff' or 'Integral'

Return value:

iErr – 0 = success, otherwise error

set_fluxEnergies

(daEnergies)

Usage: Specifies the set energies at which the flux values are calculated by the selected model.
If using the '2PtDiff' flux type, these specify *lower bounds* of energy bins.
Please consult User's Guide for valid ranges/values, which depend on the model selected.

Parameters:

daEnergies – numpy array of energy values [MeV]

Return value:

iErr – 0 = success, otherwise error

set_fluxEnergy

(dEnergy)

Usage: Specifies a single energy to be added to the list of defined flux energies.
If using the '2PtDiff' flux type, these specify *lower bounds* of energy bins.
Please consult User's Guide for valid ranges/values, which depend on the model selected.

Parameters:

dEnergy – energy value [MeV]

Return value:

iErr – 0 = success, otherwise error

clear_fluxEnergies

Usage: Clears the flux energy list defined by previous calls to the *set_fluxEnergy*() or *set_fluxEnergies*() methods.

Return value: -none-

set_fluxEnergies2

(daEnergies2)

Usage: This is only needed when using the '2PtDiff' flux type. Specifies the *upper bounds* of the energy bins, corresponding to the energies (as lower bounds) specified using the *set_fluxEnergy*() or *set_fluxEnergies*() methods.
Please consult User's Guide for valid ranges/values, which depend on the model selected.

Parameters:

daEnergies2 – numpy array of energy values [MeV]

Return value:

iErr – 0 = success, otherwise error

set_fluxEnergy2

(dEnergy)

Usage: This is only needed when using the '2PtDiff' flux type. Specifies a single energy to be added to the list of defined *upper bound* flux energies, corresponding to the energies (as lower bounds) specified using the *set_fluxEnergy*() or *set_fluxEnergies*() methods.

Please consult User's Guide for valid ranges/values, which depend on the model selected.

Parameters:

dEnergy – energy value [MeV]

Return value:

iErr – 0 = success, otherwise error

clear_fluxEnergies2

Usage: Clears the *upper bound* flux energy list defined by previous calls to the *set_fluxEnergy2*() or *set_fluxEnergies2*() methods.

Return value: -none-

set_pitchAngle

(dPitchAngle)

Usage: Specifies a uni-directional pitch angle for the model calculations (not valid for Legacy models). Multiple calls to this method add to the list of pitch angles. *Not compatible with Dose calculations.*

Parameters:

dPitchAngle – pitch angle, in degrees (0-180)

Return value:

iErr – 0 = success, otherwise error

set_pitchAngles

(daPitchAngles)

Usage: Specifies a list of uni-directional pitch angles for the model calculations (not valid for Legacy models). This replaces any previously defined pitch angle list values. *Not compatible with Dose calculations.*

Parameters:

daPitchAngles – numpy array of pitch angles, in degrees (0-180)

Return value:

iErr – 0 = success, otherwise error

clear_pitchAngles

Usage: Clears the pitch angle list defined by calls to *set_pitchAngle*() or *set_pitchAngles*() methods.

Return value:

iErr – 0 = success, otherwise error

set_fluxMean

(iVerdict = *1*)

Usage: Specifies the calculation of the 'mean' flux values by the selected model. All flux values calculated by the Legacy models are considered 'mean' fluxes.

Parameters:

iVerdict – optional flag for activating(default) or deactivating the calculation of mean flux values.

Return value:
iErr – 0 = success, otherwise error

set_fluxPercentile

(iPercent)

Usage: Specifies the calculation of 'percentile' flux values by the selected model (not valid for Legacy models). This method may be called multiple times for the specification of more than one percentile.

Parameters:
iPercent – percentile flux value (1-99) to be calculated by the model

Return value:
iErr – 0 = success, otherwise error

set_fluxPercentiles

(iaPercentiles)

Usage: Specifies the calculation of one or more 'percentile' flux values by the selected model (not valid for Legacy models). The list of percentile values supersedes any prior calls for defining percentile values.

Parameters:
iaPercentiles – numpy array of percentile flux values (1-99) to be calculated by the model

Return value:
iErr – 0 = success, otherwise error

clear_fluxPercentiles

Usage: Clears the current list of defined flux percentile values.

Return value:
iErr – 0 = success, otherwise error

set_fluxPerturbedScenario

(iScenario)

Usage: Specifies the calculation of the particular scenario number of 'perturbed mean' flux values by the selected model (not valid for Legacy models). This method may be called multiple times for the specification of more than one scenario number.

Parameters:
iScenario – scenario number (1-999) of perturbed mean flux values to be calculated by the model

Return value:
iErr – 0 = success, otherwise error

set_fluxPerturbedScenarios

(iaScenarios)

Usage: Specifies the calculation of one or more scenario number of 'perturbed mean' flux values by the selected model (not valid for Legacy models). The list of scenario numbers supersedes any prior calls for defining the scenario numbers for perturbed mean calculations.

Parameters:
iaScenarios – numpy array of scenario numbers (1-999) of perturbed mean flux values to be calculated by the model

Return value:

iErr – 0 = success, otherwise error

set_fluxPerturbedScenRange

(iScenarioMin,
iScenarioMax)

Usage: Specifies the calculation of several scenario numbers, defined by an inclusive range, of 'perturbed mean' flux values by the selected model (not valid for Legacy models). The resulting list of scenario numbers supersedes any prior calls for defining the scenario numbers for perturbed mean calculations.

Parameters:

iScenarioMin – first of the range of scenario numbers (1-999) of perturbed mean flux values to be calculated by the model

iScenarioMax – last of the range of scenario numbers (1-999) of perturbed mean flux values to be calculated by the model

Return value:

iErr – 0 = success, otherwise error

clear_fluxPerturbedScenarios

Usage: Clears the current list of defined perturbed mean scenario numbers.

Return value:

iErr – 0 = success, otherwise error

set_fluxMonteCarloScenario

(iScenario)

Usage: Specifies the calculation of the particular scenario number of 'Monte Carlo' flux values by the selected model (not valid for PLASMA or Legacy models). This method may be called multiple times for the specification of more than one scenario number.

Parameters:

iScenario – scenario number (1-999) of Monte Carlo flux values to be calculated by the model

Return value:

iErr – 0 = success, otherwise error

set_fluxMonteCarloScenarios

(iaScenarios)

Usage: Specifies the calculation of one or more scenario number of 'Monte Carlo' flux values by the selected model (not valid for PLASMA or Legacy models). The list of scenario numbers supersedes any prior calls for defining the scenario numbers for Monte Carlo calculations.

Parameters:

iaScenarios – numpy array of scenario numbers (1-999) of Monte Carlo flux values to be calculated by the model

Return value:

iErr – 0 = success, otherwise error

set_fluxMonteCarloScenRange

(iScenarioMin,
iScenarioMax)

Usage: Specifies the calculation of several scenario numbers, defined by an inclusive range, of 'Monte Carlo' flux values by the selected model (not valid for PLASMA or Legacy models). The resulting list of scenario numbers supersedes any prior calls for defining the scenario numbers for Monte Carlo calculations.

Parameters:

iScenarioMin – first of the range of scenario numbers (1-999) of Monte Carlo flux values to be calculated by the model

iScenarioMax – last of the range of scenario numbers (1-999) of Monte Carlo flux values to be calculated by the model

Return value:

iErr – 0 = success, otherwise error

clear_fluxMonteCarloScenarios

Usage: Clears the current list of defined Monte Carlo scenario numbers.

Return value:

iErr – 0 = success, otherwise error

set_monteCarloEpochTime

(dEpochTime)

Usage: Specifies the reference time used in the time progression of the Monte Carlo flux calculations.

Parameters:

dEpochTime – Monte Carlo reference time, in Modified Julian Date form

Return value:

iErr – 0 = success, otherwise error

set_monteCarloFluxPerturb

(iVerdict)

Usage: Enables (*1*) or disables (*0*) the flux perturbations in Monte Carlo calculations. By default, perturbation mode is enabled. Disabling these perturbations is generally only useful for validation or where perturbations dwarf physical features of interest.

Parameters:

iVerdict – 1 *(true)* or 0 *(false)* for the use of flux perturbations in the Monte Carlo calculations.

Return value:

iErr – 0 = success, otherwise error

set_monteCarloWorstCase

(iVerdict)

Usage: Enables (*1*) or disables (*0*) the tracking of the 'maximum-to-date' Monte Carlo flux average results *for the 'Boxcar' and 'Exponential' accumulation modes only*. By default, this tracking is disabled.

Parameters:

iVerdict – 1 *(true)* or 0 *(false)* for the tracking of the 'maximum-to-date' Monte Carlo results.

Return value:

iErr – 0 = success, otherwise error

---

get_model

Usage: Returns the name of the flux model, as specified in the *set_model*() method.
Return value:
strModel – model name string.

get_modelDBDir

Usage: Returns the directory name containing the collection of IRENE model database files that was specified in a previous call to the *set_modelDBDir*() method; otherwise, blank.
Return value:
strModelDBDir – model database directory.

get_modelDBFile

Usage: Returns the name of the database file for flux model calculations. This will be available immediately, when specified using the *set_modelDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *validate_parameters*() or *run_model*() methods.
Return value:
strModelDB – model database filename.

get_kPhiDBFile

Usage: Returns the name of the file for the K/Phi database. This will be available immediately, when specified using the *set_kPhiDBFile*() method. When the set_*modelDBDir*() method is used, the automatically determined filename will be available after a call to the *validate_parameters*() or *run_model*() methods.
Return value:
strKPhiDB – K/Phi database filename.

get_kHMinDBFile

Usage: Returns the name of the file for the K/Hmin database. This will be available immediately, when specified using the *set_kHMinDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *validate_parameters*() or *run_model*() methods.
Return value:
strKHMinDB – K/Hmin database filename

get_magfieldDBFile

Usage: Returns the name of the file for the magnetic field model database. This will be available immediately, when specified using the *set_magfieldDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *validate_parameters*() or *run_model*() methods.
Return value:
strMagfieldDB – magnetic field model database filename.

get_doseModelDBFile

Usage: Returns the name of the file for the dose calculation model database. This will be available immediately, when specified using the *set_doseModelDBFile*() method. When the *set_modelDBDir*()

method is used, the automatically determined filename will be available after a call to the *validate_parameters*() or *run_model*() methods.

Return value:

strDoseModelDB – dose calculation model database filename.

get_adiabatic

Usage: Returns the current setting for the calculation of the adiabatic invariant values, as specified in the *set_adiabatic*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*

get_fluxType

Usage: Returns the type of flux values to be calculated, as specified in the *set_fluxType*() method.

Return value:

strFluxType – flux type identifier string

get_numFluxEnergies

Usage: Returns the number of currently defined flux energy levels, as specified in the *set_fluxEnergies*() or *set_fluxEnergy*() methods.

Return value:

iNumE – number of energy levels.

get_fluxEnergies

Usage: Returns the array of energy levels, for the model flux calculations, as specified in the *set_fluxEnergies*() or *set_fluxEnergy*() methods.

Return value:

iNumE – number of energy levels; error if negative

daEnergies – numpy array of energy values, in units of MeV.

get_numFluxEnergies2

Usage: Returns the number of currently defined upper bounds of the flux energy levels, as specified in the *set_fluxEnergies2*() or *set_fluxEnergy2*() methods.

Return value:

iNumE2 – number of energy levels.

get_fluxEnergies2

Usage: Returns the upper bounds of the energy bins, for '2PtDiff' flux type, as specified in the *set_fluxEnergies2*() or *set_fluxEnergy2*() methods.

Return value:

iNumE2 – number of energy levels; error if negative

daEnergies2 – numpy array of energy values, in units of MeV.

get_numPitchAngles

Usage: Returns the number of currently defined uni-directional pitch angles, as specified in either *set_pitchAngle*() or *set_pitchAngles*() method.

Return value:

iNumP – number of pitch angles.

get_pitchAngles

Usage: Returns the array of the currently defined uni-directional pitch angles, as specified in either *set_pitchAngle*() or *set_pitchAngles*() method.

Return value:

iNumP – number of pitch angles; error if negative

daPitchAngles – numpy array of pitch angles.

get_fluxMean

Usage: Returns the current state for the calculation of the 'mean' flux values, as specified in the *set_fluxMean*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*.

get_numFluxPercentiles

Usage: Returns the number of flux percentiles defined for the model calculation, as specified in either *set_fluxPercentile*() or *set_fluxPercentiles*() method.

Return value:

iNumPer – number of flux percentiles.

get_fluxPercentiles

Usage: Returns the array of defined flux percentile values, as specified either *set_fluxPercentile*() or *set_fluxPercentiles*() method.

Return value:

iNumPer – number of flux percentiles; error if negative

iaPercentiles – numpy array of percentile flux values

get_numFluxPerturbedScenarios

Usage: Returns the number of perturbed mean scenarios defined for the model calculation, as specified in the *set_fluxPerturbedScenario*(), *set_fluxPerturbedScenarios*(), or *set_fluxPerturbedScenRange*() method.

Return value:

iNumScen – number of scenarios.

get_fluxPerturbedScenarios

Usage: Returns the array of defined perturbed flux scenario numbers, as specified in the *set_fluxPerturbedScenario*(), *set_fluxPerturbedScenarios*(), or *set_fluxPerturbedScenRange*() method.

Return value:

iNumP – number of scenario numbers; error if negative

iaScenarios – numpy array of scenario numbers.

get_numFluxMonteCarloScenarios

Usage: Returns the number of monte carlo scenarios defined for the model calculation, as specified in the *set_fluxMonteCarloScenario*() or *set_fluxMonteCarloScenarios*(), or *set_fluxMonteCarloScenRange*() method.

Return value:
iNumScen – number of scenarios.

get_fluxMonteCarloScenarios

Usage: Returns the array of defined monte carlo flux scenario numbers, as specified in the *set_fluxMonteCarloScenario*() or *set_fluxMonteCarloScenarios*(), or *set_fluxMonteCarloScenRange*() method.
Return value:
iaScenarios – numpy array of scenario numbers.

get_monteCarloEpochTime

Usage: Returns the reference time used in the time progression of the Monte Carlo flux calculations, as specified in the *set_monteCarloEpochTime*() method.
Return value:
dEpochTime – Monte Carlo reference time, in Modified Julian Date form.

get_monteCarloFluxPerturb

Usage: Returns the current Monte Carlo perturbation mode, as specified in the *set_monteCarloFluxPerturb*() method.
Return value:
iVerdict – 1 *(true)* or 0 *(false)*.

get_monteCarloWorstCase

Usage: Returns the current state for the tracking of 'maximum-to-date' Monte Carlo results, as specified in the *set_monteCarloWorstCase*() method.
Return value:
iVerdict – 1 *(true)* or 0 *(false)*.

The following methods specify the further processing to be performed using the calculated flux values.

These 'Accumulation' settings affect the results of the calculated fluence, dose rate, accumulated dose, and some forms of the processed flux values.

set_accumMode

(strAccumMode)
Usage: Specifies an accumulation mode to be used for the processing of the model flux results; this method may be called multiple times for defining multiple modes. Note: these are saved in the order in which they are defined. When no modes are specified, the accumulation mode defaults to 'Interval', with a length of 1 day, unless otherwise specified via the *set_accumInterval[Sec]*() method.
Please consult the User's Guide for more details about these accumulation modes.
Parameters:
strAccumMode – accumulation mode identifier: 'Cumul'|'Cumulative', 'Intv'|'Interval', 'Full', 'Boxcar' or 'Expon'|'Exponential' [the 'Boxcar' requires both interval and increment values to be defined]
Return value:

iErr – 0 = success, otherwise error

clear_accumModes

Usage: Removes all previous accumulation modes specified via the *set_accumMode*() method.
Return value:
iErr – 0 = success, otherwise error

set_accumInterval

(dIntervalDays)

Usage: Specifies a time duration of the accumulation of flux data for use in the calculation of the fluence and/or dose results for the 'Interval', 'Boxcar' and/or 'Exponential' modes; this method may be called multiple times for defining multiple intervals (a maximum of 9 intervals are allowed); these are saved in ascending order. When no intervals are specified, the accumulation interval defaults to 1.0 days (=86400 seconds).
Parameters:
dIntervalDays – time duration, in units of days+fraction.
Return value:
iErr – 0 = success, otherwise error

set_accumIntervalSec

(dInterval)

Usage: Same as *set_accumInterval*(), except that the duration is specified in seconds. When no intervals are specified, the accumulation interval defaults to 86400 seconds (=1.0 days).
Parameters:
dInterval – time duration, in units of seconds.
Return value:
iErr – 0 = success, otherwise error

clear_accumIntervals

Usage: Removes all previous accumulation intervals specified via *set_accumInterval[Sec]*() method.
Return value:
iErr – 0 = success, otherwise error

set_accumIncrementSec

(dIncrem)

Usage: Specifies the time delta for the shift of the 'Boxcar' accumulation mode time windows.
Parameters:
dIncrem – time delta, in seconds, for the increment of time between the start of adjacent Boxcar time windows. May be zero, for self-advancing at the input ephemeris timesteps, or be greater than zero, but less than the specified interval duration.
Return value:
iErr – 0 = success, otherwise error

set_accumIncrementFrac

(dIncremFrac)

Usage: Specifies the time delta for the shift of the Boxcar accumulation time windows, expressed as a fraction of the accumulation time window duration (specified in *set_accumInterval[Sec]*() calls) .

Parameters:

dIncremFrac – fraction, between 0.0 and 1.0 (exclusive of the ends).

Return value:

iErr – 0 = success, otherwise error

set_reportTimes

(dTimeRef,
dCadence)

Usage: Specifies the reference time and cadence (in days) for the periodic output of the Boxcar and/or Exponential flux average accumulation results. This method (and/or *set_reportTimesSec*) may be called multiple times to build a sequence of successive reporting periods with different cadences.

Parameters:

dTimeRef – Reporting *reference* time, in Modified Julian Date form (this time is not included).

dCadence – cadence of reporting the Boxcar and/or Exponential flux results, in units of days. A cadence value of '0' will halt further reporting of the values at the specified time.

Return value:

iErr – 0 = success, otherwise error

set_reportTimesSec

(dTimeRef,
dCadenceSec)

Usage: Specifies the reference time and cadence (in seconds) for the periodic output of the Boxcar and/or Exponential flux average accumulation results. This method (and/or *set_reportTimes*) may be called multiple times to build a sequence of successive reporting periods with different cadences.

Parameters:

dTimeRef – Reporting *reference* time, in Modified Julian Date form (this time is not included).

dCadence – cadence of reporting the Boxcar and/or Exponential flux results, in units of seconds. A cadence value of '0' will halt further reporting of the values at the specified time.

Return value:

iErr – 0 = success, otherwise error

set_reportAtTime

(dTimeVal)

Usage: Specifies a discrete time for the output of the Boxcar and/or Exponential flux average accumulation results. This method may be called multiple times, and may be used in coordination with other calls to the *set_reportTimes*() and/or s*et_reportTimesSec*() methods.

Parameters:

dTimeVal – Report time, in Modified Julian Date form, of the Boxcar and/or Exponential flux results.

Return value:

iErr – 0 = success, otherwise error

clear_reportTimes

Usage: Removes any previously defined 'ReportTimes' specifications (reference time & cadence).

Return value:

iErr – 0 = success, otherwise error

clear_reportAtTime

Usage: Removes any previously defined 'ReportAt' time specifications.
Return value:
iErr – 0 = success, otherwise error

set_fluence

(iVerdict = *1*)
Usage: Specifies the calculation of fluence values from the flux results of the model run. Use of the *set_accumMode*() and *set_accumInterval[Sec]*() methods will affect the frequency and value of these fluence results; when unspecified, these default to the accumulated fluence being reported at 1 day Intervals.
Parameters:
iVerdict – optional flag for activating(default) or deactivating the calculation of fluence values.
Return value:
iErr – 0 = success, otherwise error

get_numAccumModes

Usage: Returns the number of accumulation mode entries defined, as specified in calls to the *set_accumMode*() method.
Return value:
iNumModes – number of accumulation mode entries.

get_accumMode

Usage: Returns the *first* accumulation mode, as specified in calls to the *set_accumMode*() method. [If none were specified, the appropriate default mode will be set within a call to the optional *validate_parameters*() method, when all parameter specifications have been completed.]
Return value:
strAccumMode – accumulation mode; '-none-' if no modes are defined.

get_accumModeEntry

Usage: Returns the accumulation mode according to a numeric identifier, based on the *order* of calls to the *set_accumMode*() method.
Parameters:
iIdent – accumulation mode identifier, starting at 1; the maximum valid identifier is equal to the result from the *get_numAccumModes*() method.
Return value:
strAccumMode – accumulation mode; '-none-' if no modes are defined; 'error' if invalid identifier.

get_numAccumIntervals

Usage: Returns the number of accumulation intervals defined, as specified in calls to the *set_accumInterval[Sec]*() method.
Return value:
iNumIntv – number of accumulation intervals.

get_accumIntervalSec

Usage: Returns the length of the *smallest* accumulation interval specified in calls to the *set_accumInterval[Sec]*() method.
[If none were specified, the default interval will be set within a call to the optional *validate_parameters*() method, when all parameter specifications have been completed.]

Return value:

dIntvSec – accumulation interval length, in seconds; negative if an error.

get_accumIntervalSecEntry

Usage: Returns the length of the accumulation interval according to a numeric identifier, based on the *ascending* order of the intervals specified in calls to the *set_accumInterval[Sec]*() method.

Parameters:

iIdent – accumulation interval identifier, starting at 1; the maximum valid identifier is equal to the result from the *get_numAccumIntervals*() method.

Return value:

dIntvSec – accumulation interval length, in seconds; negative if an error.

get_accumIncrementSec

Usage: Returns the time delta for the shift of the 'Boxcar' accumulation mode time windows, as specified in the *set_accumIncrementSec*() method.

Return value:

dIncrSec – time delta, in seconds.

get_accumIncrementFrac

Usage: Returns the interval length fraction for the shift of the Boxcar accumulation time windows, as specified in the *set_accumIncrementFrac*() method.

Return value:

dIncrFrac – interval length fraction.

get_numReportTimes

Usage: Returns the number of 'Report Time' entries defined, as specified in calls to *set_reportTimes*() and/or *set_reportTimesSec*() methods.

Return value:

iNum – number of 'Report Time' entries.

get_reportTimesSec

Usage: Returns the vectors of the defined 'Report Time' entries, as specified in calls to *set_reportTimes*() and/or *set_reportTimesSec*() methods.

Return values:

iNum – number of 'Report Time' entries

daTimeRef – array of defined reporting *reference* times, in Modified Julian Date form.

daCadence – array of associated cadence values, in units of seconds.

get_numReportAtTime

Usage: Returns the number of 'Report At' entries defined, as specified in calls to the *set_reportAtTime*() method.

Return value: int – number of 'Report At' entries.

get_reportAtTime

Usage: Returns the array of defined 'Report At' entries, as specified in calls to *set_reportAtTime*() method.

Return values:

iNum – number of 'Report At' entries.

daTimeVal – array of defined report times, in Modified Julian Date form.

get_fluence

Usage: Returns the current state for the calculation of fluence values from the flux.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*, as specified in the *set_fluence*() method

---

Dose Calculations require the use of omni-directional (ie, *pitch angle specifications are NOT permitted*), '1PtDiff'-type differential flux values as their input. A minimum of three shielding depth values are also required. Dose calculations are available for all models except 'PLASMA' and 'CAMMICE'.

set_doseRate

(iVerdict = *1*)

Usage: Specifies the calculation of dose rate values from the flux results of the model run. Use of the *set_accumMode*() and *set_accumInterval[Sec]*() methods will affect the frequency at which these dose rate results are available; when unspecified, these default to 'Interval' dose rate averages over 1 day periods.

Parameters:

iVerdict – optional flag for activating(default) or deactivating the calculation of dose rate values.

Return value:

iErr – 0 = success, otherwise error

set_doseAccum

(iVerdict = *1*)

Usage: Specifies the calculation of cumulative or accumulated dose values from the flux results of the model run. Use of the *set_accumMode*() and *set_accumInterval[Sec]*() methods will affect the frequency at which these accumulated dose results are available; when unspecified, these default to the accumulated dose being reported at 1 day Intervals.

Parameters:

iVerdict – optional flag for activating(default) or deactivating the calculation of dose accum values.

Return value:

iErr – 0 = success, otherwise error

set_doseDepthValues

(daDepths)

Usage: Specifies the list of aluminum shielding thickness depths, in the units specified by *set_doseDepthUnits*() method. A minimum of three depth values are required for performing a model run. Nominal range: 0.100 – 111.1 mm; 3.937 – 4374 mils; 0.027 – 30.0 g/cm^2 .

Parameters:

daDepths – numpy array of depth values
Return value:
iErr – 0 = success, otherwise error

set_doseDepthUnits

(strDepthUnits)
Usage: Specifies the measurement units associated with the depth values specified using the *set_doseDepthValues*() method.
Parameters:
strDepthUnits – unit specification: 'millimeters'|'mm', 'mils' or 'gpercm2'
Return value:
iErr – 0 = success, otherwise error

set_doseDepths

(daDepths,
strDepthUnits)
Usage: Specifies both the list of aluminum shielding thickness depths, and their associated units. A minimum of three depth values are required for performing a model run. Input depth values are expected to be in increasing order, with no duplicates; they will be sorted automatically.
Nominal range: 0.100 – 111.1 mm; 3.937 – 4374 mils; 0.027 – 30.0 g/cm^2 .
Parameters:
daDepths – numpy array of depth values
strDepthUnits – unit specification: 'millimeters'|'mm', 'mils' or 'gpercm2'
Return value:
iErr – 0 = success, otherwise error

set_doseDetector

(strDetector)
Usage: Specifies the dose detector material type that lies behind the aluminum shielding.
Parameters:
strDetector – material name: 'Aluminum'|'Al', 'Graphite', 'Silicon'|'Si', 'Air', 'Bone', 'Tissue', 'Calcium'|'Ca', 'Gallium'|'Ga', 'Lithium'|'Li', 'Glass'|'SiO2', 'Water'|'H2O'
Return value:
iErr – 0 = success, otherwise error

set_doseGeometry

(strGeometry)
Usage: Specifies the geometry of the aluminum shielding in front of (or around) the detector target.
Parameters:
strGeometry – configuration name: 'Spherical4pi', 'Spherical2pi', 'FiniteSlab' or 'SemiInfiniteSlab'
Return value:
iErr – 0 = success, otherwise error

set_doseNuclearAttenMode

(strNucAttenMode)
Usage: Specifies the 'Nuclear Attenuation' mode used during the ShieldDose2 model calculations.

Parameters:
strNucAttenMode – attenuation mode: 'None', 'NuclearInteractions' or 'NuclearAndNeutrons'
Return value:
iErr – 0 = success, otherwise error

set_doseWithBrems

(iVerdict = *1*)
Usage: Specifies whether the electron dose calculations are to include the bremsstrahlung contributions or not. The default model state is to *include* the bremsstrahlung contributions.
Parameters:
iVerdict – optional flag for including(default) or excluding the bremsstrahlung contributions.
Return value: -none-

set_useDoseKernel

(iVerdict = *1*)
Usage: Specifies the calculation of the dose values using the kernel-based method.
Parameters:
iVerdict – optional flag for activating(default) or deactivating the use of the dose kernel.
Return value:
iErr – 0 = success, otherwise error

set_doseKernelDir

(strDataDir)
Usage: Specifies the directory that contains the collection of detector- and geometry-specific dose kernel XML files. The proper file is automatically selected based on the detector and geometry settings.
Parameters:
strDataDir – directory path for the dose kernel XML files.
Return value:
iErr – 0 = success, otherwise error

set_doseKernelFile

(strDataSource)
Usage: Explicitly specifies the dose kernel XML file to be used. This XML file ***must*** match the detector and geometry settings, or incorrect results may be produced.
Parameters:
strDataSource – specific Dose kernel XML filename (including path)
Return value:
iErr – 0 = success, otherwise error

---

get_doseRate

Usage: Returns the current state for the calculation of dose rate values, as specified in the *set_doseRate*() method.
Return value:
iVerdict – 1 *(true)* or 0 *(false)*.

get_doseAccum

Usage: Returns the current state for the calculation of cumulative or accumulated dose, as specified in the *set_doseAccum*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*.

get_numDoseDepthValues

Usage: Returns the number of aluminum shielding thickness depths, as specified in the *set_doseDepths*() or *set_doseDepthValues*() methods.

Return value:

iNumD – number of dose depths.

get_doseDepthValues

Usage: Returns the list of aluminum shielding thickness depths, as specified in the *set_doseDepths*() or *set_doseDepthValues*() methods.

Return value:

daDepths – numpy array of depth values.

get_doseDepthUnits

Usage: Returns the measurement units associated with the depth values, as specified in the *set_doseDepths*() or *set_doseDepthUnits*() methods.

Return value:

strDepthUnits – Dose depth units string.

get_doseDetector

Usage: Returns the dose detector material type that lies behind the aluminum shielding, as specified in the *set_doseDetector*() method.

Return value:

strDetector – material name string.

get_doseGeometry

Usage: Returns the geometry of the aluminum shielding in front of (or around) the detector target, as specified in the *set_doseGeometry*() method.

Return value:

strGeometry – Dose shielding geometry string.

get_doseNuclearAttenMode

Usage: Returns the 'Nuclear Attenuation' mode used during the ShieldDose2 model calculations, as specified in the *set_doseNuclearAttenMode*() method.

Return value:

strNuclearAttenMode – Dose calculation nuclear attenuation mode string.

get_doseWithBrems

Usage: Returns the current state for the inclusion of the bremsstrahlung contribution in the electron dose values calculated, as specified in the *set_doseWithBrems*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*.

get_useDoseKernel

Usage: Returns the current state for the dose values calculated using the kernel method, as specified in the *set_useDoseKernel*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*.

get_doseKernelDir

Usage: Returns the directory that contains the dose kernel XML files, as specified in the *set_doseKernelDir*() method.

Return value:

strDoseKernelDir – directory string for the dose kernel XML files.

get_doseKernelFile

Usage: Returns the XML filename to be used for kernel-based dose calculations, as specified in the *set_doseKernelFile*() method.

Return value:

strDoseKernelFile – Dose kernel XML filename.

These 'Aggregation' settings are used for the calculation of 'confidence levels' from the 'Perturbed Mean' and/or 'Monte Carlo' scenario results. These confidence levels are determined using the percentile calculation method recommended by the National Institute of Standards and Technology (NIST). The endpoints of 0 and100 percent levels are excluded, as well as additional neighboring levels when fewer than 100 scenarios are used in the aggregation (see the User's Guide for more information). The 0 percent level returns the lowest scenario value; the 100 percent level returns the highest scenario value. These results are statistically meaningful only when at least ten scenarios are used.

set_aggregMedian

Usage: Adds the 50% value to the list for aggregation confidence level calculations.
Return value:
iErr – 0 = success, otherwise error

set_aggregConfLevel

(iPercent)
Usage: Adds the specified percent value to the list for aggregation confidence level calculations.
Parameters:
iPercent – percent value to add to list for aggregation confidence level calculations (0-100 valid).
Return value:
iErr – 0 = success, otherwise error

set_aggregConfLevels

(viPercent)
Usage: Specifies the calculation of one or more confidence level values. The list of values supersedes any prior calls for defining these confidence level percentages.
Parameters:
viPercent – numpy array of percent values for the aggregation confidence level calculations (0-100 valid).
Return value:
iErr – 0 = success, otherwise error

clear_aggregConfLevels

Usage: Clears the accumulated list of percent values for aggregation confidence level calculations.
Return value:
iErr – 0 = success, otherwise error

set_aggregMean

Usage: Adds the 'Mean' to the list for aggregation calculations. *This is NOT a confidence level. The results of this calculation are of indeterminate meaning. Use of this method is discouraged.*
Return value:
iErr – 0 = success, otherwise error

get_numAggregConfLevels

Usage: Returns the number of confidence levels for the aggregation calculations, as specified in *set_aggregConfLevel*(), *set_aggregConfLevels*(), or through *set_aggregMean*() or *set_aggregMedian*()

methods.

Return value:

iNumConf – number of confidence levels.

get_aggregConfLevels

Usage: Returns the list of confidence level values, as specified in *set_aggregConfLevel*(), *set_aggregConfLevels*(), or through *set_aggregMean*() or *set_aggregMedian*() methods.

Return value:

iNumConf – number of confidence levels; error if negative

iaPercent – numpy array of percent values for the aggregation confidence level calculations

Legacy Model Parameter Inputs:

These methods are used for specifying the model parameters applicable only to the 'Legacy' models. The flux values calculated by these models are all 'mean', omni-directional flux values.

set_legActivityLevel

(strActivityLevel)

Usage: Specifies the geomagnetic activity level parameter for the CRRESPRO, AE8 or AP8 Legacy model run.

Parameters:

strActivityLevel – geomagnetic activity level specification:
for CRRESPRO model, 'active' or 'quiet';
for AE8 or AP8 model, 'min' or 'max'.

Return value:

iErr – 0 = success, otherwise error

set_legActivityRange

(strActivityRange)

Usage: Specifies the geomagnetic activity level parameter for the CRRESELE Legacy model run. Only one of the *set_activityRange*() or *set15DayAvgAp*() methods may be used, otherwise an error is flagged.

Parameters:

strActivityRange – geomagnetic activity level specification, in terms of Ap values:
'5-7.5', '7.5-10', '10-15', '15-20', '20-25', '>25', 'avg', 'max', or 'all'.

Return value:

iErr – 0 = success, otherwise error

set_leg15DayAvgAp

(d15DayAvgAp)

Usage: Specifies the 15-day average Ap value for the CRRESELE Legacy model run. Only one of the *set_activityRange*() or *set15DayAvgAp*() methods may be used, otherwise an error is flagged.

Parameters:

d15DayAvgAp – 15-day average Ap value; valid values are in the range of 0 – 400.

Return value:

iErr – 0 = success, otherwise error

set_legFixedEpoch

(iFixedEpoch)

Usage: Specifies the use of the model-specific fixed epoch (year) for the magnetic field model in the flux calculations. It is *highly recommended* to set this to 1 *(true)*. Unphysical results may be produced (especially at low altitudes) if set to 0 *(false)*.

Parameters:

iFixedEpoch – 1 *(true)* or 0 *(false)*; when *false*, the ephemeris year is used for the magnetic field model.

Return value:

iErr – 0 = success, otherwise error

set_legShiftSAA

(iShiftSAA)

Usage: Shifts the SAA from its fixed-epoch location to the location for the current year of the ephemeris. This setting is ignored if the ***set_fixedEpoch*** method is set to 0 *(false)*.

Parameters:

iShiftSAA – 1 *(true)* or 0 *(false)*.

Return value:

iErr – 0 = success, otherwise error

---

get_legActivityLevel

Usage: Returns the geomagnetic activity level parameter for the CRRESPRO, AE8 or AP8 Legacy model, as specified in the *seLegtActivityLevel*() method.

Return value:

strActivityLevel – geomagnetic activity level specification string.

get_legActivityRange

Usage: Returns the geomagnetic activity level parameter for the CRRESELE Legacy model, as specified in the *set_legActivityRange*() method.

Return value:

strActivityRange – geomagnetic activity level specification string.

get_leg15DayAvgAp

Usage: Returns the 15-day average Ap value for the CRRESELE Legacy model, as specified in the *set_leg15DayAvgAp*() method.

Return value:

d15DayAvgAp – 15-day average Ap value.

get_legFixedEpoch

Usage: Returns the current setting for the use of the model-specific fixed epoch, as specified in the *set_legFixedEpoch*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*.

get_legShiftSAA

Usage: Returns the current setting of shifting the SAA from its fixed-epoch location, as specified in the *set_legShiftSAA*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*.

The following methods are applicable only to the CAMMICE/MICS Legacy model. This model is set to produce flux values for twelve pre-defined energy bins (see Appendix B of the User's Guide).

set_camMagfieldModel

(strMFModel)

Usage: Specifies the magnetic field option for the CAMMICE Legacy model run. 'igrf' uses the IGRF model without an external field model. 'igrfop' adds Olson-Pfitzer/Quiet as the external field model.

Parameters:

strMFModel – magnetic field model specification: 'igrf' or 'igrfop'.

Return value:

iErr – 0 = success, otherwise error

set_camDataFilter

(strDataFilter)

Usage: Specifies the data filter option for the CAMMICE Legacy model run. 'Filtered' excludes data collected during periods when the DST index was below -100.

Parameters:

strDataFilter – data filter specification: 'all' or 'filtered' .

Return value:

iErr – 0 = success, otherwise error

set_camPitchAngleBin

(strPitchAngleBin)

Usage: Specifies the pitch angle bin for the CAMMICE Legacy model run.

Parameters:

strPitchAngleBin – pitch angle bin identification: '0-10','10-20','20-30','30-40','40-50','50-60','60-70','70-80','80-90', '100-110','110-120','120-130','130-140','140-150','150-160','160-170','170-180' or 'omni'.

Return value:

iErr – 0 = success, otherwise error

set_camSpecies

(strSpecies)

Usage: Specifies the (single) particle species for the CAMMICE Legacy model run.

Parameters:

strSpecies – species identification: 'H+', 'He+', 'He+2', 'O+', 'H', 'He', 'O', or 'Ions'.

Return value:

iErr – 0 = success, otherwise error

---

get_camMagfieldModel

Usage: Returns the magnetic field option for the CAMMICE Legacy model, as specified in the *set_camMagfieldModel*() method.

Return value:

strMFModel – magnetic field model specification string.

get_camDataFilter

Usage: Returns the data filter option for the CAMMICE Legacy model, as specified in the *set_camDataFilter*() method.

Return value:

strDataFilter – data filter specification string.

get_camPitchAngleBin

Usage: Returns the pitch angle bin for the CAMMICE Legacy model, as specified in the *set_camPitchAngleBin*() method.

Return value:

strPitchAngleBin – pitch angle bin identification string.

get_camSpecies

Usage: Returns the particle species for the CAMMICE Legacy model, as specified in the *set_camSpecies*() method.

Return value:

strSpecies – species identification string.

Model Execution and Results:

These methods are to be used after all desired input parameters have been specified.

Following a call to the *run_model*() method, the results from the requested calculations are accessible via the various *flyin_ **() and *get_ *Data*() methods.

All returned 'integral flux' values are in units of [#/cm^2/sec]; and their fluences in [#/cm^2].
All returned 'differential flux' values are in units of [#/cm^2/sec/MeV]; and their fluences in [#/cm^2/MeV].
All returned 'dose rate' values are in units of [rads/sec]; 'accumulated dose' values are in units of [rads].

Important: Please note that these *flyin_ **() and *get_ *Data*() methods return the requested data in 'chunks', defaulting to 960 entries at each method call. Therefore, multiple calls may be required to access the full set of generated model results. Following the call to the *run_model*() method, the sizing of these data access segments may be adjusted using the *set_chunkSize*() method. A call to the *reset_modelData*() method will 'reset' these data access methods, as will a call to change the chunk size. Subsequent calls to the data access methods will restart them from the beginning of the ephemeris input time and positions. Note that the 'flyin' methods ultimately call the 'get_modelData' method under the hood, so is accessing the same data 'stream' (based on the data type and percentile/scenario identifier).

The *flyin_ **() and *get_ *Data*() methods include optional arguments for specifying an accumulation mode and accumulation interval identifier. These are used to distinguish exactly which set of data is to be returned when there are accumulation(s) active. For the 'flux' data type, the 'default' accumulation mode translates to 'cumul', but for all other data types, it translates to the *first* accumulation mode that was defined. The values returned for the 'Cumul' accumulation mode are the "raw" values that are calculated at the input ephemeris times. The values returned for any other accumulation mode (with one or more defined time interval periods) will be computed averages or summations, depending on the data type; their associated time is for the end of the interval.

Some of the data access methods also return the ephemeris position information. These returned vectors of ephemeris position values are in the coordinate system and units previously specified (or accessed via the *get_coordSys*() and *get_coordUnit*() methods). Please consult the User's Guide document, "Supported Coordinate Systems" table for more details; in particular, note the 'standard' ordering of these returned coordinate values for non-Cartesian coordinate systems. For the accumulation data requests which are averages or summations (ie flux average, fluence, dose accumulation), the associated ephemeris values are purposely set to zero, as the data values do not correspond to a single discrete position.

run_model

Usage: Invokes the execution of the model calculations based on the specified parameter inputs. When errors in the inputs/settings are detected, informative messages are shown in the console output.

Return value:
iErr – 0 = success, otherwise error

get_ephemeris

Usage: Returns the ephemeris information, either generated or specified.
Return values:
iNum – int – Number of ephemeris records returned (≥0 = success, <0 = error)
daTimes – numpy array of ephemeris time values, in Modified Julian Date form
daCoord1, daCoord2, daCoord3 – numpy arrays of ephemeris position values, in the coordinate system and units previously specified (also accessible from *get_coordSys*() and *get_coordSysUnits*() methods).
Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the 'standard' ordering of the returned coordinate values for non-Cartesian coordinate systems.

get_coordSys

Usage: Returns the coordinate system of the ephemeris information returned with the model results.
Return value:
strCoordSys – coordinate system identifier; see *set[In]CoordSys*() methods for list of values, or consult the User's Guide document, "Supported Coordinate Systems" for more details.

get_coordSysUnits

Usage: Returns the coordinate system units of the ephemeris information returned with the model results.
Return value:
strCoordUnits – coordinate system unit value; 'km' or 'Re' (radius of Earth = 6731.2 km)

get_numDir

Usage: Returns the number of data directions in the generated flux data.
Return value:
int – number of directions: 1 when 'omnidirectional', otherwise, number of defined pitch angles. (when 'omnidirectional', the *get_numPitchAngles*() method returns 0).

flyin_mean2d

(strAccumMode = *'default'*,
iAccumIntvId = *1*)

Usage: Returns the 'Mean' model flux values for omni-direction model runs. A previous call to the *set_fluxMean(1)* method is required for these results to be available for the Ae9/Ap9/SPM models. This method can also be used for accessing flux results from the 'Legacy' models. Flux accumulated average values may be obtained using this method when specifying the optional arguments.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).
Parameters:
strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.
iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed

when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

da3FluxData – 2-dimensional numpy array of the 'mean' flux values. [time,energy]

flyin_mean

(strAccumMode = *'default'*,
iAccumIntvId = *1*)

Usage: Returns the 'Mean' model flux values. A previous call to the *set_fluxMean(1)* method is required for these results to be available for the Ae9/Ap9/SPM models. This method can also be used for accessing flux results from the 'Legacy' models. Flux accumulated average values may be obtained using this method when specifying the optional arguments.

The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

da3FluxData – 3-dimensional numpy array of the 'mean' flux values. [time,energy,direction]

flyin_meanPlus

(strAccumMode = *'default'*,
iAccumIntvId = *1*)

Usage: Returns the 'Mean' model flux values, along with the associated ephemeris values and pitch angles. A previous call to the *set_fluxMean(1)* method is required for these results to be available for the Ae9/Ap9/SPM models. This method can also be used for accessing flux results from the 'Legacy' models. Flux accumulated average values may be obtained using this method when specifying the optional arguments.

The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

daTimes – numpy array of ephemeris time values, in Modified Julian Date form

daCoord1, daCoord2, daCoord3 – numpy arrays of ephemeris position values, in the coordinate system and units previously specified.

da2PitchAngles – 2-dimensional numpy array of associated pitch angles; will be empty if omni-directional.[time,direction]

da3FluxData – 3-dimensional numpy array of the 'mean' flux values. [time,energy,direction]

flyin_percentile

(iPercentile,
strAccumMode = '*default*',
iAccumIntvId = *1*)

Usage: Returns the 'Percentile' model flux values for the specified percentile. The percentile number must be one of those included in previous calls to the *set_fluxPercentile()* methods. Flux accumulated average values may be obtained using this method when specifying the optional arguments.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

iPercentile – percentile number of the flux values to be returned.

strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

da3FluxData – 3-dimensional numpy array of the flux values for the specified percentile. [time,energy,direction]

flyin_percentilePlus

(iPercentile,
strAccumMode = '*default*',
iAccumIntvId = *1*)

Usage: Returns the 'Percentile' model flux values for the specified percentile, along with the associated ephemeris values and pitch angles. The percentile number must be one of those included in previous calls to the *set_fluxPercentile()* methods. Flux accumulated average values may be obtained using this method when specifying the optional arguments.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

iPercentile – percentile number of the flux values to be returned.

strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed

when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

daTimes – numpy array of ephemeris time values, in Modified Julian Date form

daCoord1, daCoord2, daCoord3 – numpy arrays of ephemeris position values, in the coordinate system and units previously specified.

da2PitchAngles – 2-dimensional numpy array of associated pitch angles; will be empty if omni-directional. [time,direction]

da3FluxData – 3-dimensional numpy array of the flux values for the specified percentile. [time,energy,direction]

flyin_perturbedMean

(iScenario,
strAccumMode = *'default'*,
iAccumIntvId = *1*)

Usage: Returns the 'Perturbed Mean' model flux values for the specified scenario number. The scenario number must be one of those included in previous calls to the *set_fluxPerturbedScen [Range]()* methods. Flux accumulated average values may be obtained using this method when specifying the optional arguments.

The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

iScenario – scenario number of the Perturbed Mean flux values to be returned.

strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

da3Data – 3-dimensional numpy array of the flux values for the specified scenario number. [time,energy,direction]

flyin_perturbedMeanPlus

(iScenario,
strAccumMode = *'default'*,
iAccumIntvId = *1*)

Usage: Returns the 'Perturbed Mean' model flux results for the specified scenario number, along with the associated ephemeris values and pitch angles. The scenario number must be one of those included in previous calls to the *set_fluxPerturbedScen[Range]()* methods. Flux accumulated average values may be obtained using this method when specifying the optional arguments.

The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

iScenario – scenario number of the Perturbed Mean flux values to be returned.

strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

daTimes – numpy array of ephemeris time values, in Modified Julian Date form

daCoord1, daCoord2, daCoord3 – numpy arrays of ephemeris position values, in the coordinate system and units previously specified.

da2PitchAngles – 2-dimensional numpy array of associated pitch angles; will be empty if omni-directional. [time,direction]

da3FluxData – 3-dimensional numpy array of the flux values for the specified scenario number. [time,energy,direction]

flyin_monteCarlo

(iScenario,
strAccumMode = *'default'*,
iAccumIntvId = *1*)

Usage: Returns the 'Monte Carlo' model flux values for the specified scenario number. The scenario number must be one of those included in previous calls to the *set_fluxMonteCarloScen[Range]()* methods. Flux accumulated average values may be obtained using this method when specifying the optional arguments.

The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

iScenario – scenario number of the Monte Carlo flux values to be returned.

strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

da3FluxData – 3-dimensional numpy array of the flux values for the specified scenario number. [time,energy,direction]

flyin_monteCarloPlus

(iScenario,
strAccumMode = *'default'*,
iAccumIntvId = *1*)

Usage: Returns the 'Monte Carlo' model flux values for the specified scenario number, along with the associated ephemeris values and pitch angles. The scenario number must be one of those included in previous calls to the *set_fluxMonteCarloScen[Range]()* methods. Flux accumulated average values may be obtained using this method when specifying the optional arguments.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

iScenario – scenario number of the Monte Carlo flux values to be returned.

strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

daTimes – numpy array of ephemeris time values, in Modified Julian Date form

daCoord1, daCoord2, daCoord3 – numpy arrays of ephemeris position values, in the coordinate system and units previously specified.

da2PitchAngles – 2-dimensional numpy array of associated pitch angles; will be empty if omni-directional.[time,direction]

da3FluxData – 3-dimensional numpy array of the flux values for the specified scenario number. [time,energy,direction]

get_modelData

(strDataType,
strFluxMode,
iCalcVal,
strAccumMode = *'default'*,
iAccumIntvId = *1*)

Usage: Returns the model results from for the specified data type, flux mode and, if applicable, the percentile or scenario identifier, and accumulation mode and interval identifier. See the following routine for accessing *aggregation* results. The associated ephemeris and pitch angles are also returned.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).
The returned dose values are in units of [rads/sec] (doserate) or [rads] (doseaccum).

Parameters:

strDataType – data type identifier: "flux"|"fluence"|"doserate"|"doseaccum"

strFluxMode – flux mode identifier: "mean"|"percent"|"perturbed"|"montecarlo"|"montecarloWC"

iCalcVal – additional model data qualifier: ignored for 'mean' flux mode; model percentile (1-99) for 'percent'; model scenario number (1-999) for 'perturbed' or 'montecarlo'. The number must be one of those included in previous calls to the appropriate flux mode parameter specification methods.

strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

daTimes – numpy array of ephemeris time values, in Modified Julian Date form; when using an accumulation mode other than 'None', this time specifies the *ending* time of the accumulation interval.

daCoord1, daCoord2, daCoord3 – numpy arrays of ephemeris position values, in the coordinate system and units previously specified.

da2PitchAngles – 2-dimensional numpy array of associated pitch angles; will be empty if omni-directional.[time,direction]

da3Data – 3-dimensional numpy array of the specified data values.[time,energy|depth,direction]

get_aggregData

(strDataType,
strFluxMode,
iPercent,
strAccumMode = *'default'*,
iAccumIntvId = *1*)

Usage: Returns the confidence level results from multiple scenarios of data input, for the specified data type, flux mode, confidence level percent, and accumulation mode and interval identifier. The associated ephemeris and pitch angles are also returned.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).
The returned dose values are in units of [rads/sec] (doserate) or [rads] (doseaccum).

Parameters:

strDataType – data type identifier: "flux"|"fluence"|"doserate"|"doseaccum"

strFluxMode – flux mode identifier: "perturbed"|"montecarlo"|"montecarloWC"

iPercent – aggregation confidence level percent (0-100 *or -1 for mean*). The number specified must be one of those included in previous calls to the *set_aggregConfLevel()* and related methods.

strAccumMode – (optional) accumulation mode of the flux values to return. The availability of the flux accumulated average values for the specified accumulation mode is dependent on those modes defined in previous call(s) to the *set_accumMode*() method. See that method description for more information. The default mode of 'default' translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

iAccumIntvId – (optional) accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *get_numAccumIntervals*() method. This argument is only needed when requesting values for intervals other than the smallest, and is dependent on those intervals defined in previous calls to the *set_accumInterval[Sec]*() method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

daTimes – numpy array of ephemeris time values, in Modified Julian Date form; when using an accumulation mode other than 'None', this time specifies the *ending* time of the accumulation interval.

daCoord1, daCoord2, daCoord3 – numpy arrays of ephemeris position values, in the coordinate system and units previously specified.

da2PitchAngles – numpy array of associated pitch angles; will be empty if omni-directional.

da3Data – 3-dimensional numpy array of the specified data values.[time,energy|depth,direction]

get_adiabaticCoords

Usage: Returns the adiabatic invariant values associated with the previously defined or generated ephemeris and pitch angles. If omnidirectional, values returned are for pitch angle of 90. The availability of the adiabatic coordinates and magnetic field information is dependent on a previous call to the *set_adiabatic*(1) method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

da2Alpha – 2-dimensional numpy array of equatorial pitch angles ('alpha') associated with the pitch angles at the ephemeris locations. [time,direction]

da2Lm – 2-dimensional numpy array of McIllwain L-shell value associated with the pitch angles at the ephemeris locations. [time,direction]

da2K – 2-dimensional numpy array of adiabatic invariant 'K' value associated with the pitch angles at the ephemeris locations. [time,direction]

da2Phi – 2-dimensional numpy array of adiabatic invariant 'Phi' associated with the pitch angles at the ephemeris locations. [time,direction]

da2Hmin – 2-dimensional numpy array of adiabatic invariant 'Hmin' associated with the pitch angles at the ephemeris locations. [time,direction]

da2Lstar – 2-dimensional numpy array of adiabatic invariant 'L*' associated with the pitch angles at the ephemeris locations. [time,direction]

daBmin – 1-dimensional numpy array of minimum IGRF model magnetic field strength value (nanoTeslas) along field line containing the ephemeris location. [time].

daBlocal – 1-dimensional numpy array of IGRF model magnetic field strength value (nanoTeslas) at the ephemeris location. [time]

daMagLT – 1-dimensional numpy array of dipole model-based magnetic local time (hours) at the ephemeris locations. [time]

get_adiabaticCoordsPlus

Usage: Returns the ephemeris values, pitch angles and corresponding adiabatic invariant values. If omnidirectional, values returned are for pitch angle of 90. The availability of the adiabatic coordinates and magnetic field information is dependent on a previous call to the *set_adiabatic*(1) method.

Return values:

iNum – int – Number of records returned (≥0 = success, <0 = error)

daTimes – numpy array of ephemeris time values, in Modified Julian Date form

daCoord1, daCoord2, daCoord3 – numpy arrays of ephemeris position values, in the coordinate system and units previously specified.

da2PitchAngles – 2-dimensional numpy array of associated pitch angles; will contain single angle of 90 for all times if omni-directional.[time,direction]

da2Alpha – 2-dimensional numpy array of equatorial pitch angles ('alpha') associated with the pitch angles at the ephemeris locations. [time,direction]

da2Lm – 2-dimensional numpy array of McIllwain L-shell value associated with the pitch angles at the ephemeris locations. [time,direction]

da2K – 2-dimensional numpy array of adiabatic invariant 'K' value associated with the pitch angles at the ephemeris locations. [time,direction]

da2Phi – 2-dimensional numpy array of adiabatic invariant 'Phi' associated with the pitch angles at the ephemeris locations. [time,direction]

da2Hmin – 2-dimensional numpy array of adiabatic invariant 'Hmin' associated with the pitch angles at the ephemeris locations. [time,direction]

da2Lstar – 2-dimensional numpy array of adiabatic invariant 'L*' associated with the pitch angles at the ephemeris locations. [time,direction]

daBmin – 1-dimensional numpy array of minimum IGRF model magnetic field strength value (nanoTeslas) along field line containing the ephemeris location. [time].

daBlocal – 1-dimensional numpy array of IGRF model magnetic field strength value (nanoTeslas) at the ephemeris location. [time]

daMagLT – 1-dimensional numpy array of dipole model-based magnetic local time (hours) at the ephemeris locations. [time]

Utility methods for your convenience

reset_modelData

Usage: Resets the data access to the model output files generated during the most recent model run. Subsequent calls to the various *flyin[]*() and *get[]*() data access methods will return data starting at the beginning of the ephemeris input times and positions.

Return value:

iErr – 0 = success, otherwise error

reset_modelRun

(iDelBinDir = *1*,
dResetParam = *0*)

Usage: Performs cleanup of the most recent model run. This method is needed only if further model run calculations are to be performed again *using the same object*, with new or revised input parameter settings.

Parameters:

iDelBinDir – optional flag for the deletion the temporary binary directory containing the most recent model run output files. If 1 *(true)* (or omitted), the directory is removed. This argument *must* be specified if the second argument is also specified. *Special Note*: the *set_delBinDir* setting only applies to when the Application class object is destroyed when the program completes, or on subsequent calls to *run_model*.

iResetParam – optional flag for the reset of all previously specified model run input parameters. If specified as 1 *(true)*, the input parameters are reset to their initial default values. No change if omitted.

Return value:

iErr – 0 = success, otherwise error

validate_parameters

Usage: Performs a validation of all input parameters, verifying that there are no conflicts between the various settings and that all required values have been specified. *Use of this method is optional*, as it is called internally by the *run_model*() method.

Return value:

iErr – 0 = success; >0: number of errors detected

reset_orbitParameters

Usage: Resets input parameters related to orbit specifications to their initial default values.

Return value:

iErr – 0 = success, otherwise error

reset_parameters

Usage: Resets *all* model run input parameters to their initial default values. Only the ExecDir and WindowsMpiMode parameters retain their settings.

Return value:

iErr – 0 = success, otherwise error

Time Conversion Utilities:

These are utility methods for the conversion between Modified Julian Date values and other date and time format values.

get_gmtsec

(iHours,
iMinutes,
dSeconds)

Usage: Determines the GMT seconds of day for the input hours, minutes and seconds.
Parameters:
iHours – hours of day (0-23)
iMinutes – minutes of hour (0-59)
dSeconds – seconds of minute (0-59.999)
Return value:
dGmtSec – GMT seconds of day

get_dayYear

(iYear,
iMonth,
iDay)

Usage: Determines the day number of year for the input year, month and day number.
Parameters:
iYear – year (1950-2049)
iMonth – month (1-12)
iDay – day of month (1-28|29|30|31)
Return value:
iDay – day number of year

get_modJulDate

(iYear,
iDdd,
dGmtsec)

Usage: Determines the Modified Julian Date for the input year, day of year and GMT seconds.
Parameters:
iYear – year (1950-2049)
iDdd – day of year (1-365|366)
dGmtsec – GMT seconds of day (0-86399.999)
Return value:
dMjd – Modified Julian Date (33282.0 - 69806.999)

get_modJulDateUnix

(iUnixTime)

Usage: Determines the Modified Julian Date for the input UNIX time value.
(*due to limitations of Unix time, this will be valid only between 01 Jan 1970 – 19 Jan 2038*).
Parameters:
iUnixTime – Unix Time, in seconds from 01 Jan 1970, 0000 GMT; (0 – MaxInt)

Return value:
dMjd – Modified Julian Date (40587.0 - 65442.134)

get_dateTime

(dMjdValue)

Usage: Determines the year, day of year and GMT seconds for the input Modified Julian Date.
Parameters:
dMjdValue – Modified Julian Date (33282.0 - 69806.999)
Return values:
iYear – year (1950-2049)
iDdd – day of year (1-365|366)
dGmtsec – GMT seconds of day (0-86399.999)

get_hms

(dGmtsec)

Usage: Determines the hours, minutes and seconds for the input GMT seconds.
Parameters:
dGmtsec – GMT seconds of day (0-86399.999)
Return values:
iHours – hours of day (0-23)
iMinutes – minutes of hour (0-59)
dSeconds – seconds of minute (0-59.999)

get_monthDay

(iYear)

Usage: Determines the month and day number for the input year and day of year.
Parameters:
iYear – year (1950-2049)
Return values:
iDdd – day of year (1-365|366)
iMonth – month (1-12)
iDay – day of month (1-28|29|30|31)

Model-Level Python API Reference

These classes provide direct programmatic access to each of the model/processing components that comprise the CmdLineIrene application. There is no parallelized processing available at this level. Error-checking is limited to within each class (no error-checking between classes), and so is unable to detect incompatible processing operations (ie uni-directional integral flux input to dose calculations). Computer system environment variables may be used when specifying database filenames and/or directories.

EphemModel Class

Import file: ephemModel.py "`from ephemModel import EphemModel`"

This class is the entry point that provides direct programmatic access to the ephemeris generation model. Additional supporting methods are available to perform coordinate conversions and calculate associated magnetic field model parameter values.
Please note that all time values, both input and output, are in Modified Julian Date (MJD) form. Conversions to and from MJD times are available from the DateTime class, described elsewhere in this Model-Level API section.
Position coordinates are always used in sets of three values, in the coordinate system and units that are specified. Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the order of the coordinate values for non-Cartesian coordinate systems.

Ephemeris generation requires the selection of an orbit propagator (and its options), a time range and time step size, and the definition of the orbit characteristics. These orbit characteristics may be defined by either a Two-Line Element (TLE) file, or a set of orbital element values. See the User's Guide '*Orbit Propagation Inputs*' section for details about each of the available settings.

Important: The number of ephemeris entries produced by each call the *computeEphemeris*() method may be controlled by the sizing specified using the *set_chunkSize*() method. When not specified, the default behavior is to produce ephemeris for the entire period as defined in the *set_times*() method. For large sets of times, this could cause the ephemeris data to potentially occupy a sizeable amount of system memory, and potentially hinder subsequent processing.

General:

EphemModel

Usage: Instantiates an instance of the EphemModel class

Return value:
'EphemModel' class object

set_chunkSize

(iChunkSize)

Usage: Specifies the number of time and position entries that are returned from each call to the *computeEphemeris*() methods. This is useful for breaking up the ephemeris data into manageable segments for its use in other calculations. Recommended sizing is 960, or 120 on systems with limited available memory resources.
When a sizing is *not* specified, it defaults to 0, meaning the *entire set* of times specified in the

set_times() methods will be calculated and returned in a *single* call to the *computeEphemeris*() methods. For large sets of times, this could cause this data to potentially occupy a sizeable amount of system memory, and potentially hinder subsequent processing.

Parameters:

iChunkSize – number of entries in processing chunk; values lower than 60 are not recommended

Return value:

iErr – 0 = success, otherwise error

get_chunkSize

Usage: Returns the current value of the 'chunk' size, as described in previous method.

Return value:

iChunk – number of entries in processing chunk

Model Parameter Inputs:

set_modelDBDir

(strDataDir)

Usage: Specifies the directory that contains the collection IRENE model database files. The various database files required are automatically selected according to the model and parameters specified. The use of this method is highly recommended, as it *eliminates* the need for the other methods that specify the individual database files; those are only needed for using alternate or non-standard versions.

Parameters:

strDataDir – directory path for the IRENE database files.

Return value:

iErr – 0 = success, otherwise error

set_magfieldDBFile

(strDataSource)

Usage: Specifies the name of the file (including path) for the magnetic field model database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/igrfDB.h5'.

Parameters:

strDataSource – magnetic field model database filename, including path

Return value:

iErr – 0 = success, otherwise error

set_prop

(strProp)

Usage: Specifies the orbit propagator algorithm to use for the ephemeris generation.

Parameters:

strProp– *valid* values: 'SatEph', 'SGP4' or 'Kepler'.

Return value:

iErr – 0 = success, otherwise error

set_sgp4Param

(strMode,
strWGS)

Usage: Specifies the mode and WGS parameter for the SGP4 orbit propagator, if being used.
Parameters:
strMode – SGP4 propagation mode; valid values: 'Standard' or 'Improved'.
strWGS – World Geodetic System version; valid values: '72old', '72' or '84'.
Return value:
iErr – 0 = success, otherwise error

set_keplerUseJ2

(iUseJ2 = 1)

Usage: Specifies the use of the 'J2' perturbation for the Kepler orbit propagator, if being used.
Parameters:
iUseJ2 – 1 *(true)* or 0 *(false)*
Return value:
iErr – 0 = success, otherwise error

---

get_modelDBDir

Usage: Returns the directory name containing the collection of IRENE model database files that was specified in a previous call to the *set_modelDBDir*() method; otherwise, blank.
Return value:
strModelDBDir – model database directory.

get_magfieldDBFile

Usage: Returns the name of the file (including path) for the magnetic field model database. This will be available immediately, when specified using the *set_magfieldDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *compute_ephemeris*() method.
Return value:
strDataSource – magnetic field model database filename.

get_prop

Usage: Returns the orbit propagator algorithm to use for the ephemeris generation.
Return value:
strProp – orbit propagator, as specified in the *set_prop()* method

get_sgp4Mode

Usage: Returns the defined SGP4 orbit propagator mode.
Return value:
strMode – SGP4 propagation mode, as specified in the *set_sgp4Param()* method

get_sgp4Wgs

Usage: Returns the defined SGP4 orbit propagator WGS version.
Return value:

strWGS – World Geodetic System version, as specified in the *set_sgp4Param()* method

get_keplerUseJ2

Usage: Returns whether the use of the 'J2' perturbation for the Kepler orbit propagator is enabled, as specified in the *set_keplerUseJ2*() method.

Return value:

iUseJ2 – 1 *(true)* or 0 *(false)*, <0 = error

---

set_times

(dStartTime,
dEndTime,
dTimeStepSec)

Usage: Specifies the start and stop times (*inclusive*), and time step, of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file.

Parameters:

dStartTime, dEndTime – start and stop time values, in Modified Julian Date form

dTimeStepSec – time step size, in seconds

Return value:

iErr – 0 = success, otherwise error

set_varTimes

(dStartTime,
dEndTime,
dTimeMinStepSec,
dTimeMaxStepSec = *3600.0*,
dTimeRoundSec = *5.0*)

Usage: Specifies the start and stop times (*inclusive*), and variable time step limits, of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file. Variable time steps are calculated based on the orbital radial values, and are useful for the more elliptical orbits (ie eccentricity>0.25). See the User's Guide document "Orbit Ephemeris File Description" section for more information.

Parameters:

dStartTime, *dEndTime* - start and stop time values, in Modified Julian Date form

dTimeMinStepSec – lower limit of variable time steps, in seconds; must be ≥ 10 seconds

dTimeMaxStepSec – upper limit of variable time steps, in seconds; must be > min, ≤ 3600 seconds

dTimeRoundSec – rounding of variable time steps, in whole seconds; use 0 for no rounding; < min

Return value:

iErr – 0 = success, otherwise error

set_startTime

(dStartTime)

Usage: Specifies the start time of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file.

Parameters:

dStartTime – start time value, in Modified Julian Date form

Return value:

iErr – 0 = success, otherwise error

set_endTime

(dEndTime)

Usage: Specifies the and stop time of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file.

Parameters:

dEndTime – stop time value, in Modified Julian Date form

Return value:

iErr – 0 = success, otherwise error

set_timeStep

(dTimeStepSec)

Usage: Specifies the fixed time step of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file.

Parameters:

dTimeStepSec – time step size, in seconds

Return value:

iErr – 0 = success, otherwise error

set_varTimeStep

(dTimeMinStepSec,
dTimeMaxStepSec = *3600.0*,
dTimeRoundSec = *5.0*)

Usage: Specifies the variable time step limits of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file. Variable time steps are calculated based on the orbital radial values, and are useful for the more elliptical orbits (ie eccentricity>0.25). See the User's Guide document "Orbit Ephemeris File Description" section for more information.

Parameters:

dTimeMinStepSec – lower limit of variable time steps, in seconds; must be ≥ 10 seconds

dTimeMaxStepSec – upper limit of variable time steps, in seconds; must be > min, ≤ 3600 seconds

dTimeRoundSec – rounding of variable time steps, in whole seconds; use 0 for no rounding; < min

Return value:

iErr – 0 = success, otherwise error

set_timesList

(daTimes)

Usage: Specifies the set of times, in Modified Julian Date form, of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file.

Parameters:

daTimes – numpy array of chronologically ordered time values, in Modified Julian Date form

Return value:

iErr – 0 = success, otherwise error

---

get_times

Usage: Returns the start and stop times, and fixed time step, for the ephemeris generation.
Return values:
dStartTime, dEndTime – start and stop time values, in Modified Julian Date form
dTimeStepSec – time step size, in seconds

get_varTimes

Usage: Returns the start and stop times, and variable time step limits, for the ephemeris generation.
Return values:
dStartTime, *dEndTime* - start and stop time values, in Modified Julian Date form
dTimeMinStepSec – lower limit of variable time steps, in seconds
dTimeMaxStepSec – upper limit of variable time steps, in seconds
dTimeRoundSec – rounding of variable time steps, in seconds

get_startTime

Usage: Returns the start time for the ephemeris generation.
Return values:
dStartTime – start time value, in Modified Julian Date form

get_endTime

Usage: Returns the stop time for the ephemeris generation.
Return values:
dEndTime – stop time value, in Modified Julian Date form

get_timeStep

Usage: Returns the fixed time step for the ephemeris generation.
Return values:
dTimeStepSec – time step size, in seconds

get_varTimeStep

Usage: Returns the variable time step limits for the ephemeris generation.
Return values:
dTimeMinStepSec – lower limit of variable time steps, in seconds
dTimeMaxStepSec – upper limit of variable time steps, in seconds
dTimeRoundSec – rounding of variable time steps, in seconds

get_numTimesList

Usage: Returns the number of time entries defined for the ephemeris generation, from the specifications in *set_times*() or *set_timesList*() methods. Note: the number of *variable* timestep times (from *set_varTimes*() specifications) is available only when the orbital parameters are sufficiently defined.
Return value:
iNumT – number of ephemeris times defined; error if negative

void getTimesList

Usage: Returns the set of time values, in Modified Julian Date form, for the ephemeris generation, from the specifications in *set_times*() or *set_timesList*() methods. Note: the *variable* timestep times (from *set_varTimes*() specifications) are available only when the orbital parameters are sufficiently defined.

Return values:

iNumT – number of ephemeris times defined; error if negative

daTimes – vector of time values, in Modified Julian Date form

clear_timesList

Usage: Clears the time entries defined for the ephemeris generation, from the specifications in *set_times*() or *set_timesList*() methods, or set_*varTimesList*(), under certain conditions.

Return value:

iErr – 0 = success, otherwise error

TLE files are required to be in the standard NORAD format (see User's Guide, Appendix F). Use of the Kepler propagator requires that the TLE file contain only one entry. For the other propagators, the TLE may contain multiple entries (for the same satellite), which must be in chronological order.

set_tleFile

(strTLEFile)

Usage: Specifies the name of the Two-Line Element (TLE) file (including path) to use with the selected orbit propagator; this parameter is not needed if a set of orbital element values are being used instead.

Parameters:

strTLEFile – path and filename of TLE file

Return value:

iErr – 0 = success, otherwise error

get_tleFile

Usage: Returns the name of the specified TLE file, if any.

Return value:

strTLEFile – path and filename of the TLE file, as specified in the *set_tleFile*() method.

set_tle

(strTLELine1,
strTLELine2)

Usage: Specifies a pair of strings to be used as a two-line element (TLE) set that describes an orbit. It is assumed that the contents of these strings are properly formatted, as described in Appendix F of the User's Guide. Improperly formatted strings can sometimes produce valid, but unintended orbits. This routine may be called multiple times so to define multiple TLEs, provided that they are for the same object, and are added in chronologically order (with no duplicate epoch times).

Parameters:

strTLELine1 – first line of TLE set

strTLELine2 – second line of TLE set

Return value:

iErr – 0 = success, otherwise error

get_numTle

Usage: Returns the number of TLEs that were defined in calls to the set_tle() method.
Return value:
iNum – 0 or greater = number of defined TLEs, otherwise error

get_tle

(iEntry)
Usage: Returns the TLE strings corresponding to the specified entry in the list of TLEs, that were defined in calls to the set_tle() method.
Parameters:
iEntry – index of entry in list of TLEs, counting from 0.
Return values:
iErr – 0 = success, otherwise error
szTLELine1 – first line of specified TLE set
szTLELine2 – second line of specified TLE set

int reset_tleInputs

Usage: Clears the list of TLEs that were defined in calls to the set_tle() method.
Return value:
iErr – 0 = success, otherwise error

The orbital element values to be specified depend on the type of orbit and/or available orbit definition references. Their use requires an associated element time to also be specified. See the User's Guide document "Orbiter Propagation Inputs" section for more details.

set_elementTime

(dElementTime)
Usage: Specifies the 'epoch' time associated with the set of orbital element values.
Parameters:
dElementTime – time, in Modified Julian Date form
Return value:
iErr – 0 = success, otherwise error

set_inclination

(dInclination)
Usage: Specifies the orbital element 'Inclination' value.
Parameters:
dInclination – orbit inclination angle, in degrees (0-180)
Return value:
iErr – 0 = success, otherwise error

set_rightAscension

(dRtAscOfAscNode)
Usage: Specifies the orbital element 'Right Ascension of the Ascending Node' value.
Parameters:
dRtAscOfAscNode – orbit ascending node position, in degrees (0-360)

Return value:
iErr – 0 = success, otherwise error

set_eccentricity

(dEccentricity)
Usage: Specifies the orbital element 'Eccentricity' value.
Parameters:
dEccentricity – orbit eccentricity value, unitless (0-<1)
Return value:
iErr – 0 = success, otherwise error

set_argOfPerigee

(dArgOfPerigee)
Usage: Specifies the orbital element 'Argument of Perigee' value.
Parameters:
dArgOfPerigee – orbit perigee position, in degrees (0-360)
Return value:
iErr – 0 = success, otherwise error

set_meanAnomaly

(dMeanAnomaly)
Usage: Specifies the orbital element 'Mean Anomaly' value.
Parameters:
dMeanAnomaly – orbit mean anomaly value, in degrees (0-360)
Return value:
iErr – 0 = success, otherwise error

set_meanMotion

(dMeanMotion)
Usage: Specifies the orbital element 'Mean Motion' value.
Parameters:
dMeanMotion – orbit mean motion value, in units of revolutions per day (>0)
Return value:
iErr – 0 = success, otherwise error

set_meanMotion1stDeriv

(dMeanMotion1stDeriv)
Usage: Specifies the orbital element 'First Time Derivative of the Mean Motion' value (this should NOT be divided by 2, as when specified in a TLE); this value is only used by the SatEph propagator.
Parameters:
dMeanMotion1stDeriv – first derivative of mean motion, in units of revs per day^2 (-10 – 10)
Return value:
iErr – 0 = success, otherwise error

set_meanMotion2ndDeriv

(dMeanMotion2ndDeriv)

Usage: Specifies the orbital element 'Second Time Derivative of the Mean Motion' value (this should NOT be divided by 6, as when specified in a TLE); this value is only used by the SatEph propagator.

Parameters:

dMeanMotion2ndDeriv – second derivative of mean motion, in units of revs per day^3 (-1 – 1)

Return value:

iErr – 0 = success, otherwise error

set_bStar

(dBStar)

Usage: Specifies the orbital element 'B*' value, for modelling satellite drag effects; this value is only used by the SGP4 propagator.

Parameters:

dBStar – ballistic coefficient value (-1 – 1)

Return value:

iErr – 0 = success, otherwise error

set_altitudeOfApogee

(dAltApogee)

Usage: Specifies the orbital element 'Apogee Altitude' value (furthest distance).

Parameters:

dAltApogee – altitude (in km) above the Earth's surface at the orbit's apogee (>0, but <~20Re)

Return value:

iErr – 0 = success, otherwise error

set_altitudeOfPerigee

(dAltPerigee)

Usage: Specifies the orbital element 'Perigee Altitude' value (closest distance).

Parameters:

dAltPerigee – altitude (in km) above the Earth's surface at the orbit's perigee (>0, but <~20Re)

Return value:

iErr – 0 = success, otherwise error

set_localTimeOfApogee

(dLocTimeApogee)

Usage: Specifies the local time of the orbit's apogee.

Parameters:

dLocTimeApogee – local time, in hours (0-24)

Return value:

iErr – 0 = success, otherwise error

set_localTimeMaxInclination

(dLocTimeMaxIncl)

Usage: Specifies the local time of the orbit's maximum inclination (ie max latitude).

Parameters:

dLocTimeMaxIncl – local time, in hours (0-24)

Return value:

iErr – 0 = success, otherwise error

set_timeOfPerigee

(dTimeOfPerigee)

Usage: Specifies the time of the orbit's perigee, as an alternative to the Mean Anomaly specification. *Any Mean Anomaly value also specified will be overridden by this value*.

Parameters:

dTimeOfPerigee – time, in Modified Julian Date form, for orbit perigee

Return value:

iErr – 0 = success, otherwise error

set_semiMajorAxis

(dSemiMajorAxis)

Usage: Specifies the orbit's semi-major axis length.

Parameters:

dSemiMajorAxis – semi-major axis length (1-75), in units of Re (radius of Earth = 6371.2 km)

Return value:

iErr – 0 = success, otherwise error

set_geosynchLon

(dGeosynchLon)

Usage: Specifies the geographic longitude of satellite in a geosynchronous orbit

Parameters:

dGeosynchLon – longitude, in degrees (-180 – 360)

Return value:

iErr – 0 = success, otherwise error

set_stateVectors

(daPos,
daVel)

Usage: Specifies the satellite's position and velocity in the GEI coordinate system at the element's 'epoch' time. Alternatively, the *set_positionGEI*() and *set_velocityGEI*() methods could be used instead.

Parameters:

daPos – numpy array containing the GEI coordinate system satellite position values (X,Y,Z), in km

daVel – numpy array containing the GEI coordinate system satellite velocity values (X,Y,Z), in km/sec

Return value:

iErr – 0 = success, otherwise error

set_positionGEI

(dPosX,
dPosY,
dPosZ)

Usage: Specifies the satellite's position in the GEI coordinate system at the element's 'epoch' time. This must be used in conjunction with the *set_velocityGEI*() method.

Parameters:

dPosX, dPosY, dPosZ – GEI coordinate system satellite position values, in km (>1Re, but <~75Re)

Return value:
iErr – 0 = success, otherwise error

set_velocityGEI

(dVelX,
dVelY,
dVelZ)

Usage: Specifies the satellite's velocity in GEI coordinate system at the element's 'epoch' time. This must be used in conjunction with the *set_positionGEI*() method.

Parameters:
dVelX, dVelY, dVelZ – GEI coordinate system satellite velocity values, in km/sec

Return value:
iErr – 0 = success, otherwise error

reset_orbitParameters

Usage: Resets all parameters that specify the orbit definition to their default values. These include the various orbital element values, element time, state arrays and TLE specifications.

Return value:
iErr – 0 = success, otherwise error

---

get_elementTime

Usage: Returns the 'epoch' time associated with the set of orbital element values.

Return value:
dElementTime – time, in Modified Julian Date form, as specified in the *set_elementTime*() method

get_inclination

Usage: Specifies the orbital element 'Inclination' value.

Return value:
dInclination – orbit inclination angle, in degrees (0-180) , as specified in the *set_inclination*() method

get_rightAscension

Usage: Returns the orbital element 'Right Ascension of the Ascending Node' value.

Return value:
dRtAscOfAscNode – orbit ascending node position, in degrees (0-360) , as specified in the *set_rightAscension*() method

get_eccentricity

Usage: Returns the orbital element 'Eccentricity' value.

Return value:
dEccentricity – orbit eccentricity value, unitless (0-<1) , as specified in the *set_eccentricity*() method

get_argOfPerigee

Usage: Returns the orbital element 'Argument of Perigee' value.

Return value:

dArgOfPerigee – orbit perigee position, in degrees (0-360) , as specified in the *set_argOfPerigee*() method

get_meanAnomaly

Usage: Returns the orbital element 'Mean Anomaly' value.
Return value:
dMeanAnomaly – orbit mean anomaly value, in degrees (0-360), as specified in the *set_meanAnomaly*() method

get_meanMotion

Usage: Returns the orbital element 'Mean Motion' value.
Return value:
dMeanMotion – orbit mean motion value, in units of revolutions per day (>0), as specified in the *set_meanMotion*() method

get_meanMotion1stDeriv

Usage: Returns the orbital element 'First Time Derivative of the Mean Motion' value.
Return value:
dMeanMotion1stDeriv – first derivative of mean motion, in units of revs per day^2 (-10 – 10), as specified in the *set_meanMotion1stDeriv*() method

get_meanMotion2ndDeriv

Usage: Returns the orbital element 'Second Time Derivative of the Mean Motion' value.
Return value:
dMeanMotion2ndDeriv – second derivative of mean motion, in units of revs per day^3 (-1 – 1), as specified in the *set_meanMotion2ndDeriv*() method

get_bStar

Usage: Returns the orbital element 'B*' value, for modelling satellite drag effects; this value is only used by the SGP4 propagator.
Return value:
dBStar – ballistic coefficient value (-1 – 1), as specified in the *set_bStar*() method

get_altitudeOfApogee

Usage: Returns the orbital element 'Apogee Altitude' value (furthest distance).
Return value:
dAltApogee – altitude (in km) above the Earth's surface at the orbit's apogee (>0), as specified in the *set_altitudeOfApogee*() method

get_altitudeOfPerigee

Usage: Returns the orbital element 'Perigee Altitude' value (closest distance).
Return value:
dAltPerigee – altitude (in km) above the Earth's surface at the orbit's perigee (>0), as specified in the *set_altitudeOfPerigee*() method

get_localTimeOfApogee

Usage: Returns the local time of the orbit's apogee.
Return value:
dLocTimeApogee – local time, in hours (0-24), as specified in the *set_localTimeOfApogee*() method

get_localTimeMaxInclination

Usage: Returns the local time of the orbit's maximum inclination (ie max latitude).
Return value:
dLocTimeMaxIncl – local time, in hours (0-24), as specified in the *set_localTimeMaxInclination*() method

get_timeOfPerigee

Usage: Returns the time of the orbit's perigee, as an alternative to the Mean Anomaly specification. *Any Mean Anomaly value also specified will be overridden by this value*.
Return value:
dTimeOfPerigee – time, in Modified Julian Date form, for orbit perigee, as specified in the *set_timeOfPerigee*() method

get_semiMajorAxis

Usage: Returns the orbit's semi-major axis length.
Return value:
dSemiMajorAxis – semi-major axis length (1-10), in units of Re (radius of Earth = 6371.2 km), as specified in the *set_semiMajorAxis*() method

get_geosynchLon

Usage: Returns the geographic longitude of satellite in a geosynchronous orbit
Return value:
dGeosynchLon – longitude, in degrees (-180 – 360), as specified in the *set_geosynchLon*() method

get_stateVectors

Usage: Returns the satellite's position and velocity in the GEI coordinate system at the element's 'epoch' time, as specified in either the *set_stateVectors*() or the *set_positionGei*() and *set_velocityGei*() methods.
Return value:
daPos – numpy array containing the GEI coordinate system satellite position values (X,Y,Z), in km
daVel – numpy array containing the GEI coordinate system satellite velocity values (X,Y,Z), in km/sec

get_positionGEI

Usage: Returns the satellite's position in the GEI coordinate system at the element's 'epoch' time.
Return value:
dPosX, dPosY, dPosZ – GEI coordinate system satellite position values, in km

get_velocityGEI

Usage: Returns the satellite's velocity in GEI coordinate system at the element's 'epoch' time.
Return value:
dVelX, dVelY, dVelZ – GEI coordinate system satellite velocity values, in km/sec

set_mainField

(strMainField)

Usage: Defines the main magnetic field model to be used in coordinate conversions and magnetic field model calculations. The IRENE standard uses the 'FastIGRF' model, and is the default setting. Note that the use of the dipole main field models require the external field model to be set to 'None'.

Parameters:

strMainField – identifier for the 'main' magnetic field model:
'FastIGRF', 'IGRF', 'OffsetDipole' | 'Offset', 'TitltedDipole' | 'Tilted'

Return value:

iErr – 0 = success, otherwise error

set_externalField

(strExternalField)

Usage: Defines the external magnetic field model to be used in coordinate conversions and magnetic field model calculations. The IRENE standard uses the 'OlsonPfitzer' model, and is the default setting. Note that the use of the dipole main field models require the external field model to be set to 'None'. The use of the Tsyganenko89 external field model also requires the specification of the Kp index value, defined with the *set_kpValue*() or *set_kpValues*() methods

Parameters:

strExternalField – identifier for the 'external' magnetic field model:
'None', 'OlsonPfitzer' | 'OP', 'Tsyganenko89' | 'Tsyg89' | 'T89'

Return value:

iErr – 0 = success, otherwise error

get_mainField

Usage: Returns the identifier for the 'main' magnetic field model that was specified in a previous call to the *set_mainField*() method; otherwise, the default setting.

Return value:

strMainField – 'main' magnetic field model identifier.

get_externalField

Usage: Returns the identifier for the 'external' magnetic field model that was specified in a previous call to the *set_externalField*() method; otherwise, the default setting.

Return value:

strExternalField – 'external' magnetic field model identifier.

set_kpValue

(dKpVal)

Usage: Specifies the constant Kp index value to be used in all subsequent calculations (as needed). Any previously defined list of time history Kp values using the *set_kpValues*() method is cleared.

Parameters:

dKpVal – Kp Index value. Valid range = 0.0 – 9.0.

Return value:

iErr – 0 = success, otherwise error

int set_kpValues

(dRefTime,
daKpVals)

Usage: Defines a list of time history of three-hour Kp index values to be used in all subsequent calculations (as needed); the appropriate Kp index value will be determined from the current calculation's time setting vs the specified reference time. The first Kp index value will be used when the current time is prior to the reference time; the last Kp index value will be used when the current time is beyond the defined time history limit. This automated determination of Kp supersedes any previously defined constant Kp value using the *set_kpValue*() method.

Parameters:

dRefTime - reference time value, in Modified Julian Date form; must be at 0000 GMT of day.

daKpVals – numpy array with time history of three-hour Kp index values. Valid range = 0.0 – 9.0.

Return value:

iErr – 0 = success, otherwise error

double getKpValue

Usage: Returns the current defined Kp value. This may be the constant value specified in the *set_kpValue*() method, or the appropriate entry from the time history value defined in the *set_kpValues*() method, according to the most recently used calculation time value.

Return value:

iErr – 0 = success, otherwise error

dKpVal – current Kp index value

get_kpValuesRefTime

Usage: Returns the reference time of the currently defined time history of Kp values, as specified in the *set_kpValues*() method. A value of -1.0 will be returned if no time history is defined.

Return value:

Return value:

iErr – 0 = success, otherwise error

dRefTime – reference time, in MJD form, for the current Kp index history.

get_kpValuesEndTime

Usage: Returns the end time of the currently defined time history of Kp values, as specified in the *set_kpValues*() method. A value of -1.0 will be returned if no time history is defined.

Return values:

iErr – 0 = success, otherwise error

dEndTime – end time, in MJD form, for the current Kp index history.

Model Execution and Results:

The ephemeris computation requires that a propagator model be selected, the magnetic field database be specified (used for coordinate conversions), an ephemeris generation time range and (fixed or variable) time step be defined (or list of discrete times), and the orbit described by either using TLEs or an appropriate set of element values and reference time.

computeGei

Usage: Returns the generated ephemeris information in the GEI coordinate system, for the current time segment. The number of entries that are returned from each call to the *computeEphemeris*() method is specified using the *set_chunkSize*() method. If this 'chunk' size is not specified, or set to '0', the ephemeris for the entire time period, defined by the *set_times*() methods, is returned in a single call. If a great number of ephemeris entries are requested in a single call, the amount of memory required (automatically allocated) may become excessive and/or hinder further processing. When a non-zero 'chunk' size is specified, multiple calls to this method may be required for accessing the ephemeris information for the entire time period; however, this provides the ephemeris information in manageable segments, for its use in other subsequent calculation tasks. This segmentation enables the processing performance to be tuned to the available system memory.

Return values:

iNum – Number of ephemeris records returned (≥0 = success, <0 = error)

daTimes – numpy array of ephemeris time values, in Modified Julian Date form

daXPos, daYPos, daZPos – numpy arrays of ephemeris position values, in the 'GEI' coordinate system and units of 'km'

daXVel, daYVel, daZVel – numpy arrays of ephemeris velocity values, in the 'GEI' coordinate system and units of 'km/sec'

compute

(strCoordSys,
strCoordUnits)

Usage: Returns the generated ephemeris information in the coordinate system and units specified, for the current time segment. The number of entries that are returned from each call to the *computeEphemeris*() method is specified using the *set_chunkSize*() method. If this 'chunk' size is not specified, or set to '0', the ephemeris for the entire time period, defined by the *set_times*() methods, is returned in a single call. If a great number of ephemeris entries are requested in a single call, the amount of memory required (automatically allocated) may become excessive and/or hinder further processing. When a non-zero 'chunk' size is specified, multiple calls to this method may be required for accessing the ephemeris information for the entire time period; however, this provides the ephemeris information in manageable segments, for its use in other subsequent calculation tasks. This segmentation enables the processing performance to be tuned to the available system memory.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value. 1 Re = 6371.2 km.

Return values:

iNum – Number of ephemeris records returned (≥0 = success, <0 = error)

daTimes – numpy array of ephemeris time values, in Modified Julian Date form

daCoord1, daCoord2, daCoord3 – numpy arrays of ephemeris position values, in the coordinate system and units specified.
Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the 'standard' ordering of the returned coordinate values for non-Cartesian coordinate systems.

restart

Usage: When the 'chunk' size, specified using the *set_chunkSize*() method, is larger than 0, this method explicitly resets the *computeEphemeris*() methods to begin the ephemeris generation at the previously defined 'start time' at their next call. This reset is done automatically when the chunk size is changed, or any of the orbital element parameters or propagator settings is modified.
Return value:
iErr – 0 = success, otherwise error

convertCoords

(strCoordSysIn,
strCoordUnitsIn,
daTimes,
daCoord1In,
daCoord2In,
daCoord3In,
strCoordSysOut,
strCoordUnitsOut)

Usage: Converts the set of input times, position coordinates from one coordinate system to another.
Parameters:
strCoordSysIn – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL';
Please consult the User's Guide document, "Supported Coordinate Systems" for more details.
strCoordUnitsIn – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.
daTimes – numpy array of time values, in Modified Julian Date form
daCoord1In, daCoord2In, daCoord3In – numpy arrays of position values, in the coordinate system and units specified by the *strCoordSysIn* and *strCoordUnitsIn* parameter values.
Consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' ordering of the input and output coordinate values for non-Cartesian coordinate systems.
strCoordSysOut – 'new' coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.
strCoordUnitsOut – 'new' units: 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.
Return values:
iErr – 0 = success, otherwise error
daCoord1Out, daCoord2Out, daCoord3Out – numpy arrays of position values, in the 'new' coordinate system and units specified by the *strCoordSysOut* and *strCoordUnitsOut* parameter values.

convertCoordsSingle

(strCoordSysIn,
strCoordUnitsIn,
dTime,
dCoord1,

dCoord2,
dCoord3,
strCoordSysOut,
strCoordUnitsOut)

Usage: Converts a single input time, position coordinates from one coordinate system to another.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordSysIn – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

dTimes – time value, in Modified Julian Date form

dCoord1, dCoord2, dCoord3 – position values, in the coordinate system and units specified by the *strCoordSysIn* and *strCoordUnitsIn* parameter values.

Consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' ordering of the input and output coordinate values for non-Cartesian coordinate systems.

strCoordSysOut – 'new' coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnitsOut – 'new' units: 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

Return values:

iErr – 0 = success, otherwise error

dCoord1Out, dCoord2Out, dCoord3Out – position values, in the 'new' coordinate system and units specified by the *strCoordSysOut* and *strCoordUnitsOut* parameter values.

compute_bField

(strCoordSys,
strCoordUnits,
daTimes,
daCoord1,
daCoord2,
daCoord3)

Usage: Calculates several magnetic field parameters for the specified set of time and position inputs. Results will be affected by specifications for the 'main' and 'external' magnetic field model, using the *set_mainField*() and *set_externalField*() methods, respectively. When the 'Tsyg89' external field model is being used, an additional specification of the Kp index value will be required, using the *set_kpValue*() or *set_kpValues*() methods. Depending on the position(s) specified, some or all return values may not be able to be determined; when this occurs, *fill* values are returned instead (ie, -1.0e31 or ±100.0). In specific cases, the returned valid Lm value(s) may be negated; this indicates that the underlying calculations included a magnetic field line trace that briefly went subterranean.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

daTimes – numpy array of time values, in Modified Julian Date form

daCoord1, daCoord2, daCoord3 – numpy arrays of position values, in the coordinate system and units specified by the *strCoordSys* and *strCoordUnit* parameter values.

Consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' ordering of the input and output coordinate values for non-Cartesian coordinate systems.

Return values:

iErr – 0 = success, otherwise error

da2BVecGeo – 2-dimensional numpy array of B vector component values [nT], at each of the times+positions specified; B vectors are in the GEO coordinate system. [time,coord]

daBMag – numpy array of B vector magnitude values [nT], at each of the times+positions specified.

daBMin – numpy array of B minimum vector magnitude values [nT], at the magnetic equator associated with each of the times+positions specified.

daLm – numpy array of Lshell values [unitless], associated with each of the times+positions specified.

compute_bFieldSingle

(strCoordSys,
strCoordUnits,
dTime,
dCoord1,
dCoord2,
dCoord3)

Usage: Calculates several magnetic field parameters for the single time and position input. Results will be affected by specifications for the 'main' and 'external' magnetic field model, using the *set_mainField*() and *set_externalField*() methods, respectively. When the 'Tsyg89' external field model is being used, an additional specification of the Kp index value will be required, using the *set_kpValue*() or *set_kpValues*() methods. Depending on the position specified, some or all return values may not be able to be determined; when this occurs, *fill* values are returned instead (ie, -1.0e31 or ±100.0). In specific cases, the returned valid Lm value may be negated; this indicates that the underlying calculations included a magnetic field line trace that briefly went subterranean.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

dTime – time value, in Modified Julian Date form

dCoord1, dCoord2, dCoord3 – position values, in the coordinate system and units specified by the *strCoordSys* and *strCoordUnit* parameter values.

Consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' ordering of the input and output coordinate values for non-Cartesian coordinate systems.

Return values:

iErr – 0 = success, otherwise error

daBVecGeo – numpy array of B vector component values [nT], at the time+position specified; B vector is in the GEO coordinate system.

dBMag – B magnitude value [nT], at the time+position specified.

dBMin – B minimum vector magnitude value [nT], at the magnetic equator associated with the time+position specified.

dLm – Lshell value [unitless], associated with the time+position specified.

compute_invariants

(strCoordSys,
strCoordUnits,
daTimes,
daCoord1,
daCoord2,
daCoord3,
daPitchAngles)

Usage: Calculates magnetic field parameters as a function of pitch angle for the specified set of time and position inputs. Results will be affected by specifications for the 'main' and 'external' magnetic field model, using the *set_mainField*() and *set_externalField*() methods, respectively. When the 'Tsyg89' external field model is being used, an additional specification of the Kp index value will be required, using the *set_kpValue*() or *set_kpValues*() methods. Depending on the position(s) specified, some or all return values may not be able to be determined; when this occurs, *fill* values are returned instead (ie, -1.0e31, ±100.0 or -1.0). In specific cases, the returned valid Lm value(s) may be negated; this indicates that the underlying calculations included a magnetic field line trace that briefly went subterranean.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

daTimes – numpy array of time values, in Modified Julian Date form

daCoord1, daCoord2, daCoord3 – numpy arrays of position values, in the coordinate system and units specified by the *strCoordSys* and *strCoordUnit* parameter values.
Consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' ordering of the input and output coordinate values for non-Cartesian coordinate systems.

daPitchAngles – numpy array of pitch angles, in degrees (0-90). Must be in *descending* order.

Return values:

iErr – 0 = success, otherwise error

daBMin – numpy array of B minimum vector magnitude values [nT], at the magnetic equator associated with each of the times+positions specified.

da2BMinPosGeo – 2-dimensional numpy array of B minimum positions, associated with each of the times+positions specified; BMin positions are in the GEO/km coordinate system. [time,coord]

da2BVecGeo – 2-dimensional numpy array of B vector component values [nT], at each of the times+positions specified; B vectors are in the GEO coordinate system. [time,coord]

da2Lm – 2-dimensional numpy array of Lshell values [unitless], associated with each of the times+positions specified, for each of the pitch angles specified. [time,pitchAngle]

da2I – 2-dimensional numpy array of "I" values [unitless], associated with each of the times+positions specified, for each of the pitch angles specified. [time,pitchAngle]

compute_invariantsSingle

(strCoordSys,
strCoordUnits,
dTime,
dCoord1,
dCoord2,
dCoord3,

daPitchAngles)

Usage: Calculates magnetic field parameters as a function of pitch angle for the specified time and position input. Results will be affected by specifications for the 'main' and 'external' magnetic field model, using the *set_mainField*() and *set_externalField*() methods, respectively. When the 'Tsyg89' external field model is being used, an additional specification of the Kp index value will be required, using the *set_kpValue*() or *set_kpValues*() methods. Depending on the position specified, some or all return values may not be able to be determined; when this occurs, *fill* values are returned instead (ie, -1.0e31, ±100.0 or -1.0). In specific cases, the returned valid Lm value(s) may be negated; this indicates that the underlying calculations included a magnetic field line trace that briefly went subterranean.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

dTime – time value, in Modified Julian Date form

dCoord1, dCoord2, dCoord3 – position values, in the coordinate system and units specified by the *strCoordSys* and *strCoordUnit* parameter values.
Consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' ordering of the input and output coordinate values for non-Cartesian coordinate systems.

daPitchAngles – numpy array of pitch angles, in degrees (0-90). Must be in *descending* order.

Return values:

iErr – 0 = success, otherwise error

dBMin – B minimum vector magnitude value [nT], at the magnetic equator associated with the time+position specified.

daBMinPosGeo – numpy array of B minimum position, associated with the time+position specified; BMin position is in the GEO/km coordinate system.

daBVecGeo – numpy array of B vector component values [nT], at the time+position specified; B vector is in the GEO coordinate system.

daLm – numpy array of Lshell values [unitless], associated with the time+position specified, for each of the pitch angles specified.

daI – numpy array of "I" values [unitless], associated with the times+positions specified, for each of the pitch angles specified.

Ae9Ap9Model Class

Import file: ae9ap9Model.py "`from ae9ap9Model import Ae9Ap9Model`"

This class is the entry point that provides direct programmatic access to the Ae9, Ap9 and SPM model. Please note that all time values, both input and output, are in Modified Julian Date (MJD) form. Conversions to and from MJD times are available from the DateTime class, described elsewhere in this Model-Level API section. Position coordinates are always used in sets of three values, in the coordinate system and units that are specified. Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the order of the coordinate values for non-Cartesian coordinate systems.

General:

Ae9Ap9Model

Usage: Instantiates an instance of the Ae9Ap9Model class
Return value:
'Ae9Ap9Model' class object

Model Parameter Inputs:

set_model

(strModel)

Usage: Specifies the name of the flux model to be used in the calculations. Note the 'Plasma' model names now includes the species type.
See the following 'Legacy Model Parameter Inputs' section for Legacy model-specific options.
Parameters:
strModel – model name: 'AE9', 'AP9', 'PlasmaE', 'PlasmaH', 'PlasmaHe' or 'PlasmaO'
Legacy models: 'AE8', 'AP8', 'CRRESELE', 'CRRESPRO' or 'CAMMICE'
Return value:
iErr – 0 = success, otherwise error

set_modelDBDir

(strDataDir)

Usage: Specifies the directory that contains the collection IRENE model database files. The various database files required are automatically selected according to the model and parameters specified. The use of this method is highly recommended, as it *eliminates* the need for the other methods that specify the individual database files; those are only needed for using alternate or non-standard versions.
Parameters:
strDataDir – directory path for the IRENE database files.
Return value:
iErr – 0 = success, otherwise error

set_modelDBFile

(strModelDBFile)

Usage: Specifies the name of the database file (including path) for flux model calculations. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

Please consult the User's Guide for the exact database filename associated with each model. Once initialized via calls to either *loadModelDB*() or *set_fluxEnvironment*() methods, the specified model database *cannot* be changed. For best results, use separate model objects for different model species.

Parameters:

strModelDBFile – model database filename, including path

Return value:

iErr – 0 = success, otherwise error

set_kPhiDBFile

(strKPhiDBFile)

Usage: Specifies the name of the file (including path) for the K/Phi database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/fastPhi_net.mat'.

Parameters:

strDataSource – database filename, including path

Return value:

iErr – 0 = success, otherwise error

set_kHMinDBFile

(strKHMinDBFile)

Usage: Specifies the name of the file (including path) for the K/Hmin database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/fast_hmin_net.mat'.

Parameters:

strDataSource – database filename, including path

Return value:

iErr – 0 = success, otherwise error

set_magfieldDBFile

(strMagfieldDBFile)

Usage: Specifies the name of the file (including path) for the magnetic field model database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/igrfDB.h5'.

Parameters:

strDataSource – magnetic field model database filename, including path

Return value:

iErr – 0 = success, otherwise error

load_modelDB

Usage: Performs the initial loading of information from the model database file specified in the *set_modelDBFile*() method. Use of this method is optional, as it is called internally on the initial call to

the *set_fluxEnvironment*() method. However, it is required if the *get_model[Name|Species]*() methods are called before the initial call to *set_fluxEnvironment*().
Once initialized, these model databases specifications *cannot* be changed.

Return value:

iErr – 0 = success, otherwise error

---

get_modelName

Usage: Returns the name of the model described by the model database file, either specified in a previous call to *set_modelDBFile*() method, or automatically determined using the specification in calls to the *set_model*() and *set_modelDBDir*() methods. This returned model name is available only after a call to the *load_modelDB*() or *set_fluxEnviron**() methods.

Return value:

strModelName – name of model associated with database ('AE9', 'AP9', 'SPME', 'SPMHE' or 'SPMO')

get_modelSpecies

Usage: Returns the name of the particle species of the model database file, either specified in the previous call to *set_modelDBFile*() method, or automatically determined using the specification in calls to the *set_model*() and *set_modelDBDir*() methods. This returned species name is available only after a call to either the *load_modelDB*() or *set_fluxEnviron**() methods.

Return value:

strModelSpecies – name of species of the associated with database ('e-', 'H+', 'He+' or 'O+')

get_model

Usage: Returns the name of the flux model, as specified in the *set_model*() method.

Return value:

strModel – name of model.

get_modelDBDir

Usage: Returns the directory name containing the collection of IRENE model database files that was specified in a previous call to the *set_modelDBDir*() method; otherwise, blank.

Return value:

strModelDBDir – model database directory.

get_modelDBFile

Usage: Returns the name of the database file (including path) for flux model calculations. This will be available immediately, when specified using the *set_modelDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *load_modelDB*() or *set_fluxEnviron**() methods.

Return value:

strModelDBFile – model database filename, including path.

get_kPhiDBFile

Usage: Returns the name of the file (including path) for the K/Phi database. This will be available immediately, when specified using the *set_kPhiDBFile*() method. When the *setModelDBDir*() method is

used, the automatically determined filename will be available after a call to the *load_modelDB*() or *set_fluxEnviron**() methods.

Return value:

strDataSource – database filename, including path.

get_kHMinDBFile

Usage: Returns the name of the file (including path) for the K/Hmin database. This will be available immediately, when specified using the *set_kHMinDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *load_modelDB*() or *set_fluxEnviron**() methods.

Return value:

strDataSource – database filename, including path.

get_magfieldDBFile

Usage: Returns the name of the file (including path) for the magnetic field model database. This will be available immediately, when specified using the *set_magfieldDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *load_modelDB*() or *set_fluxEnviron**() methods.

Return value:

strDataSource – magnetic field model database filename, including path.

Model Execution and Results:

The *set_fluxEnviron**() methods are used to specify the ephemeris (time and position) and particle energies for the flux calculation of the Ae9Ap9 model. The multiple versions of this method provide different ways to define the flux particle direction(s) – omnidirection (default), fixed (over time) or variable pitch angles, or explicit direction arrays. The various *computeFlyin**() (or *computeFlux**()) methods return the respective types of flux values using the most recently defined 'flux environment' specifications.

The returned flux values are in units of [#/cm^2/sec] (for integral) or [#/cm^2/sec/MeV] (for differential).

For best model performance, it is recommended that the amount of ephemeris information being supplied as input to the *set_fluxEnviron**() methods be moderated. This method performs numerous calculations on each time, position coordinate and pitch angle(s) combination, retaining these intermediate results in additional internally allocated memory. To avoid stressing the system memory resources, limit the number of entries in the time and coordinate arrays for each call: a value of 120 is advised for systems with limited memory, 960 for typical systems, but no larger than 2400, even with ample amounts available memory. The subsequent call to the needed *computeFlyin[type]*() method(s) will return fluxes for the current ephemeris segment.

The ephemeris information produced by the *EphemModel.compute*() can be segmented through the use of that class's *set_chunkSize*() method. The segmentation of ephemeris information from other sources will need to be accomplished by the calling process.

set_fluxEnvironOmni

(strFluxType,
daEnergies,

daEnergies2,
daTimes,
strCoordSys,
strCoordUnits,
daCoords1,
daCoords2,
daCoords3)

Usage: Specifies the flux type, energies and ephemeris time and positions to be used for *omni-directional* flux model calculations. Note that use of '2PtDiff' flux type requires that both sets of energy values to be specified. Please consult User's Guide for the valid energy ranges/values, which depend on the model being used.

Parameters:

strFluxType – flux type identifier: '1PtDiff', '2PtDiff' or 'Integral'

daEnergies – numpy array of energy values [MeV]; if '2PtDiff' flux type, these specify lower bounds of energy bins.

daEnergies2 – ignored unless flux type is '2ptDiff'; array of energy values [MeV], specifying upper bounds of energy bins

daTimes – numpy array of time values, in Modified Julian Date form. May be identical times or times in chronological order, associated with position coordinates.

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

daCoords1, daCoords2, daCoords3 – numpy arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system and units specified by the strCoordSys and strCoordUnits parameters. Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' order of the coordinate values for non-Cartesian coordinate systems.

Return value:

iErr – 0 = success, otherwise error

set_fluxEnvironFixPitch

(strFluxType,
daEnergies,
daEnergies2,
daTimes,
strCoordSys,
strCoordUnits,
daCoords1,
daCoords2,
daCoords3,
daPitchAngles)

Usage: Specifies the flux type, energies and ephemeris time and positions, with a *fixed* set of pitch angles, to be used for *uni-directional* flux model calculations. Note that use of '2PtDiff' flux type requires that both sets of energy values to be specified. Please consult User's Guide for the valid energy ranges/values, which depend on the model being used.

Parameters:

strFluxType – flux type identifier: '1PtDiff', '2PtDiff' or 'Integral'

daEnergies – numpy array of energy values [MeV]; if ‘2PtDiff’ flux type, these specify lower bounds of energy bins.

daEnergies2 – ignored unless flux type is ‘2ptDiff’; array of energy values [MeV], specifying upper bounds of energy bins

daTimes – numpy array of time values, in Modified Julian Date form. May be identical times or times in chronological order, associated with position coordinates.

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

daCoords1, daCoords2, daCoords3 – numpy arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system and units specified by the strCoordSys and strCoordUnits parameters. Please consult the User’s Guide document, “Supported Coordinate Systems” for more details; in particular, note the expected ‘standard’ order of the coordinate values for non-Cartesian coordinate systems.

daPitchAngles – numpy array of pitch angles, in degrees (0-180); to be used at each time/position

Return value:

iErr – 0 = success, otherwise error

set_fluxEnvironVarPitch

(strFluxType,
daEnergies,
daEnergies2,
daTimes,
strCoordSys,
strCoordUnits,
daCoords1,
daCoords2,
daCoords3,
da2PitchAngles)

Usage: Specifies the flux type, energies and ephemeris time and positions, with a *varying* set of pitch angles, to be used for *uni-directional* flux model calculations. Note that use of ‘2PtDiff’ flux type requires that both sets of energy values to be specified. Please consult User’s Guide for the valid energy ranges/values, which depend on the model being used.

Parameters:

strFluxType – flux type identifier: ‘1PtDiff’, ‘2PtDiff’ or ‘Integral’

daEnergies – numpy array of energy values [MeV]; if ‘2PtDiff’ flux type, these specify lower bounds of energy bins.

daEnergies2 – ignored unless flux type is ‘2ptDiff’; array of energy values [MeV], specifying upper bounds of energy bins

daTimes – numpy array of time values, in Modified Julian Date form. May be identical times or times in chronological order, associated with position coordinates.

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

daCoords1, daCoords2, daCoords3 – numpy arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system and units specified by the strCoordSys and strCoordUnits parameters. Please consult the User’s Guide document, “Supported

Coordinate Systems" for more details; in particular, note the expected 'standard' order of the coordinate values for non-Cartesian coordinate systems.

da2PitchAngles – 2-dimensional numpy array of pitch angles, in degrees (0-180). [time,pitch angles]

Return value:

iErr – 0 = success, otherwise error

set_fluxEnvironDirVec

(strFluxType,
daEnergies,
daEnergies2,
daTimes,
strCoordSys,
strCoordUnits,
daCoords1,
daCoords2,
daCoords3,
daFluxDir1,
daFluxDir2,
daFluxDir3)

Usage: Specifies the flux type, energies and ephemeris time and positions, at a set of *varying* direction arrays, to be used for *uni-directional* flux model calculations. Note that use of '2PtDiff' flux type requires that both sets of energy values to be specified. Please consult User's Guide for the valid energy ranges/values, which depend on the model being used.

Parameters:

strFluxType – flux type identifier: '1PtDiff', '2PtDiff' or 'Integral'

daEnergies – numpy array of energy values [MeV]; if '2PtDiff' flux type, these specify lower bounds of energy bins.

daEnergies2 – ignored unless flux type is '2ptDiff'; array of energy values [MeV], specifying upper bounds of energy bins

daTimes – numpy array of time values, in Modified Julian Date form. May be identical times or times in chronological order, associated with position coordinates.

strCoordSys – Cartesian coordinate system identifier: 'GEI','GEO','GSM','GSE','SM' or 'MAG';
Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'

daCoords1, daCoords2, daCoords3 – numpy arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system and units specified by the strCoordSys and strCoordUnits parameters. Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

daFluxDir1, daFluxDir2, daFluxDir3 – numpy arrays of direction array values associated with times and position arrays. These direction values must be in the same Cartesian coordinate system as the position vectors. These may be a unit vector, or in the same units as the position vectors.

Return value:

iErr – 0 = success, otherwise error

get_pitchAngles

Usage: Returns the set of uni-directional pitch angles used in the model calculations, corresponding to the dimensions of the direction arrays input to the *set_fluxEnvironVarPitch*() method.

Return value:
iErr – 0 = success, otherwise error
da2PitchAngles – 2-dimensional numpy array of pitch angles, in degrees (0-180). [time,direction]

get_numTimes

Usage: Returns the number of defined times specified in the *set_fluxEnviron[]*() methods.
Return value:
iNum – number of times in current ephemeris definition

get_numEnergies

Usage: Returns the number of defined energies specified in the *set_fluxEnviron[]*() methods.
Return value:
iNumE – number of energies in current ephemeris definition

get_numDirections

Usage: Returns the number of defined directions specified in the *set_fluxEnviron[]*() methods.
Return value:
iNumD – number of directions in current ephemeris definition

computeFlyinMean or ***computeFluxMean***

Usage: Returns the 'Mean' model flux at the times, positions, energies (and possibly directions) specified in the most recent call to one of the *set_fluxEnviron[]*() methods.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).
This same method may also be called as *computeFluxMean*(), with same exact arguments.
Return values:
iErr – 0 = success, otherwise error
da3FluxData – 3-dimensional numpy array of the 'mean' flux values. [time,energy,direction]

computeFlyinPercentile or ***computeFluxPercentile***

(iPercentile)

Usage: Returns the model flux results for the specified model Percentile number, at the times, positions, energies (and possibly directions) specified in the most recent call to one of the *set_fluxEnviron[]*() methods.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).
This same method may also be called as *computeFluxPercentile*(), with same exact arguments.
Parameters:
iPercentile – percentile number of the flux values to be returned.
Return values:
iErr – 0 = success, otherwise error
da3FluxData – 3-dimensional numpy array of the flux values for the specified percentile. [time,energy,direction]

computeFlyinPerturbedMean or ***computeFluxPerturbedMean***

(iScenario)

Usage: Returns the model flux results for the specified Perturbed Mean scenario number, at the times, positions, energies (and possibly directions) specified in the most recent call to one of the *set_fluxEnviron[]*() methods.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).
This same method may also be called as *computeFluxPerturbedMean*(), with same exact arguments.

Parameters:

iScenario – scenario number of the Perturbed Mean flux values to be returned.

Return values:

iErr – 0 = success, otherwise error

da3FluxData – 3-dimensional numpy array of the flux values for the specified scenario number. [time,energy,direction]

computeFlyinScenario or ***computeFluxScenario***

(dEpochTime,
iScenario,
iFluxPert = *1*)

Usage: Returns the model flux results for the specified Monte Carlo scenario number, with the specified time progression reference time, at the times, positions, energies (and possibly directions) specified in the most recent call to one of the *set_fluxEnviron[]*() methods.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).
This same method may also be called as *computeFluxScenario*(), with same exact arguments.

Parameters:

dEpochTime – Monte Carlo reference time, in Modified Julian Date form

iScenario – scenario number of the Monte Carlo flux values to be returned.

iFluxPert – optional flag, needed only for developmental testing, for controlling application of flux perturbations within calculations. Defaults to 1 *(true)* when omitted (recommended).

Return values:

iErr – 0 = success, otherwise error

da3FluxData – 3-dimensional numpy array of the flux values for the specified scenario number. [time,energy,direction]

get_defaultPitchAngles

Usage: Returns the set of default pitch angles used in onmidirectional model calculations, using the *set_fluxEnvironOmni*() method.

Return value:

daPitchAngles – numpy array of the 32 default pitch angles, in degrees (0-180). [direction]

AccumModel Class

Import file: accumModel.py "`from accumModel import AccumModel`"

This class is the entry point that provides direct programmatic access to the accumulation model, which performs calculations on the flux data values over time. For the proper processing of the flux data, the *loadBuffer*() method is used collect data for each 'chunk'; the desired *compute**() methods *must be called each time* for that data to be properly processed for its contribution to the desired accumulation; these 'compute' calls may not necessarily return results at each call. Different types of accumulations of may be active simultaneously.

General:

AccumModel

Usage: Instantiates an instance of the AccumModel class
Return value:
'AccumModel' class object

Model Parameter Inputs:

set_interval

(dIntervalDays)

Usage: Specifies the time duration of the accumulation of time-tagged data for use in the calculation of integrated data results. This defined interval (via this method or *set_intervalSec*()) is used only with the *computeIntvFluence*(), *computeBoxcarFluence*() and *computeExponentialFlux*() methods. If unspecified, the interval duration defaults to 1 day (86400 seconds).

Parameters:
dIntervalDays – time duration, in units of days+fraction *[must be greater than smallest timestep of ephemeris]*

Return value:
iErr – 0 = success, otherwise error

set_intervalSec

(dIntervalSecs)

Usage: Specifies the time duration of the accumulation of time-tagged data for use in the calculation of the integrated data results. This defined interval (via this method or *set_interval*()) is used only with the *computeIntvFluence*(), *computeBoxcarFluence*() and *computeExponentialFlux*() methods. If unspecified, the interval duration defaults to 86400 seconds (1 day).

Parameters:
dIntervalSecs – time duration, in seconds *[must be greater than smallest timestep of ephemeris]*

Return value:
iErr – 0 = success, otherwise error

set_increment

(dIncrement)

Usage: Specifies the time delta for the shift of the 'Boxcar' accumulation mode time windows. This defined increment is used only with the *computeBoxcarFluence*() method.

Parameters:

dIncrement – time delta, in seconds, for the increment of time between the start of adjacent Boxcar time windows. May be zero, for self-advancing at the input ephemeris timesteps, or be greater than zero, but less than the value specified in the *set_interval*() method.

Return value:

iErr – 0 = success, otherwise error

get_interval

Usage: Returns the time duration of the accumulation of time-tagged data for use in the calculation of integrated data results, as specified in either the *set_interval*() or *set_intervalSec*() method.

Return value:

dIntervalSecs – time duration, in seconds.

get_increment

Usage: Specifies the time delta for the shift of the 'Boxcar' accumulation mode time windows, as specified in the *set_increment*() method.

Return value:

dIncrement – time delta, in seconds.

Model Execution and Results:

loadBuffer

(daTimes,
da3Data)

Usage: Loads the sets of input time-tagged data into the accumulation buffer, replacing any previous buffer contents.

Parameters:

daTimes – numpy array of time values, in Modified Julian Date form

da3Data – 3-dimensional numpy array of data values to be loaded into buffer. [time,energy,direction]

Return value:

iErr – 0 = success, otherwise error

addToBuffer

(dTime,
da2Data)

Usage: Adds a single set of input time-tagged data to the current contents of the accumulation buffer.

Parameters:

dTime – time value, in Modified Julian Date form

da2Data – 2-dimensional numpy array of data values to be loaded into buffer. [energy,direction]

Return value:

iErr – 0 = success, otherwise error

computeFluence

Usage: Processes the current contents of the data buffer, then returns the cumulative time-integrated data results at the time tags of the current buffer contents. *Any specified accumulation time interval has no effect on these calculations.*

Return values:

iNum – 0 or greater = number of data records returned in arrays; otherwise error

daFluenceTimes – numpy array of times, in Modified Julian Date form

da3Fluence – 3-dimensional numpy array of the calculated fluence values [time,energy,direction]

computeIntvFluence

(iReturnPartial = *0*)

Usage: Processes the current contents of the data buffer, then returns the time-integrated data results from any *completed* accumulation intervals (whose duration is previously specified in calls to the *set_interval[Sec]*() methods). Linear interpolation of the buffered data values is performed when their associated times do not line up with the start or stop times of the accumulation intervals. The returned times correspond to the end times of these completed accumulation intervals. The last argument should be omitted or set to 0 *(false)* until all data has been processed; the subsequent call with the last argument set to 1 *(true)* will return any remaining fluence data (for a partial interval period).

Parameters:

iReturnPartial – optional flag for returning the calculated fluence values for an incomplete accumulation interval, if any. Set to 1 *(true)* only after *all* data has been processed.

Return values:

iNum – 0 or greater = number of data records returned in arrays; otherwise error

daFluenceTimes – numpy array of times, in Modified Julian Date form, corresponding to the end of the intervals.

da3Fluence – 3-dimensional numpy array of the calculated fluence values for zero or more completed accumulation intervals. [time,energy,direction]

iaIndices – numpy array of the (nearest) indices to the current data buffer entries at which the completed accumulation intervals end. -1 means at or off end of buffer. This information is useful for the annotation of supplementary information associated with the buffered data, such as a time-varying pitch angles.

accumIntvFluence

(daFluenceTimes,
da3Fluence,
iAccumReset = *0*)

Usage: Sums the input fluence values over time, returning the cumulative fluence values since the initial call or the last reset. This method is intended to be used in tandem with 'Interval' accumulation fluence results from the *computeIntvFluence*() method.

Parameters:

daFluenceTimes – numpy array of times, in Modified Julian Date form

da3Fluence – 3-dimensional numpy array of the previously calculated fluence values. [time,energy,direction]

iAccumReset – 1 *(true)* or 0 *(false)*; optional flag for forcing a reset of the internal fluence accumulation data.

Return values:

iNum – 0 or greater = number of data records returned in arrays; otherwise error

da3FluenceIntvAccum – 3-dimensional numpy array of the corresponding *cumulative* fluence values since the initial call to this routine, or a reset was indicated.

computeFullFluence

(iReturnFinal = *0*)

Usage: Processes the current contents of the data buffer. Because this is performing an accumulation over the 'full' time duration, no results are normally returned. After all data has been processed, the single set of time-integrated data results, for the entire time duration, is returned in the subsequent call to this method where the iReturnFinal argument is set to 1 *(true)*. *Any specified accumulation time interval has no effect on these calculations.*

Parameters:

iReturnFinal – 1 *(true)* or 0 *(false)*; optional flag for returning the single set of cumulative fluence values. Set to 1 only after *all* data has been processed.

Return values:

iNum – 0 or greater = number of data records returned in arrays; otherwise error

daFluenceTimes – numpy array of time, in Modified Julian Date form

da3Fluence – 3-dimensional numpy array of the calculated fluence values. Due to the nature of this accumulation, this array will be empty except when iReturnFinal is 1 *(true)*. [time,energy,direction]

computeBoxcarFluence

(iReturnPartial = *0*)

Usage: Processes the current contents of the data buffer, then returns the time-integrated data results from any *completed* boxcar accumulation intervals. The duration of the boxcar windows is previously specified in calls to the *set_interval[Sec]*() methods; the time spacing between the start times of subsequent boxcar windows is specified with the *set_increment*() method . Linear interpolation of the buffered data values is performed when their associated times do not line up with the start or end times of the boxcar accumulation intervals. The returned times correspond to the end times of these completed boxcar accumulation intervals.

The returned boxcar fluence values may subsequently be used as input to the *computeAverageFlux*() method to calculate the associated boxcar interval average flux. The *applyWorstToDate*() method can be used to further process these results, when the appropriate 'maximum' array of values is maintained.

Parameters:

iReturnPartial – 1 *(true)* or 0 *(false)*; optional flag for returning the calculated fluence values for the oldest incomplete boxcar accumulation interval, if any. Set to 1 *(true)* only after *all* data has been processed. Also causes the boxcar accumulation information to be reset.

Return values:

iNum – 0 or greater = number of data records returned in arrays; otherwise error

daFluenceTimes – numpy array of times, in Modified Julian Date form

da3Fluence – 3-dimensional numpy array of the calculated fluence values for zero or more completed boxcar accumulation intervals. [time,energy,direction]

iaIndices – numpy array of the (nearest) indices to the current data buffer entries at which the completed accumulation intervals end. -1 means at or off end of buffer. This information is useful for the annotation of supplementary information associated with the buffered data, such as a time-varying pitch angles.

computeAverageFlux

(daFluenceTimes,
da3Fluence,
dIntervalSec)

Usage: Computes the average flux rate over fixed-length intervals, based on the input fluence and interval time.

Parameters:

daFluenceTimes – numpy array of times, in Modified Julian Date form

da3Fluence – 3-dimensional numpy array of the previously calculated fluence values, associated with the interval end times specified in daFluenceTimes. [time,energy,direction]

dIntervalSec – duration of time interval, in seconds, used in the calculation of the fluence values. Specify the value used in the *set_timeInterval[Sec]*() method for 'interval'- and 'boxcar'-type accumulations; specify '0' for use with cumulative fluences (no accumulation); specify '-1' for 'full' accumulation fluence.

Return values:

iNum – 0 or greater: number of sets of flux averages returned; otherwise error

da3FluxAvg – 3-dimensional numpy array of the calculated flux average values [time,energy,direction]

computeExponentialFlux

(iFinal = *0*)

Usage: Processes the current contents of the data buffer, then returns the exponential average flux data results at the input data times; the calculations are dependent on the interval value specified in a call to the *set_interval[Sec]*() method.

The *applyWorstToDate*() method can be used to further process these results, when the appropriate 'maximum' array of values is maintained.

Parameters:

iFinal – optional flag for process completion; set to 1 (*true*) only after all data has been processed.

Return values:

iNum – 0 or greater = number of data records returned in arrays; otherwise error

daExpFluxTimes – numpy array of times, in Modified Julian Date form

da3ExpFlux – 3-dimensional numpy array of the calculated exponential average flux values for zero or more input data entries. [time,energy,direction]

iaIndices – numpy array of the (nearest) indices to the current data buffer entries at which the completed accumulation intervals end. -1 means at or off end of buffer. This information is useful for the annotation of supplementary information associated with the buffered data, such as a time-varying pitch angles.

applyWorstToDate

(da3Data,
da2MaxData,
iReset = *0*)

Usage: Scans the input data, determining the maximum values over time in each of the other two dimensions. These maximum values are returned, as well as the 'worst to date' for the input data.

Parameters:

da3Data – 3-dimensional numpy array of data, of no specific type. [time,energy|depth,direction]

da2MaxData – input 2-dimensional numpy array of the previously determined maximum data values. [energy|depth,direction]. A "reset" of these maximum values is implied when an empty da3MaxData array is passed as *input*.

iReset – 1 *(true)* or 0 *(false)*; optional flag to reset current maximum data vales

Return value:

iErr – 0 = success, otherwise error

da3DataWorst – 3-dimensional numpy array of the corresponding 'worst to date' of the input data values. [time,energy|depth,direction]

da2MaxData – returned 2-dimensional numpy array of the currently determined maximum data values. [energy|depth,direction].

reset_fluence

Usage: clears the internally stored timestep-based cumulative fluence accumulation data. Subsequent calls to c*omputeFluence*() method will start a new set of fluence accumulation.

Return value:

iErr – 0 = success, otherwise error

reset_intvFluence

Usage: clears the internally stored interval-based cumulative fluence accumulation data. Subsequent calls to c*omputeIntvFluence*() method will start a new set of fluence accumulation.

Return value:

iErr – 0 = success, otherwise error

reset_fullFluence

Usage: clears the internally stored full fluence accumulation data. Subsequent calls to c*omputeFullFluence*() method will start a new set of fluence accumulation.

Return value:

iErr – 0 = success, otherwise error

reset_boxcarFluence

Usage: clears the internally stored boxcar fluence accumulation data. Subsequent calls to c*omputeBoxcarFluence*() method will start a new set of fluence accumulation.

Return value:

iErr – 0 = success, otherwise error

reset_exponentialFlux

Usage: clears the internally stored exponential flux average data. Subsequent calls to c*omputeExponentialFlux*() method will start a new set of exponential flux average calculations.

Return value:

iErr – 0 = success, otherwise error

get_fluenceStartTime

Usage: Returns the starting time for the cumulative fluence accumulation data.

Return value:

dStartTime – fluence accumulation data start time, in MJD form.

get_intvFluenceStartTime

Usage: Returns the starting time for the current interval of fluence accumulation data.

Return value:

dStartTime – fluence accumulation data start time, in MJD form.

get_fullFluenceStartTime

Usage: Returns the starting time for the full fluence accumulation data.
Return value:
dStartTime – fluence accumulation data start time, in MJD form.

get_boxcarFluenceStartTime

Usage: Returns the starting time for the oldest active boxcar interval of fluence accumulation data.
Return value:
dStartTime – boxcar fluence accumulation data start time, in MJD form.

get_lastLength

Usage: Returns the last interval time duration of the most recent *compute*Fluence*|*Flux*() method call in which the 'iReturnPartial' input parameter was set to 1 *(true)*.
Return value:
dLength – last interval time duration, in seconds. -1 if no partial interval occurred.

DoseModel Class

Import file: doseModel.py "`from doseModel import DoseModel`"

This class is the entry point that provides direct programmatic access to the ShieldDose2 model for the calculation of radiation dose rates received by a target material behind or inside aluminum shielding. Input flux values *must* be 1pt Differential, Omni-directional fluxes, otherwise dose results are invalid.

General:

DoseModel

Usage: Instantiates an instance of the DoseModel class
Return value:
'DoseModel' class object

Model Parameter Inputs:

set_modelDBDir

(strDataDir)

Usage: Specifies the directory that contains the collection IRENE model database files. The various database files required are automatically selected according to the model and parameters specified. The use of this method is highly recommended, as it *eliminates* the need for the other methods that specify the individual database files; those are only needed for using alternate or non-standard versions.
Parameters:
strDataDir – directory path for the IRENE database files.
Return value:
iErr – 0 = success, otherwise error

set_modelDBFile

(strModelDBFile)

Usage: Specifies the name of the file (including path) for the dose calculation model database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).
This database name is in the form of '<path>/sd2DB.h5'.
Parameters:
strModelDBFile – model database filename, including path
Return value:
iErr – 0 = success, otherwise error

set_species

(strSpecies)

Usage: Specifies the particle species for the ShieldDose2 model calculations.
Parameters:
strSpecies – species identification: 'H+'|'protons' or 'e-'|'electrons'.
Return value:
iErr – 0 = success, otherwise error

set_energies

(daEnergies,
string strEnergyUnits = "MeV")

Usage: Specifies the set of energies (and units) of the flux values that are to be input to the *computeFluxDose[Rate]*() methods.

Parameters:

daEnergies – numpy array of energy values

strEnergyUnits – energy units: 'eV', 'keV', 'MeV' or 'GeV'; if omitted, this defaults to 'MeV'

Return value:

iErr – 0 = success, otherwise error

set_depths

(daDepths,
string strDepthUnits = "mm")

Usage: Specifies the list of aluminum shielding thickness depths, and their associated units. A minimum of three depth values are required for performing the dose calculations. Input depth values must be in increasing order, with no duplicates.

Nominal range: 0.100 – 111.1 mm; 3.937 – 4374 mils; 0.027 – 30.0 g/cm^2 .

Parameters:

daDepths – numpy array of depth values; in ascending order, no duplicates

strDepthUnits – unit specification: 'millimeters'|'mm', 'mils' or 'gpercm2'; defaults to 'mm' if omitted

Return value:

iErr – 0 = success, otherwise error

set_detector

(strDetector)

Usage: Specifies the dose detector material type, behind (or inside) the aluminum shielding.

Parameters:

strDetector – material name: 'Aluminum'|'Al', 'Graphite', 'Silicon'|'Si', 'Air', 'Bone', 'Tissue', 'Calcium'|'Ca' | 'CaF2', 'Gallium'|'Ga' | 'GaAs', 'Lithium'|'Li' | 'LiF', 'Glass'|'SiO2', 'Water'|'H2O'

Return value:

iErr – 0 = success, otherwise error

set_geometry

(strGeometry)

Usage: Specifies the geometry of the aluminum shielding in front of (or around) the detector target.

Parameters:

strGeometry – configuration name: 'Spherical4pi', 'Spherical2pi', 'FiniteSlab' or 'SemiInfiniteSlab'

Return value:

iErr – 0 = success, otherwise error

set_nuclearAttenMode

(strNucAttenMode)

Usage: Specifies the 'Nuclear Attenuation' mode used during the ShieldDose2 model calculations.

Parameters:

strNucAttenMode – attenuation mode: 'None', 'NuclearInteractions' or 'NuclearAndNeutrons'

Return value:
iErr – 0 = success, otherwise error

set_withBrems

(bool bVerdict = *true*)

Usage: Specifies whether the electron dose calculations are to include the bremsstrahlung contributions or not. The default model state is to *include* the bremsstrahlung contributions.

Parameters:
bVerdict – optional flag for including(default) or excluding the bremsstrahlung contributions.

Return value: -none-

---

get_modelDBDir

Usage: Returns the directory name containing the collection of IRENE model database files that was specified in a previous call to the *set_modelDBDir*() method; otherwise, blank.

Return value:
strModelDBDir – model database directory.

get_modelDBFile

Usage: Returns the name of the file (including path) for the dose calculation model database. This will be available immediately, when specified using the *set_modelDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to one of the *compute**() methods.

Return value:
strModelDBFile – model database filename, including path.

get_species

Usage: Returns the particle species for the ShieldDose2 model calculations, as specified in the *set_species*() method.

Return value:
strSpecies – species identification name.

get_numEnergies

Usage: Returns the number of currently defined energies, as specified in the *set_energies*() method.

Return value:
iNumE – 0 or greater = number of defined energies, otherwise error.

get_numDepths

Usage: Returns the number of currently defined depths, as specified in the *set_depths*() method.

Return value:
iNumD – 0 or greater = number of defined depths, otherwise error.

get_detector

Usage: Returns the dose detector material type, behind (or inside) the aluminum shielding, as specified in the *set_detector*() method.

Return value:

strDetector – material name.

get_geometry

Usage: Returns the geometry of the aluminum shielding in front of (or around) the detector target, as specified in the *set_geometry*() method.

Return value:

strGeometry – configuration name.

get_nuclearAttenMode

Usage: Returns the 'Nuclear Attenuation' mode used during the ShieldDose2 model calculations, as specified in the *set_nuclearAttenMode*() method.

Return value:

strNucAttenMode – attenuation mode.

get_withBrems

Usage: Returns the current state for the inclusion of the bremsstrahlung contribution in the electron dose values calculated, as specified in the *set_withBrems*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*.

Model Execution and Results:

computeFluxDose

(daFluxData)

Usage: Returns the dose results, based on the input particle flux values and the previously specified particle species, flux energies, shielding and detector parameters.

Parameters:

daFluxData – numpy array of omni-directional differential flux values, over the energy levels previously specified via the *set_energies*() method, for a single time; or may be fluence values.
These input values are expected to be in units of #/cm^2/sec/MeV (flux) or #/cm^2/MeV (fluence)

Return values:

iErr – 0 = success, otherwise error

daDoseData – numpy array of dose model results, over the shielding depths previously specified via the *set_depths*() method. When a flux value is input, these returned values are a 'dose rate' [rads/sec]; an input of fluence values returns the associated accumulated dose value [rads].

computeFluxDoseRate

(da3FluxData)

Usage: Returns the modeled dose rate values, based on the input particle flux values and the previously specified particle species, flux energies, shielding and detector parameters, for one or more times.

Parameters:

da3FluxData – 3-dimensional numpy array of omni-directional differential flux values, over the energy levels previously specified via the *set_energies*() method, for a single direction (*omni-directional only*), for one or more times.[time,energy,direction]
These input flux values are expected to be in units of #/cm^2/sec/MeV

Return value:

iErr – 0 = success, otherwise error

da3DoseData – 3-dimensional numpy array of dose rate values [rads/sec], for a single direction, over the shielding depths previously specified via the *set_depths*() method. [time,depths,direction]

computeFluenceDose

(da3FluenceData)

Usage: Returns the modeled accumulated dose values, based on the input particle flux values and the previously specified particle species, flux energies, shielding and detector parameters, for one or more times.

Parameters:

da3FluenceData – 3-dimensional numpy array of omni-directional differential fluence values, over the energy levels previously specified via the *set_energies*() method, for a single direction (*omni-directional only*), for one or more times.[time,energy,direction]

These input fluence values are expected to be in units of #/cm^2/MeV

Return value:

iErr – 0 = success, otherwise error

da3DoseVal – 3-dimensional numpy array of accumulated dose values [rads], for a single direction, over the shielding depths previously specified via the *set_depths*() method. [time,depths,direction]

DoseKernel Class

Import file: kernelModel.py "`from kernelModel import DoseKernel`"

This class is the entry point that provides direct programmatic access the kernel-based method for the calculation of radiation dose rates received by a target material behind or inside aluminum shielding. Input flux values *must* be 1pt Differential, Omni-directional fluxes, otherwise dose results are invalid. The Dose Kernel xml files were produced based on results from the ShieldDose2 model. For more information, see Appendix K of the User's Guide document.

General:

DoseKernel

Usage: Instantiates an instance of the DoseKernel class
Return value:
'DoseKernel' class object

Model Parameter Inputs:

set_kernelXmlPath

(strKernelXmlPath)
Usage: Specifies the path for the collection of dose-specific kernel xml files.
(The proper file is automatically selected based on the specified geometry and detector material.)
Parameters:
strKernelXmlPath – dose kernel xml file collection path
Return value:
iErr – 0 = success, otherwise error

set_kernelXmlFile

(strKernelXmlFile)
Usage: Specifies the name of the file (including path) for the kernel-based dose calculation method.
The dose xml file specified *must* correspond to the specified geometry and detector material.
Parameters:
strKernelXmlFile – dose kernel xml filename, including path
Return value:
iErr – 0 = success, otherwise error

set_species

(strSpecies)
Usage: Specifies the particle species for the kernel-based dose calculations.
Parameters:
strSpecies – species identification: 'H+'|'protons' or 'e-'|'electrons'.
Return value:
iErr – 0 = success, otherwise error

set_energies

(daEnergies,
string strEnergyUnits = "MeV")

Usage: Specifies the set of energies (and units) of the flux values that are to be input to the *computeFluxDose[Rate]*() methods.

Parameters:

daEnergies – numpy array of energy values

strEnergyUnits – energy units: 'eV', 'keV', 'MeV' or 'GeV'; if omitted, this defaults to 'MeV'

Return value:

iErr – 0 = success, otherwise error

set_depths

(daDepths,
string strDepthUnits = "mm")

Usage: Specifies the list of aluminum shielding thickness depths, and their associated units. A minimum of three depth values are required for performing the dose calculations. Input depth values must be in increasing order, with no duplicates.

Nominal range: 0.100 – 111.1 mm; 3.937 – 4374 mils; 0.027 – 30.0 g/cm^2 .

Parameters:

daDepths – numpy array of depth values; in ascending order, no duplicates

strDepthUnits – unit specification: 'millimeters'|'mm', 'mils' or 'gpercm2'; defaults to 'mm' if omitted

Return value:

iErr – 0 = success, otherwise error

set_detector

(strDetector)

Usage: Specifies the dose detector material type, behind (or inside) the aluminum shielding.

Parameters:

strDetector – material name: 'Aluminum'|'Al', 'Graphite', 'Silicon'|'Si', 'Air', 'Bone', 'Tissue', 'Calcium'|'Ca'|'CaF2', 'Gallium'|'Ga'|'GaAs', 'Lithium'|'Li'|'LiF', 'Glass'|'SiO2', 'Water'|'H2O'

Return value:

iErr – 0 = success, otherwise error

set_geometry

(strGeometry)

Usage: Specifies the geometry of the aluminum shielding in front of (or around) the detector target.

Parameters:

strGeometry – configuration name: 'Spherical4pi', 'Spherical2pi', 'FiniteSlab' or 'SemiInfiniteSlab'

Return value:

iErr – 0 = success, otherwise error

set_nuclearAttenMode

(strNucAttenMode)

Usage: Specifies the 'Nuclear Attenuation' mode used for the kernel-based dose calculations.

Parameters:

strNucAttenMode – attenuation mode: 'None'
(other modes of 'NuclearInteractions' and 'NuclearAndNeutrons' are currently unsupported)

Return value:

iErr – 0 = success, otherwise error

set_withBrems

(bool bVerdict = *true*)

Usage: Specifies whether the electron dose calculations are to include the bremsstrahlung contributions or not. The default model state is to *include* the bremsstrahlung contributions.

Parameters:

bVerdict – optional flag for including(default) or excluding the bremsstrahlung contributions.

Return value: -none-

get_species

Usage: Returns the particle species for the kernel-based dose calculations, as specified in the *set_species*() method.

Return value:

strSpecies – species identification name.

get_numEnergies

Usage: Returns the number of currently defined energies, as specified in the *set_energies*() method.

Return value:

iNumE – 0 or greater = number of defined energies, otherwise error

get_numDepths

Usage: Returns the number of currently defined depths, as specified in the *set_depths*() method.

Return value:

iNumD – 0 or greater = number of defined depths, otherwise error

get_detector

Usage: Returns the dose detector material type, behind (or inside) the aluminum shielding, as specified in the *set_detector*() method.

Return value:

strDetector – material name.

get_geometry

Usage: Returns the geometry of the aluminum shielding in front of (or around) the detector target, as specified in the *set_geometry*() method.

Return value:

strGeometry – configuration name.

get_nuclearAttenMode

Usage: Returns the 'Nuclear Attenuation' mode used during the kernel-based dose calculations, as specified in the *set_nuclearAttenMode*() method.

Return value:

strNucAttenMode – attenuation mode name.

get_withBrems

Usage: Returns the current state for the inclusion of the bremsstrahlung contribution in the electron dose values calculated, as specified in the *set_withBrems*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*.

Model Execution and Results:

computeFluxDose

(daFluxData)

Usage: Returns the dose results, based on the input particle flux values and the previously specified particle species, flux energies, shielding and detector parameters.

Parameters:

daFluxData – numpy array of omni-directional differential flux values, over the energy levels previously specified via the *set_energies*() method, for a single time; or may be fluence values. These input values are expected to be in units of #/cm^2/sec/MeV (flux) or #/cm^2/MeV (fluence)

Return values:

iErr – 0 = success, otherwise error

daDoseData – numpy array of dose model results, over the shielding depths previously specified via the *set_depths*() method. When a flux value is input, these returned values are a 'dose rate' [rads/sec]; an input of fluence values returns the associated accumulated dose value [rads].

computeFluxDoseRate

(da3FluxData)

Usage: Returns the modeled dose rate values, based on the input particle flux values and the previously specified particle species, flux energies, shielding and detector parameters, for one or more times.

Parameters:

da3FluxData – 3-dimensional numpy array of omni-directional differential flux values, over the energy levels previously specified via the *set_energies*() method, for a single direction (*omni-directional only*), for one or more times.[time,energy,direction]
These input flux values are expected to be in units of #/cm^2/sec/MeV

Return value:

iErr – 0 = success, otherwise error

da3DoseData – 3-dimensional numpy array of dose rate values [rads/sec], for a single direction, over the shielding depths previously specified via the *set_depths*() method. [time,depths,direction]

computeFluenceDose

(da3FluenceData)

Usage: Returns the modeled accumulated dose values, based on the input particle flux values and the previously specified particle species, flux energies, shielding and detector parameters, for one or more times.

Parameters:

da3FluenceData – 3-dimensional numpy array of omni-directional differential fluence values, over the energy levels previously specified via the *set_energies*() method, for a single direction (*omni-directional only*), for one or more times.[time,energy,direction]
These input fluence values are expected to be in units of #/cm^2/MeV

Return value:

iErr – 0 = success, otherwise error

da3DoseVal – 3-dimensional numpy array of accumulated dose values [rads], for a single direction, over the shielding depths previously specified via the *set_depths*() method. [time,depths,direction]

AggregModel Class

Import file: aggregModel.py "`from aggregModel import AggregModel`"

This class is the entry point that provides direct programmatic access to the aggregation model. This is used for the collection (or 'aggregation') of data values from multiple scenarios of the 'Perturbed Mean' or 'Monte Carlo' calculations. Once all of the scenario data sets have been loaded into the data aggregation, via the *addScenToAgg*() method, the confidence levels may be calculated using the various *compute**() methods. It is recommended that the aggregation contain at least ten sets of scenario data in order to produce statistically meaningful results.
Important Note: the confidence levels of 0 and 100 percent are excluded from the normal percentile calculations, as they are the 'endpoints', and return simply the minimum or maximum scenario value. When the number of scenarios is less than 100, additional neighboring percent values are also excluded. See the 'Accumulation and Aggregation Inputs' section of the User's Guide document for more details.

General:

AggregModel

Usage: Instantiates an instance of the AggregModel class
Return value:
'AggregModel' class object

Model Parameter Inputs:

reset

Usage: Clears all time-tagged data from the scenario data aggregation.
Return value:
iErr – 0 = success, otherwise error

addScenToAgg

(daScenTimes,
da3ScenData)

Usage: Loads the sets of input time-tagged 'Perturbed Mean' or 'Monte Carlo' scenario data into a new or existing aggregation. The input data sizes and dates of subsequent calls must match those of the first call following a call to the *reset* () method.
Parameters:
daScenTimes – numpy array of time values, in Modified Julian Date form
da3ScenData – 3-dimensional numpy array of scenario data values to be loaded into the aggregation. [time,energy|depth,direction]
Return value:
iErr – 0 = success, otherwise error

get_aggDataDim

Usage: Returns the dimensions of the aggregation data defined using the *addScenToAgg*() method.
Return value:
iNum1, iNum2, iNum3 – length of time, energy or depth, and direction dimensions of aggregation data.

get_numScenarios

Usage: Returns the current number of aggregation scenarios defined using the *addScenToAgg*() method.

Return value:

iNumScen – number of current aggregation scenarios.

Model Execution and Results:

computeConfLevel

(iPercent)

Usage: Calculates the specified confidence level values from the current contents of the scenario data aggregation, at each time direction and energy level or shield depth.

Parameters:

iPercent – input confidence level percent value (0-100)

Return values:

iErr – 0 = success, otherwise error

daPercTimes – numpy array of times, in Modified Julian Date form

da3PercData – 3-dimensional numpy array of the calculated confidence level values [time,energy|depth,direction]

computeMedian

Usage: Calculates the median (50% confidence level) data values from the current contents of the scenario data aggregation, at each time direction and energy level or shield depth.

Return value:

iErr – 0 = success, otherwise error

daPercTimes – numpy array of times, in Modified Julian Date form

da3PercData – 3-dimensional numpy array of the calculated aggregation median values [time,energy|depth,direction]

computeMean

Usage: Calculates the average ('mean') data values from the current contents of the scenario data aggregation, at each time direction and energy level or shield depth.

This is NOT a confidence level. The results of this calculation are of indeterminate meaning.
Use of this method is strongly discouraged.

Return values:

iErr – 0 = success, otherwise error

daPercTimes – numpy array of times, in Modified Julian Date form

da3PercData – 3-dimensional numpy array of the calculated aggregation mean values [time,energy|depth,direction]

AdiabatModel Class

Import file: adiabatModel.py "`from adiabatModel import AdiabatModel`"

This class is the entry point that provides direct programmatic access to the Adiabat model, for the calculation of the adiabatic invariant values associated with an input set of times, spatial coordinates and pitch angles. Please note that all time values, both input and output, are in Modified Julian Date (MJD) form. Conversions to and from MJD times are available from the DateTimeUtil class, described elsewhere in this Model-Level API section. Position coordinates are always used in sets of three values, in the coordinate system and units that are specified. Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the order of the coordinate values for non-Cartesian coordinate systems.
More information about the geomagnetic and adiabatic invariant values may be found in Appendix D and E of the User's Guide document.

General:

AdiabatModel

Usage: Instantiates an instance of the AdiabatModel class
Return value:
'AdiabatModel' class object

Model Parameter Inputs:

set_modelDBDir

(strDataDir)

Usage: Specifies the directory that contains the collection IRENE model database files. The various database files required are automatically selected according to the model and parameters specified. The use of this method is highly recommended, as it *eliminates* the need for the other methods that specify the individual database files; those are only needed for using alternate or non-standard versions.
Parameters:
strDataDir – directory path for the IRENE database files.
Return value:
iErr – 0 = success, otherwise error

set_kPhiDBFile

(strKPhiDBFile)

Usage: Specifies the name of the file (including path) for the K/Phi database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).
This database name is in the form of '<path>/fastPhi_net.mat'.
Parameters:
strKPhiDBFile – database filename, including path
Return value:
iErr – 0 = success, otherwise error

set_kHMinDBFile

(strKHMinDBFile)

Usage: Specifies the name of the file (including path) for the K/Hmin database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/fast_hmin_net.mat'.

Parameters:

strKHMinDBFile – database filename, including path

Return value:

iErr – 0 = success, otherwise error

set_magfieldDBFile

(strMagfieldDBFile)

Usage: Specifies the name of the file (including path) for the magnetic field model database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/igrfDB.h5'.

Parameters:

strMagfieldDBFile – magnetic field model database filename, including path

Return value:

iErr – 0 = success, otherwise error

---

get_modelDBDir

Usage: Returns the directory name containing the collection of IRENE model database files that was specified in a previous call to the *set_modelDBDir*() method; otherwise, blank.

Return value:

strModelDBDir – model database directory.

get_kPhiDBFile

Usage: Returns the name of the file (including path) for the K/Phi database. This will be available immediately, when specified using the *set_kPhiDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to one of the *compute_coordinate**() methods.

Return value:

strKPhiDBFile – database filename, including path.

get_kHMinDBFile

Usage: Returns the name of the file (including path) for the K/Hmin database. This will be available immediately, when specified using the *set_kHMinDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to one of the *compute_coordinate**() methods.

Return value:

strKHMinDBFile – database filename, including path.

get_magfieldDBFile

(strMagfieldDBFile)

Usage: Returns the name of the file (including path) for the magnetic field model database. This will be available immediately, when specified using the *set_magfieldDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to one of the *computeCoordinate**() or *convertCoords**() methods.

Return value:

strMagfieldDBFile – magnetic field model database filename, including path.

Adiabatic invariant limit defaults: K = 0.0 – 25.0; Hmin = -1500.0 – 50000.0 [km]; Phi = 0.125 – 2.5
Any K, Hmin or Phi values calculated to be outside of their respective limits are set to -1.0×10^{-31} fill value.

set_kMin

(dKMin = 0.0)

Usage: Specifies the minimum 'K' value returned from *computeCoordinateSet*() methods.

Parameters:

dKMin – 'K' adiabatic value minimum (0.0 if not specified)

Return value:

iErr – 0 = success, otherwise error

set_kMax

(dKMax = 25.0)

Usage: Specifies the maximum 'K' value returned from *computeCoordinateSet*() methods.

Parameters:

dKMax – 'K' adiabatic value maximum (25.0 if not specified)

Return value:

iErr – 0 = success, otherwise error

set_hminMin

(dHminMin = -1500.0)

Usage: Specifies the minimum 'Hmin' value, altitude in km, returned from *computeCoordinateSet*() methods.

Parameters:

dHminMin – 'Hmin' adiabatic value minimum (-1500.0 if not specified)

Return value:

iErr – 0 = success, otherwise error

set_hminMax

(dHminMax = 50000.0)

Usage: Specifies the maximum 'Hmin' value, altitude in km, returned from *computeCoordinateSet*() methods.

Parameters:

dHminMin – 'Hmin' adiabatic value maximum (50000.0 if not specified)

Return value:

iErr – 0 = success, otherwise error

set_phiMin

(dPhiMin = 0.125)

Usage: Specifies the minimum 'Phi' value returned from *computeCoordinateSet*() methods.

Parameters:

dPhiMin – 'Phi' adiabatic value minimum (0.125 if not specified)

Return value:

iErr – 0 = success, otherwise error

set_phiMax

(dPhiMax = 2.5)

Usage: Specifies the maximum 'Phi' value returned from *computeCoordinateSet*() methods.

Parameters:

dPhiMin – 'Phi' adiabatic value maximum (2.5 if not specified)

Return value:

iErr – 0 = success, otherwise error

updateLimits

Usage: Implements any changes to the K, Hmin or Phi limit specifications, but requires that the database files have already been specified via *set_kPhiDBFile*(), *set_kHMinDBFile*() and *set_magfieldDBFile*() methods. Use of this method is needed only if any of these limits are changed *after* the initial call to one of the *computeCoordinateSet*() methods.

Return value:

iErr – 0 = success, otherwise error

get_kMin

Usage: Returns the minimum 'K' value returned from *computeCoordinateSet*() methods.

Return value:

dKMin – 'K' adiabatic value minimum, as specified in the *set_kMin*() method.

get_kMax

Usage: Returns the maximum 'K' value returned from *computeCoordinateSet*() methods.

Return value:

dKMax – 'K' adiabatic value maximum, as specified in the *set_kMax*() method.

get_hminMin

Usage: Returns the minimum 'Hmin' value, altitude in km, returned from *computeCoordinateSet*() methods.

Return value:

dHminMin – 'Hmin' adiabatic value minimum, as specified in the *set_hminMin*() method.

get_hminMax

Usage: Specifies the maximum 'Hmin' value, altitude in km, returned from *computeCoordinateSet*() methods.

Return value:

dHminMin – 'Hmin' adiabatic value maximum, as specified in the *set_hminMax*() method.

get_phiMin

Usage: Specifies the minimum 'Phi' value returned from *computeCoordinateSet*() methods.
Return value:
dPhiMin – 'Phi' adiabatic value minimum, as specified in the *set_phiMin*() method.

get_phiMax

Usage: Specifies the maximum 'Phi' value returned from *computeCoordinateSet*() methods.
Return value:
dPhiMin – 'Phi' adiabatic value maximum, as specified in the *set_phiMax*() method.

Model Execution and Results:

computeCoordinateSet

(strCoordSys,
strCoordUnits,
daTimes,
daCoord1,
daCoord2,
daCoord3,
daPitchAngles)

Usage: Calculates the adiabatic invariant and magnetic field values associated with the input times, position and fixed set of pitch angles. The magnetic field values are independent of pitch angle.
Parameters:
strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL';
Please consult the User's Guide document, "Supported Coordinate Systems" for more details.
strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.
daTimes – numpy array of time values, in Modified Julian Date form
daCoord1, daCoord2, daCoord3 – numpy arrays of position values, in the coordinate system and units specified by the *strCoordSys* and *strCoordUnit* parameter values.
Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' ordering of the input coordinate values for non-Cartesian coordinate systems.
daPitchAngles – numpy array of fixed pitch angles, to be used for all time/position coordinates in the adiabatic invariant value calculations. Valid range: 0.0 - 180.0 degrees.
Return values:
iErr – 0 = success, otherwise error
da2Alpha – 2-dimensional numpy array of equatorial pitch angles ('alpha') associated with the pitch angles at the ephemeris locations. [time,direction]
da2Lm – 2-dimensional numpy array of McIllwain L-shell value associated with the pitch angles at the ephemeris locations. [time,direction]
da2K – 2-dimensional numpy array of adiabatic invariant 'K' value associated with the pitch angles at the ephemeris locations. [time,direction]
da2Phi – 2-dimensional numpy array of adiabatic invariant 'Phi' associated with the pitch angles at the ephemeris locations. [time,direction]
da2Hmin – 2-dimensional numpy array of adiabatic invariant 'Hmin' associated with the pitch angles at the ephemeris locations. [time,direction]

da2Lstar – 2-dimensional numpy array of adiabatic invariant 'L*' associated with the pitch angles at the ephemeris locations. [time,direction]

daBmin – 1-dimensional numpy array of minimum IGRF model magnetic field strength value (nanoTeslas) along field line containing the ephemeris location. [time].

daBlocal – 1-dimensional numpy array of IGRF model magnetic field strength value (nanoTeslas) at the ephemeris location. [time]

daMagLT – 1-dimensional numpy array of magnetic local time (hours) at the Bmin positions. [time]

da2dB – 2-dimensional numpy array of the local magnetic field vector components at the ephemeris locations. [time,components]

da2dI – 2-dimensional numpy array of the local magnetic field current "I" value associated with the pitch angles at the ephemeris locations. [time,direction]

computeCoordinateSetVarPitch

(strCoordSys,
strCoordUnits,
daTimes,
daCoord1,
daCoord2,
daCoord3,
da2PitchAngles)

Usage: Calculates the adiabatic invariant and magnetic field values associated with the input times, position and a *time-varying* set of pitch angles. The magnetic field values are independent of pitch angle.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – coordinate units, 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

daTimes – numpy array of time values, in Modified Julian Date form

daCoord1, daCoord2, daCoord3 – numpy arrays of position values, in the coordinate system and units specified by the *strCoordSys* and *strCoordUnit* parameter values.
Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' ordering of the input coordinate values for non-Cartesian coordinate systems.

da2PitchAngles – 2-dimensional numpy array of pitch angles, to be used for each time/position coordinates in the adiabatic invariant value calculations. Valid range: 0.0 - 180.0 degrees. [time,direction]

Return values:

iErr – 0 = success, otherwise error

da2Alpha – 2-dimensional numpy array of equatorial pitch angles ('alpha') associated with the pitch angles at the ephemeris locations. [time,direction]

da2Lm – 2-dimensional numpy array of McIllwain L-shell value associated with the pitch angles at the ephemeris locations. [time,direction]

da2K – 2-dimensional numpy array of adiabatic invariant 'K' value associated with the pitch angles at the ephemeris locations. [time,direction]

da2Phi – 2-dimensional numpy array of adiabatic invariant 'Phi' associated with the pitch angles at the ephemeris locations. [time,direction]

da2Hmin – 2-dimensional numpy array of adiabatic invariant 'Hmin' associated with the pitch angles at the ephemeris locations. [time,direction]

da2Lstar – 2-dimensional numpy array of adiabatic invariant 'L*' associated with the pitch angles at the ephemeris locations. [time,direction]

daBmin – 1-dimensional numpy array of minimum IGRF model magnetic field strength value (nanoTeslas) along field line containing the ephemeris location. [time].

daBlocal – 1-dimensional numpy array of IGRF model magnetic field strength value (nanoTeslas) at the ephemeris location. [time]

daMagLT – 1-dimensional numpy array of magnetic local time (hours) at the Bmin positions. [time]

da2dB – 2-dimensional numpy array of the local magnetic field vector components at the ephemeris locations. [time,components]

da2dI – 2-dimensional numpy array of the local magnetic field current "I" value associated with the pitch angles at the ephemeris locations. [time,direction]

calcDirPitchAngles

(strCoordSys,
strCoordUnits,
daTimes,
daCoordX,
daCoordY,
daCoordZ,
da2DirX,
da2DirY,
da2DirZ)

Usage: Calculates the pitch angles corresponding to the input times, position and direction arrays.

Parameters:

strCoordSys – *Cartesian* coordinate system identifier: 'GEI','GEO','GSM','GSE','SM' or 'MAG';
Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – coordinate units, 'km' or 'Re'

daTimes – numpy array of time values, in Modified Julian Date form

daCoordX, daCoordY, daCoordZ – numpy arrays of position values, in the Cartesian coordinate system and units specified by the *strCoordSys* and *strCoordUnit* parameter values.

da2DirX, da2DirY, da2DirZ – 2-dimensional numpy arrays of direction values, in the Cartesian coordinate system specified by the *strCoordSys* parameter value. These direction values may be full magnitude or unit arrays. [time,direction]

Return values:

iErr – 0 = success, otherwise error

da2PitchAngles – 2-dimensional numpy array of pitch angles corresponding to the input time, position and direction information. [time,direction]

convertCoords

(strCoordSys,
strCoordUnits,
daTimes,
daCoord1,
daCoord2,
daCoord3,

strNewCoordSys,
strNewCoordUnits)

Usage: Converts the set of input times, position coordinates from one coordinate system to another.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

daTimes – numpy array of time values, in Modified Julian Date form

daCoord1, daCoord2, daCoord3 – numpy arrays of position values, in the coordinate system and units specified by the *strCoordSys* and *strCoordUnit* parameter values.

Consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' ordering of the input and output coordinate values for non-Cartesian coordinate systems.

strNewCoordSys – 'new' coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strNewCoordUnits – 'new' units: 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

Return values:

iErr – 0 = success, otherwise error

daNewCoord1, daNewCoord2, daNewCoord3 – numpy arrays of position values, in the 'new' coordinate system and units specified by the *strNewCoordSys* and *strNewCoordUnit* parameter values.

convertCoordsSingle

(strCoordSys,
strCoordUnits,
dTime,
dCoord1,
dCoord2,
dCoord3,
strNewCoordSys,
strNewCoordUnits)

Usage: Converts a single input time, position coordinates from one coordinate system to another.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

dTimes – time value, in Modified Julian Date form

dCoord1, dCoord2, dCoord3 – position values, in the coordinate system and units specified by the *strCoordSys* and *strCoordUnit* parameter values.

Consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' ordering of the input and output coordinate values for non-Cartesian coordinate systems.

strNewCoordSys – 'new' coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strNewCoordUnits – 'new' units: 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

Return values:

iErr – 0 = success, otherwise error

dNewCoord1, dNewCoord2, dNewCoord3 – position values, in the ‘new’ coordinate system and units specified by the *strNewCoordSys* and *strNewCoordUnit* parameter values.

RadEnvModel Class

Import file: radenvModel.py "`from radenvModel import RadEnvModel`"

This class is the entry point that provides direct programmatic access to the AE8, AP8, CRRESELE and CRRESPRO 'legacy' radiation belt models, providing 'mean' omni-directional particle flux values for the given time and position, with the specified model options and/or conditions.

General:

RadEnvModel

Usage: Instantiates an instance of the RadEnvModel class
Return value:
'RadEnvModel' class object

Model Parameter Inputs:

set_model

(strModel)
Usage: Specifies the name of the flux model to be used in the calculations.
Parameters:
strModel – model name: 'AE8', 'AP8', 'CRRESELE' or 'CRRESPRO'.
Return value:
iErr – 0 = success, otherwise error

set_modelDBDir

(strDataDir)
Usage: Specifies the directory that contains the collection IRENE model database files. The various database files required are automatically selected according to the model and parameters specified. The use of this method is highly recommended, as it *eliminates* the need for the other methods that specify the individual database files; those are only needed for using alternate or non-standard versions.
Parameters:
strDataDir – directory path for the IRENE database files.
Return value:
iErr – 0 = success, otherwise error

set_modelDBFile

(strModelDB)
Usage: Specifies the name of the database file (including path) for legacy flux model calculations. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).
This database name is in the form of '<path>/radiationBeltDB.h5'.
Parameters:
strModelDB – model database filename, including path.
Return value:
iErr – 0 = success, otherwise error

set_magfieldDBFile

(strMagfieldDB)

Usage: Specifies the name of the file (including path) for the magnetic field model database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).

This database name is in the form of '<path>/igrfDB.h5'.

Parameters:

strMagfieldDB – magnetic field model database filename, including path.

Return value:

iErr – 0 = success, otherwise error

---

get_model

Usage: Returns the name of the flux model, as specified in the *set_model*() method.

Return value:

strModel – model name string.

get_modelDBDir

Usage: Returns the directory name containing the collection of IRENE model database files that was specified in a previous call to the *set_modelDBDir*() method; otherwise, blank.

Return value:

strModelDBDir – model database directory.

get_modelDBFile

Usage: Returns the name of the database file for flux model calculations. This will be available immediately, when specified using the *set_modelDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *computeFlux*() method.

Return value:

strModelDB – model database filename.

get_magfieldDBFile

Usage: Returns the name of the file for the magnetic field model database. This will be available immediately, when specified using the *set_magfieldDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *computeFlux*() method.

Return value:

strMagfieldDB – *magnetic* field model database filename.

---

set_fluxType

(strFluxType)

Usage: Specifies the type of flux values to be calculated by the model.

Parameters:

strFluxType – flux type identifier: '1PtDiff'|'Differential'|'Diff' or 'Integral'

Return value:

iErr – 0 = success, otherwise error

set_energies

(daEnergies)

Usage: Specifies the set energies [MeV] at which the flux values are calculated by the selected model. Please consult User's Guide, Appendix A for valid ranges/values, which depend on the model selected.

Parameters:

daEnergies – numpy array of energy values, in units of MeV

Return value:

iErr – 0 = success, otherwise error

set_activityLevel

(strActivityLevel)

Usage: Specifies the geomagnetic activity level parameter for the CRRESPRO, AE8 or AP8 models.

Parameters:

strActivityLevel – geomagnetic activity level specification:
for CRRESPRO model, 'active' or 'quiet';
for AE8 or AP8 model, 'min' or 'max'.

Return value:

iErr – 0 = success, otherwise error

set_activityRange

(strActivityRange)

Usage: Specifies the geomagnetic activity level parameter for the CRRESELE model. Only one of the *set_activityRange*() or *set15DayAvgAp*() methods may be used, otherwise an error is flagged.

Parameters:

strActivityRange – geomagnetic activity level specification, in terms of Ap values:
'5-7.5', '7.5-10', '10-15', '15-20', '20-25', '>25', 'avg', 'max', or 'all'.

Return value:

iErr – 0 = success, otherwise error

set_15dayAp

(d15DayAvgAp)

Usage: Specifies the 15-day average Ap value for the CRRESELE model. Only one of the *set_activityRange*() or *set15DayAvgAp*() methods may be used, otherwise an error is flagged.

Parameters:

d15DayAvgAp – 15-day average Ap value.

Return value:

iErr – 0 = success, otherwise error

set_fixedEpoch

(iFixedEpoch)

Usage: Specifies the use of the model-specific fixed epoch (year) for the magnetic field model in the flux calculations. It is *highly recommended* to set this to 1 *(true)*. Unphysical results may be produced (especially at low altitudes) if set to 0 *(false)*.

Parameters:

iFixedEpoch – 1 *(true)* or 0 *(false)*; when *false*, the ephemeris year is used for the magnetic field model.

Return value:

iErr – 0 = success, otherwise error

set_shiftSAA

(iShiftSAA)

Usage: Shifts the SAA from its fixed-epoch location to the location for the current year of the ephemeris. This setting is ignored if the ***set_fixedEpoch*** method is set to 0 *(false)*.

Parameters:

iShiftSAA – 1 *(true)* or 0 *(false)*.

Return value:

iErr – 0 = success, otherwise error

---

get_fluxType

Usage: Returns the type of flux values to be calculated by the model.

Return value:

strFluxType – flux type identifier, as specified in the *set_fluxType*() method.

get_numEnergies

Usage: Returns the number of currently defined energies, as specified in the *set_energies*() method.

Return value:

iNumE – 0 or greater = number of energies, otherwise error

get_activityLevel

Usage: Returns the geomagnetic activity level parameter for the CRRESPRO, AE8 or AP8 Legacy model, as specified in the *set_activityLevel*() method.

Return value:

strActivityLevel – geomagnetic activity level specification string.

get_activityRange

Usage: Returns the geomagnetic activity level parameter for the CRRESELE Legacy model, as specified in the *set_activityRange*() method.

Return value:

strActivityRange – geomagnetic activity level specification string.

get_ get_15dayAp

Usage: Returns the 15-day average Ap value for the CRRESELE Legacy model, as specified in the *set_15dayAp*() method.

Return value:

d15DayAvgAp – 15-day average Ap value.

get_fixedEpoch

Usage: Returns the current setting for the use of the model-specific fixed epoch, as specified in the *set_fixedEpoch*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*.

get_shiftSAA

Usage: Returns the current setting of shifting the SAA from its fixed-epoch location, as specified in the *set_shiftSAA*() method.

Return value:

iVerdict – 1 *(true)* or 0 *(false)*.

---

set_coordSys

(strCoordSys,
strCoordUnits)

Usage: Specifies the coordinate system and units for the position values that are specified by the *set_ephemeris*() method. When not specified, these settings default to 'GEI' and 'Re'. "Re" = radius of the Earth, defined as 6371.2 km.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL';
Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value

Return value:

iErr – 0 = success, otherwise error

set_ephemeris

(daTimes,
daCoords1,
daCoords2,
daCoords3)

Usage: Specifies the ephemeris time and positions to be used for the model calculations.

Parameters:

daTimes – numpy array of time values, in Modified Julian Date form. May be identical times or times in chronological order, associated with position coordinates.

daCoords1, daCoords2, daCoords3 – numpy arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system and units specified by *set_coordSys*().

Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' order of the coordinate values for non-Cartesian coordinate systems.

Return value:

iErr – 0 = success, otherwise error

---

get_coordSys

Usage: Returns the coordinate system name, as specified in the *set_coordSys*() method.
Return value:
strCoordSys – coordinate system name.

get_coordSysUnits

Usage: Returns the coordinate system units, as specified in the *set_coordSys*() method.
Return value:
strCoordSysUnits – coordinate system units ('Re' or 'km').

get_numEphemeris

Usage: Returns the number of currently defined ephemeris entries, as specified in the *set_ephemeris*() method.
Return value:
iNumE – 0 or greater = number of ephemeris entries, otherwise error

Model Execution and Results:

computeFlux

(da2FluxData)
Usage: Returns the mean model flux from the specified model, based on the various model parameter inputs.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).
Parameters:
da2FluxData – 2-dimensional numpy array of the mean flux values. [time, energy]
Return value:
iErr – 0 = success, otherwise error

CammiceModel Class

Import file: cammiceModel.py "`from cammiceModel import CammiceModel`"

This class is the entry point that provides direct programmatic access to CAMMICE/MICS 'legacy' plasma particle model. This model is set to produce flux values for twelve pre-defined energy bins: 1.0-1.3, 1.8-2.4, 3.2-4.2, 5.6-7.4, 9.9-13.2, 17.5-23.3, 30.9-41.1, 54.7-72.8, 80.3-89.7, 100.1-111.7, 124.7-139.1, 155.3-193.4 **keV**). *The returned flux results cannot be used as for dose calculations.*

General:

CammiceModel

Usage: Instantiates an instance of the CammiceModel class
Return value:
'CammiceModel' class object

Model Parameter Inputs:

set_modelDBDir

(strDataDir)

Usage: Specifies the directory that contains the collection IRENE model database files. The various database files required are automatically selected according to the model and parameters specified. The use of this method is highly recommended, as it *eliminates* the need for the other methods that specify the individual database files; those are only needed for using alternate or non-standard versions.
Parameters:
strDataDir – directory path for the IRENE database files.
Return value:
iErr – 0 = success, otherwise error

set_modelDBFile

(strModelDB)

Usage: Specifies the name of the database file (including path) for legacy flux model calculations. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).
This database name is in the form of '<path>/cammiceDB.h5'.
Parameters:
strModelDB – model database filename, including path
Return value:
iErr – 0 = success, otherwise error

set_magfieldDBFile

(strMagfieldDB)

Usage: Specifies the name of the file (including path) for the magnetic field model database. The use of this method is *not needed* when *set_modelDBDir*() is specified, except when an alternate or non-standard database file is to be used (this overrides any automatic file specification).
This database name is in the form of '<path>/igrfDB.h5'.
Parameters:
strMagfieldDB – magnetic field model database filename, including path

Return value:
iErr – 0 = success, otherwise error

get_modelDBDir

Usage: Returns the directory name containing the collection of IRENE model database files that was specified in a previous call to the *set_modelDBDir*() method; otherwise, blank.
Return value:
strModelDBDir – model database directory.

get_modelDBFile

Usage: Returns the name of the database file for flux model calculations. This will be available immediately, when specified using the *set_modelDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *computeFlux*() methods.
Return value:
strModelDB – model database filename.

get_magfieldDBFile

Usage: Returns the name of the file for the magnetic field model database. This will be available immediately, when specified using the *set_magfieldDBFile*() method. When the *set_modelDBDir*() method is used, the automatically determined filename will be available after a call to the *computeFlux*() method.
Return value:
strMagfieldDB – *magnetic* field model database filename.

set_magfieldModel

(strMFModel)
Usage: Specifies the magnetic field option for the CAMMICE model run. 'igrf' uses the IGRF model without an external field model. 'igrfop' adds Olson-Pfitzer/Quiet as the external field model.
Parameters:
strMFModel – magnetic field model specification: 'igrf' or 'igrfop'.
Return value:
iErr – 0 = success, otherwise error

set_dataFilter

(strDataFilter)
Usage: Specifies the data filter option for the CAMMICE model run. 'Filtered' excludes data collected during periods when the DST index was below -100.
Parameters:
strDataFilter – data filter specification: 'all' or 'filtered'.
Return value:
iErr – 0 = success, otherwise error

set_pitchAngleBin

(strPitchAngleBin)
Usage: Specifies the pitch angle bin for the CAMMICE model run.

Parameters:

strPitchAngleBin – pitch angle bin identification: '0-10','10-20','20-30','30-40','40-50','50-60','60-70','70-80','80-90', '90-100','100-110','110-120','120-130','130-140','140-150','150-160','160-170','170-180' or 'omni'.

Return value:

iErr – 0 = success, otherwise error

set_species

(strSpecies)

Usage: Specifies the (single) particle species for the CAMMICE model run.

Parameters:

strSpecies – species identification: 'H+', 'He+', 'He+2', 'O+', 'H', 'He', 'O', or 'Ions'.

Return value:

iErr – 0 = success, otherwise error

set_coordSys

(strCoordSys,
strCoordUnits)

Usage: Specifies the coordinate system and units for the position values that are specified by the *set_ephemeris*() method. When not specified, these settings default to 'GEI' and 'Re'. "Re" = radius of the Earth, defined as 6371.2 km.

Parameters:

strCoordSys – coordinate system identifier: 'GEI','GEO','GDZ','GSM','GSE','SM','MAG','SPH' or 'RLL'; Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

strCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value

Return value:

iErr – 0 = success, otherwise error

set_ephemeris

(daTimes,
daCoords1,
daCoords2,
daCoords3)

Usage: Specifies the ephemeris time and positions to be used for the model calculations.

Parameters:

daTimes – numpy array of time values, in Modified Julian Date form. May be identical times or times in chronological order, associated with position coordinates.

daCoords1, daCoords2, daCoords3 – numpy arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system and units specified by *set_coordSys*().

Please consult the User's Guide document, "Supported Coordinate Systems" for more details; in particular, note the expected 'standard' order of the coordinate values for non-Cartesian coordinate systems.

Return value:

iErr – 0 = success, otherwise error

get_numEphemeris

Usage: Returns the number of currently defined ephemeris, as specified in the *set_ephemeris*() method.

Return value:

iNumE – 0 or greater = number of ephmeris, otherwise error.

get_magfieldModel

Usage: Returns the magnetic field option for the CAMMICE model run, as specified in the *set_magfieldModel*() method.

Return value:

strMFModel – magnetic field model specification.

get_dataFilter

Usage: Returns the data filter option for the CAMMICE model run, as specified in the *set_dataFilter*() method.

Return value:

strDataFilter – data filter specification.

get_pitchAngleBin

Usage: Returns the pitch angle bin for the CAMMICE model run, as specified in the *set_pitchAngleBin*() method.

Return value:

strPitchAngleBin – pitch angle bin identification.

get_species

Usage: Returns the (single) particle species for the CAMMICE model run, as specified in the *set_species*() method..

Return value:

strSpecies – species identification.

get_coordSys

Usage: Returns the coordinate system name, as specified in the *set_coordSys*() method.

Return value:

strCoordSys – coordinate system name.

get_coordSysUnits

Usage: Returns the coordinate system units, as specified in the *set_coordSys*() method.

Return value:

strCoordSysUnits – coordinate system units ('Re' or 'km').

Model Execution and Results:

computeFlux

(da2FluxData)

Usage: Returns the mean model flux from the CAMMICE model, based on the various model parameter inputs.

Parameters:

da2FluxData – 2-dimensional numpy array of the mean flux values. [time, energy]

Return value:

iErr – 0 = success, otherwise error

DateTimeUtil Class

Import file: dateTime.py "`import dateTime`"

This class is the entry point that provides direct programmatic access to date and time conversion utilities.

Utility Results:

get_gmtsec

(iHours,
iMinutes,
dSeconds)

Usage: Determines the GMT seconds of day for the input hours, minutes and seconds.
Parameters:
iHours – hours of day (0-23)
iMinutes – minutes of hour (0-59)
dSeconds – seconds of minute (0-59.999)
Return value:
dMjd – GMT seconds of day

get_dayYear

(iYear,
iMonth,
iDay)

Usage: Determines the day number of year for the input year, month and day number.
Parameters:
iYear – year (1950-2049)
iMonth – month (1-12)
iDay – day of month (1-28|29|30|31)
Return value:
iDayYear – day number of year

get_modJulDate

(iYear,
iDdd,
dGmtsec)

Usage: Determines the Modified Julian Date for the input year, day of year and GMT seconds.
Parameters:
iYear – year (1950-2049)
iDdd – day of year (1-365|366)
dGmtsec – GMT seconds of day (0-86399.999)
Return value:
dMjd – Modified Julian Date (33282.0 - 69806.999)

get_modJulDateUnix

(iUnixTime)

Usage: Determines the Modified Julian Date for the input UNIX time value.

(due to limitations of Unix time, this will be valid only between 01 Jan 1970 – 19 Jan 2038).
Parameters:
iUnixTime – Unix Time, in seconds from 01 Jan 1970, 0000 GMT; (0 – MaxInt)
Return value:
dMjd – Modified Julian Date (40587.0 - 65442.134)

get_dateTime

(dModJulDate)
Usage: Determines the year, day of year and GMT seconds for the input Modified Julian Date.
Parameters:
dModJulDate – Modified Julian Date (33282.0 - 69806.999)
Return values:
iYear – year (1950-2049)
iDdd – day of year (1-365|366)
dGmtsec – GMT seconds of day (0-86399.999)

get_hms

(dGmtsec)
Usage: Determines the hours, minutes and seconds for the input GMT seconds.
Parameters:
dGmtsec – GMT seconds of day (0-86399.999)
Return values:
iHours – hours of day (0-23)
iMinutes – minutes of hour (0-59)
dSeconds – seconds of minute (0-59.999)

get_monthDay

(iYear,
iDdd)
Usage: Determines the month and day number for the input year and day of year.
Parameters:
iYear – year (1950-2049)
iDdd – day of year (1-365|366)
Return values:
iMonth – month (1-12)
iDay – day of month (1-28|29|30|31)

To contact the IRENE (AE9/AP9/SPM) model development team, email ae9ap9@vdl.afrl.af.mil .

The IRENE model package and related information can be obtained from AFRL's Virtual Distributed Laboratory (VDL) website: https://www.vdl.afrl.af.mil/programs/ae9ap9