AFRL-RV-PS-
TR-2014-0018

AE9/AP9/SPM MODEL APPLICATION PROGRAMMING INTERFACE, VERSION 1.00.000

Paul Whelan

Atmospheric and Environmental Research, Inc.
131 Hartwell Ave.
Lexington, MA 02421

18 February 2014

Technical Report

AIR FORCE RESEARCH LABORATORY
Space Vehicles Directorate
3550 Aberdeen Ave SE
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776

DTIC COPY

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 377 ABW Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RV-PS-TR-2014-0018 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

//SIGNED//	//SIGNED//
Michael J. Starks, DR-III	Edward J. Masterson, Colonel, USAF
Senior Electrical Engineer, RVBXR	Chief, Battlespace Environment Division

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
18-02-2014	Technical Report	01 Jan 2009 to 31 Jan 2012

4. TITLE AND SUBTITLE
AE9/AP9/SPM MODEL APPLICATION PROGRAMMING INTERFACE, VERSION 1.00.000

5a. CONTRACT NUMBER
FA9453-12-C-0231

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
63401F

6. AUTHOR(S)
Paul Whelan

5d. PROJECT NUMBER
5021

5e. TASK NUMBER
PPM00012091

5f. WORK UNIT NUMBER
EF007946

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Atmospheric and Environmental Research, Inc.
131 Hartwell Ave.
Lexington, MA 02421

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Space Vehicles Directorate
3550 Aberdeen Avenue SE
Kirtland AFB, NM 87117-5776

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RVBXR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-RV-PS-TR-2014-0018

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited. (377ABW-2012-1187 dtd 5 Sep 2012)

13. SUPPLEMENTARY NOTES

14. ABSTRACT
While the AE9/AP9/SPM Radiation and Plasma Environment Model is distributed with both GUI and command-line applications for running the model, there are situations where it is more appropriate to more directly integrate other applications with the model. The model is distributed with an Application Programming Interface (API) suite for such situations. This report provides documentation on the API suite for AE9/AP9/SPM, covering installation and setup and details on the APIs for C++, C, and FORTRAN.

15. SUBJECT TERMS
AE9/AP9/SPM, radiation belt model, space plasma model, application programming interface

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Michael Starks
Unclassified	Unclassified	Unclassified	Unlimited	46	19b. TELEPHONE NUMBER *(include area code)*

This page is intentionally left blank.

AE9/AP9/SPM Model Application Programming Interface, Version 1.00.000

Table of Contents

This page is intentionally left blank.

Overview

The AE-9/AP-9 Radiation Belt 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 AE-9/AP-9 model calls and data directly into a new or existing application, an Application Programming Interface (API) is also distributed with the model.

The AE-9/AP-9 Radiation Belt model supports programmatic access through a suite of APIs accessible from a number of programming languages. The model is written in C++ and direct access to all classes and methods of the model is available using the source distribution of the model. Additional APIs are provided through a set of C and Fortran wrapper methods at the highest level of the model, using a set of "fly-in" routines modeled after those found in the Irbemlib API.

Getting Started

Installation and Setup

AE-9/AP-9 is distributed as a zip file that comes with a pre-built Windows 32-bit binary distribution of the model and a complete set of source for building API libraries, as well as binary distributions for other platforms. To generate a set of libraries, shared objects and/or dll files, please refer to the build instructions provided with the distribution in Ae9Ap9/documents/ Build_Instructions_for_AE9AP9.pdf.

Note that the source distribution build process uses CMake to generate an "out-of-source" build. That means that output binary and library files are located in a separate directory structure from the source files. This is done to facilitate builds for multiple platforms, as well as debug and release builds for any given platform. However, the ramification of this is that the location of libraries and header files will depend on the choices made during the build process and may vary. This is reflected in the library paths shown below through the use of curly brackets '{}'.

The following directories within the AE-9AP-9 distribution will contain header files that may be required to compile an application that utilizes the AE-9AP-9 libraries:

… Ae9Ap9/source/Ae9Ap9/trunk/models/include
… Ae9Ap9/source/SpWx_Ae9Ap9/Common/include
… Ae9Ap9/source/SpWx_Ae9Ap9/Models/include

After performing a source build, the following directories within the AE-9AP-9 distribution will contain library, shared object and/or dll files that may be required to link and run an application with AE-9AP-9 libraries.

… Ae9Ap9/source/build_{linux|win32}/Ae9Ap9/trunk/lib{/debug|release}
… Ae9Ap9/source/build_{linux|win32}/Ae9Ap9_SpWx/Models/lib{/windows/Debug.Net|Release.Net}

Shared Objects, DLLs and Static Libraries

Applications can link to the AE-9 model using static libraries or through DLLs under Windows or shared object files under Linux operating systems. To run an application that references the AE-9 through either DLL or shared object implementations, those binaries must be moved to a directory specified in the PATH (Windows) or LD_LIBRARY_PATH (Linux) environment variables. Alternatively, the directories in which they reside can be added to those path specifications. Those directories are as follows:

…Ae9Ap9/source/build_win32/Ae9Ap9/trunk/bin{/Debug|Release} - ae9ap9.dll
…Ae9Ap9/source/build_linux/Ae9Ap9/trunk/lib - libae9ap9.so
…Ae9Ap9/source/build_linux/Ae9Ap9_SpWx/Models/lib - various shared objects

C++ API

The AE-9/AP-9 application is written in C++ and all source and header files are provided with the distribution. This gives the client application developer a great deal of freedom in determining at what level and granularity to integrate with the model. However, for most applications it is recommended that client applications access the model through the top level "Application" layer. The application layer consists of the Ae9Ap9Application and AEOrbitPropagator classes, along with a small suite of optional aggregation classes derived from the AEAggregator class. These classes and their methods are described in detail in the C++ API Reference section of this document.

A note to Windows C++ developers: The AE-9/AP-9 classes heavily utilize STL containers and classes as method parameters. This makes it problematic to expose these methods across a dll boundary. Only functions of the C API are exposed from the ae9ap9.dll. Therefore, it is recommended that on the Windows platform, C++ client applications either link statically to AE-9/AP-9 libraries or call the model through the C interface methods to use the dll. On Linux platforms, client applications can use the C++ API to call directly into the shared object implementation of the model.

C API

The AE-9/AP-9 C language API is provided for access to the model from C and other compatible language client applications. It can also be used to access the DLL implementation of the model on the Windows platform from C++ applications. The C API utilizes client allocated array data structures in place of collection classes used by the underlying C++ interface and classes. Note that the C API assumes that all buffers for return data are allocated at the client application level and that relevant sizes are passed into the API by the calling routine. This is intended to eliminate any ambiguity as to responsibility for memory allocation and deallocation and to reduce or eliminate memory leaks. The AE-9/AP-9 C language API is defined in the file AECInterface.h. The methods of this interface are described in detail in the C API Reference section of this document.

Fortran API

The AE-9/AP-9 Fortran language API is provided for access to the model from Fortran and other compatible language client applications. The Fortran API is comparable to the C API, however, native

Fortran applications pass data by reference and use different calling conventions. The Fortran API also utilizes client allocated array data structures in place of collection classes used by the underlying C++ interface and classes. Note that the Fortran API assumes that all buffers for return data are allocated at the client application level and that relevant sizes are passed into the API by the calling routine. This is intended to eliminate any ambiguity as to responsibility for memory allocation and deallocation and to reduce or eliminate memory leaks. The AE-9/AP-9 Fortran language API is defined in the file AEFInterface.h. The methods of this interface are described in detail in the Fortran API Reference section of this document.

3rd Party Dependencies

The AE-9/AP-9 model has dependencies on the external 3rd party libraries Boost® (for linear algebra functions and data structures) and on HDF5® (for internal databases). Please refer to the Build_Instructions_for_AE9AP9.pdf document in the Ae9Ap9/documents directory for details on installation of these libraries. It is likely that include and library directories from these installations will need to be added to build settings for client applications and that shared binaries will need to be added to the appropriate system paths.

API Reference

C++ API

Ae9Ap9Application Class

Header file: Ae9Ap9Application.h

Description: This class is the main entry point into the application layer of the Ae9Ap9 project. The Ae9Ap9Application class provides a suite of methods that provide synchronous access to the underlying Ae9Ap9 model. Client applications requiring multi-processing capabilities should access the model through the underlying Ae9Ap9Model class.

Public methods:

Ae9Ap9Application

Usage: Default constructor

Parameters: none
Return values: none

~Ae9Ap9Application

Usage: Destructor

Parameters: none

Return values: none

void setModelDataSource

(const string& strDataSource)

Usage: Set the path and file name of the hdf5 format database containing the data of the selected model. (ie: for high energy electrons, pass "{path}/ AE9V10_runtime_tables.mat") Call this once at startup to initialize the model.

Parameters:
strDataSource – path and file name of the hdf5 database

Return values: none

void setKPhiNeuralNetDataSource
(const string& strDataSource)

Usage: Set the path and file name of the hdf5 format database containing the K/Phi space neural network used by the selected model. (ie: pass "{path}/ fastPhi_net.mat") Call this once at startup to initialize the model.

Parameters:
strDataSource – path and file name of the hdf5 neural net database

Return values: none

void setKHMinNeuralNetDataSource
(const string& strDataSource)

Usage: Set the path and file name of the hdf5 format database containing the K/Hmin space neural network used by the selected model. (ie: pass "{path}/ fast_hmin_net.mat") Call this once at startup to initialize the model.

Parameters:
strDataSource – path and file name of the hdf5 neural net database

Return values: none

void setMagfieldModelDataSource
(const string& strDataSource)

Usage: Set the path and file name of the hdf5 format database containing the magnetic field model data used by the selected model. (ie: pass "{path}/ igrfDB.h5") . Call this once at startup to initialize the model.

Parameters:
strDataSource – path and file name of the hdf5 database

Return values: none

int setFluxEnvironment

(eModelType, eFluxType, vvdEnergies,
vdTimes, eCoordSys, vdCoordsAxis1,
vdCoordsAxi2, vdCoordsAxis3)

Usage: This method computes flux weights for a grouping of satellite positions at a given set of energies and times when computing omnidirectional flux. It should be called once prior to calling any combination of fly-in flux computation methods for that time period. Note: use as large a grouping of satellite positions as can be processed on available hardware.

Parameters:

eModel - type of model to be run: eAEModelElectron, eAEModelProton, eAEModelSpecies

eFluxType - enum defining type of flux to compute (1 point differential, 2 point differential, integral) see AEEnums.h

vvdEnergies - 2 dimensional vector of doubles defining energy levels (MeV) at which to compute flux. Note: column 2 used only for 2 pt differential flux type, which requires computation of flux between two energy levels

vdTimes - vector of date/times in modified julian date format at which to compute flux. Lengths of time and position vectors must match, with time corresponding to a position at that index in the vectors.

eCoordSys - enum defining the coordinate system of positions and directions

vdCoordsAxis1,

vdCoordsAxis2,

vdCoordsAxis3 - Coordinates along axes in 3d space in the eCoordSys coordinate system of each position at which flux is to be computed. These vectors should match the vdTimes vector in length and correspond to those times at each position.

Return values:

int – 0 success, else (see AEErrors.h)

int setFluxEnvironment

(eModelType, eFluxType, vvdEnergies,
vdTimes, eCoordSys, vdCoordsAxis1,
vdCoordsAxi2, vdCoordsAxis3,
vvdFluxDir1, vvdFluxDir2, vvdFluxDir3)

Usage: This method computes flux weights for a grouping of satellite positions at a given set of energies and times when computing directional flux. This method provides an alternative to computing a single omnidirectional flux. It allows the user to specify one or more flux directions at

each time and position. The flux for each combinatin of passed time, position and direction will be computed.

Flux directions should be passed as x,y,z components of unit vectors using the passed coordinate system. This method should be called once prior to calling any combination of fly-in flux computation methods for that time period. Note: use as large a grouping of satellite positions as can be processed on available hardware.

Parameters:

eModel - type of model to be run: eAEModelElectron, eAEModelProton, eAEModelSpecies

eFluxType - enum defining type of flux to compute (1 point differential, 2 point differential, integral) see AEEnums.h

vvdEnergies - 2 dimensional vector of doubles defining energy levels (MeV) at which to compute flux. Note: column 2 used only for 2 pt differential flux type, which requires computation of flux between two energy levels

vdTimes - vector of date/times in modified julian date format at which to compute flux. Lengths of time and position vectors must match, with time corresponding to a position at that index in the vectors.

eCoordSys - enum defining the coordinate system of positions and directions

vdCoordsAxis1,
vdCoordsAxis2,
vdCoordsAxis3 - Coordinates along axes in 3d space in the eCoordSys coordinate system of each position at which flux is to be computed. These vectors should match the vdTimes vector in length and correspond to those times at each position.

vvdFluxDir1,
vvdFluxDir2,
vvdFluxDir3 - Directions at which to compute flux at each timestep in vdTimes. Multiple directions can be computed at each timestep. Thus, these vectors are 2 dimensional (nTimes,nDirections).

Return values:

int – 0 success, else (see AEErrors.h)

int flyInMean
(vvdvector& vvvdFluxData)

Usage: This method computes mean flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Bad values are returned as AE_NaN (defined in AEErrors.h). The function ae9ap9::isnan(value) should be used to test for bad values.

Parameters:

vvvdFluxData –a returned 3 dimensional vector (time, energy, direction) of flux values (in MeV)

Return Values:
int – 0 success, else (see AEErrors.h)

int flyInPerturbedMean
(int iScenario, vvdvector& vvvdFluxData)

Usage: This method computes perturbed mean flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Perturbed means are a statistical distribution of mean flux based on measurement uncertainty only. Bad values are returned as AE_NaN (defined in AEErrors.h). The function ae9ap9::isnan(value) should be used to test for bad values.

Parameters:
iScenario – perturbed mean scenario number (1..999) for repeatability
vvvdFluxData – a returned 3 dimensional vector (time, energy, direction) of flux values (in MeV)

Return Values:
int – 0 success, else (see AEErrors.h)

int flyInPercentile
(int iPercentile, vvdvector& vvvdFluxData)

Usage: This method computes percentile flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Bad values are returned as AE_NaN (defined in AEErrors.h). The function ae9ap9::isnan(value) should be used to test for bad values.

Parameters:
iPercentile – percentile (1..99) flux to compute at each time, energy and position
vvvdFluxData – a returned 3 dimensional vector (time, energy, direction) of percentile flux values (in MeV)

Return Values:
int – 0 success, else (see AEErrors.h)

int flyInScenario
(dEpoch, iScenario, vvvdFluxData, bPerturbFluxMap = true)

Usage: This method computes monte carlo flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Monte carlo fluxes are a statistical distribution of flux based on both

measurement and temporal uncertainty. Bad values are returned as AE_NaN (defined in AEErrors.h). The function ae9ap9::isnan(value) should be used to test for bad values.

Parameters:

dEpoch – start date/time (in Modified Julian Date) for the scenario
iScenario – monte carlo scenario number (1..999) for repeatability
vvdFluxData - a returned 3 dimensional vector (time, energy, direction) of percentile flux values (in MeV)
bPerturbFluxMap – if false, turns off measurement uncertainty mean perturbation. Do Not Use. For testing purposes only.

Return Values:

int – 0 success, else (see AEErrors.h)

const string& getErrorText

Usage: Retrieves the text associated with a non-zero returned error code

Parameters: none

Return Values:

const string& - text associated with a previously returned error code

AEOrbitPropagator Class

Header file: AEOrbitPropagator.h

Description: This class encapsulates three choices of underlying orbit propagator implementations; an SGP4-based propagator, a SatEph implementation and a Kepler+J2 only propagator. Clients of this class can choose which to use.

Public methods:

AEOrbitPropagator

Usage: Default constructor

Parameters: none

Return values: none

~AEOrbitPropagator

Usage: Destructor

Parameters: none

Return values: none

void setPropagatorTypeSGP4

Usage: Forces use of the SGP4 orbit propagator (default)

Parameters: none

Return values: none

void setPropagatorTypeSatEph

Usage: Forces use of the SatEph orbit propagator

Parameters: none

Return values: none

void setPropagatorTypeKepler

Usage: Forces use of the Kepler-J2 orbit propagator

Parameters: none

Return values: none

void useSGP4ImprovedMode

(bool bUse)

Usage: Enables or disables use of the “improved mode” when using the SGP4 propagator

Parameters:
bUse – if true, enables “improved mode”, else uses standard mode

Return values: none

void setSGP4WGSConst72Old

Usage: Sets use of the WGS 1972 Old mode constant datum for SGP4

Parameters: none

Return values: none

void setSGP4WGSConst72

Usage: Sets use of the WGS 1972 newer constant s when using SGP4

Parameters: none

Return values: none

void setSGP4WGSConst84

Usage: Sets the use of WGS 1984 datum constants when using SGP4

Parameters: none

Return values: none

void setTLEFile
(const string& strTLEFile)

Usage: Sets the name of a TLE (two line element) file to be used in orbit propagation

Parameters:
strTLEFile – path and file name of the TLE file to read

Return values: none

void clearTLEFields

Usage: Resets all internal TLE-related fields to default values

Parameters: none

Return values: none

void addTLE

(const string& strTLELine1,
long lSatelliteNumber,
char cClass,
int iIntDesYear,
int iLaunchNumber,
const string& strPiece,
int iEpochYear,
double dEpochDayOfYear,
double dMeanMotion1stDeriv,
double dMeanMotion2ndDeriv,
double dBStar,
int iEphemType,
int iElementNumber,
const string& strTLELine2,
double dInclination,
double dRightAscension,
double dEccentricity,
double dArgofPerigee,
double dMeanAnomaly,
double dMeanMotion,
long lEpochRevolution
)

Usage: Adds a set of fields defining a two line element to an internal collection of TLEs to be used in orbit propagation

Parameters:

strTLELine1 – string representing line 1 of the two line element
lSatelliteNumber – five digit satellite number
cClass – character classification 'U' for unclassified
iIntDesYear – last two digits of launch year
iLaunchNumber – launch number of the year
strPiece – piece of the launch
iEpochYear – last two digits of the year

dEpochDayOfYear – day and fractional day
dMeanMotion1stDeriv – first derivative of mean motion
dMeanMotion2ndDeriv – second derivative of mean motion
dBStar – b star drag term
iEphemType – ephemeris type
iElementNumber – element number
strTLELine2 – string representing line 2 of the two line element
dInclination – inclination in degrees
dRightAscension – right ascension of ascending node in degrees
dEccentricity - eccentricity
dArgofPerigee – argument of perigee in degrees
dMeanAnomaly – mean anomaly in degrees
dMeanMotion – mean motion in revs per day
lEpochRevolution – revolution number at epoch

Return values: none

void setTLEs

```
( const vector<string>& vstrTLELine1s,
 const vector<long>&   vlSatelliteNumbers,
 const vector<char>&   vcClasses,
 const vector<int>&    viIntDesYears,
 const vector<int>&    viLaunchNumbers,
 const vector<string>& vstrPieces,
 const vector<int>&    viEpochYears,
 const vector<double>& vdEpochDaysOfYear,
 const vector<double>& vdMeanMotion1stDerivs,
 const vector<double>& vdMeanMotion2ndDerivs,
 const vector<double>& vdBStars,
 const vector<int>&    viEphemTypes,
 const vector<int>&    viElementNumbers,
 const vector<string>& vstrTLELine2s,
 const vector<double>& vdInclinations,
 const vector<double>& vdRightAscensions,
 const vector<double>& vdEccentricities,
 const vector<double>& vdArgsofPerigee,
 const vector<double>& vdMeanAnomalies,
 const vector<double>& vdMeanMotions,
 const vector<long>&   vlEpochRevolutions
 )
```

Usage: Uses a set of field vectors defining two line elements to populate an internal collection of TLEs to be used in orbit propagation

Parameters:
vstrTLELine1s – vector of strings representing line 1 of the two line elements
vlSatelliteNumbers – vector of five digit satellite numbers
vcClasses – vector of character classification 'U' for unclassified
viIntDesYears – vector of last two digits of launch year
viLaunchNumbers – vector of launch number of the year
vstrPieces – vector of piece of the launch
viEpochYears – vector of last two digits of the year
vdEpochDaysOfYear – vector of day and fractional day
vdMeanMotion1stDerivs – vector of first derivative of mean motion
vdMeanMotion2ndDerivs – vector of second derivative of mean motion
vdBStars – vector of b star drag terms
viEphemTypes – vector of ephemeris types
viElementNumbers – vector of element numbers
vstrTLELine2s – vector of strings representing line 2 of the two line element
vdInclinations – vector of inclination in degrees
vdRightAscensions – vector of right ascension of ascending node in degrees
vdEccentricities – vector of eccentricities
vdArgsofPerigee – vector of arguments of perigee in degrees
vdMeanAnomalies – vector of mean anomalies in degrees
vdMeanMotions – vector of mean motions, in revs per day
vlEpochRevolutions – vector of revolution numbers at epoch

Return values: none

void setMeanElements

(double dElementTimeMJD,
double dInclination,
double dArgofPerigee,
double dMeanAnomaly,
double dMeanMotion1stDeriv,
double dEccentricity,
double dRightAscension,
double dMeanMotion,
double dMeanMotion2ndDeriv,
double dBStar)

Usage: Sets the required mean orbital elements when using orbital elements to drive the orbit propagator.

Parameters:
dElementTimeMJD – date/time (MJD) of the orbital elements
dInclination - inclination in degrees
dArgofPerigee - argument of perigee in degrees
dMeanAnomaly – mean anomaly in degrees

dMeanMotion1stDeriv – first derivative of mean motion
dEccentricity - eccentricity
dRightAscension - right ascension of ascending node in degrees
dMeanMotion – mean motion in revs per day
dMeanMotion2ndDeriv – second derivitive of mean motion
dBStar – b star drag term

Return values: none

void setTimes

(double dStartMJD, double dEndMJD, double dTimestepSecs)

Usage: Sets the time range and cadence for calculating ephemeris

Parameters:
dStartMJD – start date/time (in MJD) of the ephemeris timeframe
dEndMJD – end date/time (in MJD) of the ephemeris timeframe
dTimestepSecs – cadence (in seconds) at which to calculate ephemeris

Return values: none

void setTimes

(const vector<double>& vdTimesMJD)

Usage: Sets times at which to generate ephemeris when using TLEs to drive the orbit propagator

Parameters:
vdTimesMJD – date/times (in MJD) at which to generate ephemeris

Return values: none

void setOrbitType

(const string& strOrbit)

Usage: Sets the type of orbit to compute for the Kepler/J2 propagator. Valid values are 'classical', 'mean','solar','rv','geosync'. This setting dictates which orbital elements are required for input. See the Ae9Ap9 User's Guide for details.

Parameters:
strOrbit – text string representing above Kepler orbit types

Return values: none

void setOrbitalElementEpoch

(double dEpochMJD)

Usage: Sets the date and time for the passed orbital element (Kepler only)

Parameters:
dEpochMJD – date /time (MJD) of orbital element epoch

Return values: none

void setUseJ2Perturbations

(bool bUseJ2)

Usage: Enables or disables computation of J2 perturbations (Kepler only)

Parameters:
bUseJ2 – boolean indicating whether to use J2 perturbation computation

Return values: none

void setInclination

(double dInclinationInDeg)

Usage: Sets the orbital inclination in degrees (Kepler only)

Parameters:
dInclinationInDeg – orbital inclination in degrees

Return values: none

void setEccentricity

(double dEccentricity)

Usage: Sets the eccentricity of the orbit (Kepler only)

Parameters:
dEccentricity – eccentricity of orbit

Return values: none

void setArgOfPerigee

(double dArgOfPerigeeInDeg)

Usage: Sets the argument of perigee in degrees (Kepler only)

Parameters:
dArgOfPerigeeInDeg – argument of perigee (Dg)

Return values: none

void setMeanAnomalyAtEpoch

(double dMeanAnomalyInDeg)

Usage: Sets the mean anomaly at the epoch time (Dg) (Kepler only)

Parameters:
dMeanAnomalyInDeg – mean anomaly in degrees

Return values: none

void setMeanMotion

(double dMeanMotionInRevPerDay)

Usage: Sets the mean motion (rev/day) (Kepler only)

Parameters:
dMeanMotionInRevPerDay – mean motion in rev/day

Return values: none

void setRightAscension

(double dLongitudeInDeg)

Usage: Sets the right ascension of ascending node in degrees (Kepler only)

Parameters:
dLongitudeInDeg –longitude of ascending node (Dg)

Return values: none

void setAltitudeOfPerigee

(double dAltitudeInKm)

Usage: Sets the altitude (km) at perigee (Kepler only)

Parameters:
dAltitudeInKm – altitude in km at perigee

Return values: none

void setAltitudeOfApogee

(double dAltitudeInKm)

Usage: Sets the altitude (km) at apogee (Kepler only)

Parameters:
dAltitudeInKm – altitude in km at apogee

Return values: none

void setLocalTimeOfApogee

(double dLocalTimeInHours)

Usage: Sets the local time (Hrs) at apogee (Kepler only)

Parameters:
dLocalTimeInHours – local time (hrs) at apogee

Return values: none

void setLocalTimeOfMaxInclination

(double dLocalTimeInHours)

Usage: Sets the local time (hrs) at max inclination (Kepler only)

Parameters:
dLocalTimeInHours – local time (hrs) at max inclination

Return values: none

void setTimeOfPerigee

(double dTimeMJD)

Usage: Sets the time (MJD) of perigee (Kepler only)

Parameters:
dTimeMJD – time in MJD of perigee

Return values: none

void setSemimajorAxis

(double dAxisInRe)

Usage: Sets the length (RE) of the orbit semimajor axis (Kepler only)

Parameters:
dAxisInRe – length in RE of the semimajor axis

Return values: none

void setPositionGEI

(double dX, double dY, double dZ)

Usage: Sets the position (GEI) at epoch (orbital element time) (Kepler only)

Parameters:
dX, dY, dZ – vector defining position in GEI at epoch

Return values: none

void setVelocityGEI

(double dU, double dV, double dW)

Usage: Sets the velocity (GEI) at epoch (orbital element time) (Kepler only)

Parameters:
dU, dV, dW – vector defining velocity in GEI at epoch

Return values: none

void setGeosyncLongitude

(double dLongitudeDeg)

Usage: Sets the longitude (Dg) for a geosynchronous orbit (Kepler only)

Parameters:
dLongitudeDeg – longitude (Dg) of a geosynchronous orbit

Return values: none

void setMagfieldModelDB

(const string& strDb)

Usage: Sets the path of the magnetic field model database for use in coordinate conversions by the Kepler model.

Parameters:
strDb – path and file name of the magnetic field model db (same db used in model).

Return values: none

int computeEphemeris

(vector<double>& vdTimesMJD,
vector<double>& vdXsGEO,
vector<double>& vdYsGEO,
vector<double>& vdZsGEO,
vector<double>& vdXDotsGEO,
vector<double>& vdYDotsGEO,
vector<double>& vdZDotsGEO)

Usage: Compute ephemeris for previously set TLE or mean elements and timeframe

Parameters:
vdTimesMJD – (returned) vector of times at which ephemeris computed
vdXsGEO – (returned) vector of x axis components of positions
vdYsGEO – (returned) vector of y axis components of positions
vdZsGEO – (returned) vector of z axis components of positions
vdXDotsGEO – (returned) vector of x axis components of velocity
vdYDotsGEO – (returned) vector of y axis components of velocity
vdZDotsGEO – (returned) vector of z axis components of velocity

Return values:
int – 0 success, else (see AEErrors.h)

AEAggregator Class

base class (public interface for all aggregator classes)

Header file: AEAggregator.h

Description: This class is the base class for all flux aggregators supplied with Ae9Ap9. Classes derived from this class summarize 3d floating point data in a variety of ways. The base class simply defines a common interface.

Public methods:

AEAggregator

Usage: Default constructor

Parameters: none

Return values: none

virtual ~AEAggregator

Usage: Destructor

Parameters: none

Return values: none

virtual void reset

Usage: clears in-progress aggregated results

Parameters: none

Return values: none

virtual int add

(const dvector& vdDateTimes, const vvdvector& vvvdData)

Usage: inserts new data into an existing aggregation

Parameters:

vdDateTimes – vector of date/times (in MJD) of data to add to the aggregation
vvvdData – 3d vector of data (time,energy,direction) to aggregate

Return values:
0 – success, else error (see AEErrors.h)

virtual void getAggregatedData

(dvector& vdDateTimesMJD,
vvdvector& vvvdData,
bool bIncludeIncompleteInterval = false)

Usage: retrieves completed aggregation intervals, and
optionally, the one currently in progress

Parameters:
vdDateTimesMJD – (return) vector of date/times (in MJD) of aggregated data
vvvdData – (return) 3d vector of aggregated data (time,energy,direction)
bIncludeIncompleteInterval – if true, current period summed even if short
of aggregation interval (usually used at end of orbit)

Return values:
0 – success, else error (see AEErrors.h)

virtual void clearAggregatedData

Usage: empties the collection of completed aggregation intervals

Parameters: none

Return values: none

int getAggregationInterval

Usage: returns the aggregation interval (in # samples)

Parameters: none

Return values:
int – aggregation interval (in # samples)

void setAggregationInterval

(int iNumSamples)

Usage: set the aggregation interval (in # samples)

Parameters:
iNumSamples – aggregation interval (in number of samples)

Return values: none

C Language API

Header file: AECInterface.h

Description: AECInterface.h provides C language wrapper functions to the methods of the Ae9Ap9Application class described above in the C++ API section of this document. The C interface can be used to access the model when linking statically or as a shared object or dll. Note that access to aggregation classes and orbit propagators are not currently supported through the C interface.

Methods:

int AE9AP9APP_setFluxEnvironment_c

(char szModelType,*
char szFluxType,*
char szCoordSys,*
int iNumEnergyLevels,
int iNumEnergyDims,
double pdEnergies,*
int iNumTimes,
const double pdTimes,*
const double pdCoordsAxis1,*
const double pdCoordsAxis2,*
const double pdCoordsAxis3)*

Usage: This method computes flux weights for a grouping of satellite positions at a given set of energies and times when computing omnidirectional flux. It should be called once prior to calling any combination of fly-in flux computation methods for that time period. Note: use as large a grouping of satellite positions as can be processed on available hardware.

Parameters:

szModelType - type of model to be run: Electron, Proton, ModelSpecies (plasma)
szFluxType - type of flux to compute (1 point differential "1PTDIFF", 2 point differential "2PTDIFF", integral "INTEGRAL")
szCoordSys - coordinate system of positions and directions: GEO, GEI, GDZ, GSM, GSE, SSM, MAG, SPH, RLL
iNumEnergyLevels – number of energies passed (per dimension)
iNumEnergyDims – number of energy dimensions (1 unless 2pt. diff flux type)
pdEnergies - array of doubles defining energy levels (MeV) at which to compute flux. Note: column 2 used only for 2 pt differential flux type, which requires computation of flux between two energy levels (row major)
iNumTimes – number of time values passed in pdTimes

pdTimes - array of date/times in modified julian date format at which to compute flux. Lengths of time and position arrays must match, with time corresponding to a position at the same index in the arrays.
pdCoordsAxis1,
pdCoordsAxis2,
pdCoordsAxis3 - Coordinates along axes in 3d space in the eCoordSys coordinate system of each position at which flux is to be computed. These arrays should match the vdTimes array in length and correspond to those times at each position.

Return values:
0 – success, else error (see AEErrors.h)

int AE9AP9APP_setFluxEnvironmentDir_c

(char szModelType,*
char szFluxType,*
char szCoordSys,*
int iNumEnergyLevels,
int iNumEnergyDims,
double pdEnergies,*
int iNumTimes,
const double pdTimes,*
const double pdCoordsAxis1,*
const double pdCoordsAxis2,*
const double pdCoordsAxis3,*
int iNumDirs,
double pdFluxDir1,*
double pdFluxDir2,*
double pdFluxDir3)*

Usage: returns the aggregation interval (in # samples)

Parameters:
szModelType - type of model to be run: Electron, Proton, ModelSpecies (plasma)
szFluxType - type of flux to compute (1 point differential “1PTDIFF”, 2 point differential “2PTDIFF”, integral “INTEGRAL”)
szCoordSys - coordinate system of positions and directions: GEO, GEI, GDZ, GSM, GSE, SSM, MAG, SPH, RLL
iNumEnergyLevels – number of energies passed (per dimension)
iNumEnergyDims – number of energy dimensions (1 unless 2pt. diff flux type)
pdEnergies - array of doubles defining energy levels (MeV) at which to compute flux. Note: column 2 used only for 2 pt differential flux type, which requires computation of flux between two energy levels (row major)
iNumTimes – number of time values passed in pdTimes

pdTimes - array of date/times in modified julian date format at which to compute flux. Lengths of time and position arrays must match, with time corresponding to a position at the same index in the arrays.
pdCoordsAxis1,
pdCoordsAxis2,
pdCoordsAxis3 - Coordinates along axes in 3d space in the eCoordSys coordinate system of each position at which flux is to be computed. These arrays should match the vdTimes array in length and correspond to those times at each position.
iNumDirs – size of the directions dimension in the 2d direction arrays below
pdFluxDir1,
pdFluxDir2,
pdFluxDir3 – Directions at which to compute flux at each timestep. Multiple directions can be computed at each timestep. Thus, the arrays are 2d (time,direction). Defined in szCoordSys coordinates

Return values:
0 – success, else error (see AEErrors.h)

int AE9AP9APP_flyInMean_c

(double pdFluxData)*

Usage: This method computes mean flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Bad values are returned as AE_NaN (defined in AEErrors.h).

Parameters:
pdFluxData –(return) 3D array (time,energy,direction) of flux data (MeV) Sufficient memory should be allocated by caller, as follows: (# times passed to setFluxEnvironment * # Energy levels * # directions [1 for omni]), storage order: row major t1e1d1, t1e1d2, t1e1d3, t1e2d1, t1e2d2, t1e2d3, t2e1d1...

Return Values:
int – 0 success, else (see AEErrors.h)

int AE9AP9APP_flyInPerturbedMean_c

(int iScenario, double pdFluxData)*

Usage: This method computes perturbed mean flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Perturbed means are a statistical distribution of mean flux based on

measurement uncertainty only. Bad values are returned as AE_NaN (defined in AEErrors.h).

Parameters:

iScenario – perturbed mean scenario number (1..999) for repeatability

pdFluxData –(return) 3D array (time,energy,direction) of flux data (MeV) Sufficient memory should be allocated by caller, as follows: (# times passed to setFluxEnvironment * # Energy levels * # directions [1 for omni]), storage order: row major t1e1d1, t1e1d2, t1e1d3, t1e2d1, t1e2d2, t1e2d3, t2e1d1...

Return values:

int – 0 success, else (see AEErrors.h)

int AE9AP9APP_flyInPercentile_c

(int iPercentile, double pdFluxData)*

Usage: This method computes percentile flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Bad values are returned as AE_NaN (defined in AEErrors.h). The function ae9ap9::isnan(value) should be used to test for bad values.

Parameters:

iPercentile – percentile (1..99) flux to compute at each time, energy and position

pdFluxData –(return) 3D array (time,energy,direction) of flux data (MeV) Sufficient memory should be allocated by caller, as follows: (# times passed to setFluxEnvironment * # Energy levels * # directions [1 for omni]), storage order: row major t1e1d1, t1e1d2, t1e1d3, t1e2d1, t1e2d2, t1e2d3, t2e1d1...

Return values:

int – 0 success, else (see AEErrors.h)

int AE9AP9APP_flyInScenario_c

(double dEpoch,
int iScenario,
double pdFluxData,*
bool bPerturbFluxMap = true);

Usage: This method computes monte carlo flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Monte carlo fluxes are a statistical distribution of flux based on both

measurement and temporal uncertainty. Bad values are returned as AE_NaN (defined in AEErrors.h).

Parameters:

dEpoch – start date/time (in Modified Julian Date) for the scenario

iScenario – monte carlo scenario number (1..999) for repeatability

pdFluxData –(return) 3D array (time,energy,direction) of flux data (MeV) Sufficient memory should be allocated by caller, as follows: (# times passed to setFluxEnvironment * # Energy levels * # directions [1 for omni]), storage order: row major t1e1d1, t1e1d2, t1e1d3, t1e2d1, t1e2d2, t1e2d3, t2e1d1...

bPerturbFluxMap – if false, turns off measurement uncertainty mean perturbation. Do Not Use. For testing purposes only.

Return values:

int – 0 success, else (see AEErrors.h)

const char AE9AP9APP_getErrorText_c*

Usage: retrieve text associated with a returned error code

Parameters: none

Return values:

const char* - ptr to a null terminated string of error text

int AE9AP9APP_setModelDataSource_c

(const char szDataSource, int iLength);*

Usage: Set the path and file name of the hdf5 format database containing the data of the selected model. (ie: for high energy electrons, pass "{path}/ AE9V10_runtime_tables.mat") Call this once at startup to initialize the model.

Parameters:

szDataSource – path and file name of the hdf5 database

iLength – length of the path/file name string passed

Return values:

int – 0 success, else (see AEErrors.h)

int AE9AP9APP_setKPhiNeuralNetDataSource_c

(const char szDataSource, int iLength)*

Usage: Set the path and file name of the hdf5 format database containing the K/Phi space neural network used by the selected model. (ie: pass "{path}/ fastPhi_net.mat") Call this once at startup to initialize the model.

Parameters:

szDataSource – path and file name of the hdf5 neural net database

iLength – length of the path/file name string passed

Return values:

int – 0 success, else (see AEErrors.h)

int AE9AP9APP_setKHMinNeuralNetDataSource_c

(const char szDataSource,*
int iLength);

Usage: Set the path and file name of the hdf5 format database containing the K/Hmin space neural network used by the selected model. (ie: pass "{path}/ fast_hmin_net.mat") Call this once at startup to initialize the model.

Parameters:

szDataSource – path and file name of the hdf5 neural net database

iLength – length of the path/file name string passed

Return values:

int – 0 success, else (see AEErrors.h)

int AE9AP9APP_setMagfieldModelDataSource_c

(const char szDataSource, int iLength)*

Usage: Set the path and file name of the hdf5 format database containing the magnetic field model data used by the selected model. (ie: pass "{path}/ igrfDB.h5") . Call this once at startup to initialize the model.

Parameters:

szDataSource – path and file name of the hdf5 database

iLength – length of the path/file name string passed

Return values:
int – 0 success, else (see AEErrors.h)

void AE9AP9APP_cleanup_c

Usage: Call this to ensure the underlying model gets deallocated.
Failure to do so can result in HDF5 console messages on exit.

Parameters: none

Return values: none

const char AE9AP9APP_getVersion_c*

Usage: Call to obtain version number of Ae9Ap9 library in use

Parameters: none

Return values:
const char* – null terminated string containing Ae9Ap9 version

Fortran API

Header file: AEFInterface.h

Description: AEFInterface.h provides Fortran language wrapper functions to the methods of the Ae9Ap9Application class described above in the C++ API section of this document. The Fortran interface can be used to access the model when linking statically or as a shared object or dll. Note that access to aggregation classes and orbit propagators are not currently supported through the Fortran interface.

Methods:

int ae9ap9app_setfluxenvironment_f

(char pchModelType,*
int piModelTypeLen,*
char pchFluxType,*
int piFluxTypeLen,*
char pchCoordSys,*
int piCoordSysLen,*
int piNumEnergies,*
int piNumEnergyDims,*
double pdEnergies,*
int piNumTimes,*
const double pdTimes,*
const double pdCoordsAxis1,*
const double pdCoordsAxis2,*
const double pdCoordsAxis3)*

Usage: This method computes flux weights for a grouping of satellite positions at a given set of energies and times when computing omnidirectional flux. It should be called once prior to calling any combination of fly-in flux computation methods for that time period. Note: use as large a grouping of satellite positions as can be processed on available hardware.

Parameters:

pchModelType - type of model to be run: Electron, Proton, ModelSpecies (plasma)
piModelTypeLen – length of preceding model type character array
pchFluxType - type of flux to compute (1 point differential “1PTDIFF”, 2 point differential “2PTDIFF”, integral “INTEGRAL”)
piFluxTypeLen – length of preceding flux type character array
pchCoordSys - coordinate system of positions and directions: GEO, GEI, GDZ, GSM, GSE, SSM, MAG, SPH, RLL
piCoordSysLen – length of preceding coordinate system character array
piNumEnergyLevels – number of energies passed (per dimension)
piNumEnergyDims – number of energy dimensions (1 unless 2pt. diff flux type)

pdEnergies - array of doubles defining energy levels (MeV) at which to compute flux. Note: column 2 used only for 2 pt differential flux type, which requires computation of flux between two energy levels (row major)

piNumTimes – number of time values passed in pdTimes

pdTimes - array of date/times in modified julian date format at which to compute flux. Lengths of time and position arrays must match, with time corresponding to a position at the same index in the arrays.

pdCoordsAxis1,
pdCoordsAxis2,
pdCoordsAxis3 - Coordinates along axes in 3d space in the eCoordSys coordinate system of each position at which flux is to be computed. These arrays should match the vdTimes array in length and correspond to those times at each position.

Return values:

0 – success, else error (see AEErrors.h)

int ae9ap9app_setfluxenvironmentdir_f

```
( char* pchModelType,
 int* piModelTypeLen,
 char* pchFluxType,
 int* piFluxTypeLen,
 char* pchCoordSys,
 int* piCoordSysLen,
 int* piNumEnergies,
 int* piNumEnergyDims,
 double* pdEnergies,
 int* piNumTimes,
 const double* pdTimes,
 const double* pdCoordsAxis1,
 const double* pdCoordsAxis2,
 const double* pdCoordsAxis3,
 int* piNumFluxDirs,
 double* pdFluxDir1,
 double* pdFluxDir2,
 double* pdFluxDir3 );
```

Usage: This method computes flux weights for a grouping of satellite positions at a given set of energies and times when computing directional flux. It should be called once prior to calling any combination of fly-in flux computation methods for that time period. Note: use as large a grouping of satellite positions as can be processed on available hardware.

Parameters:

pchModelType - type of model to be run: Electron, Proton, ModelSpecies (plasma)
piModelTypeLen – length of preceding model type character array
pchFluxType - type of flux to compute (1 point differential “1PTDIFF”, 2 point differential “2PTDIFF”, integral “INTEGRAL”)
piFluxTypeLen – length of preceding flux type character array
pchCoordSys - coordinate system of positions and directions: GEO, GEI, GDZ, GSM, GSE, SSM, MAG, SPH, RLL
piCoordSysLen – length of preceding coordinate system character array
piNumEnergyLevels – number of energies passed (per dimension)
piNumEnergyDims – number of energy dimensions (1 unless 2pt. diff flux type)
pdEnergies - array of doubles defining energy levels (MeV) at which to compute flux. Note: column 2 used only for 2 pt differential flux type, which requires computation of flux between two energy levels (row major)
piNumTimes – number of time values passed in pdTimes
pdTimes - array of date/times in modified julian date format at which to compute flux. Lengths of time and position arrays must match, with time corresponding to a position at the same index in the arrays.
pdCoordsAxis1,
pdCoordsAxis2,
pdCoordsAxis3 - Coordinates along axes in 3d space in the eCoordSys coordinate system of each position at which flux is to be computed. These arrays should match the vdTimes array in length and correspond to those times at each position.
piNumDirs – size of the directions dimension in the 2d direction arrays below
pdFluxDir1,
pdFluxDir2,
pdFluxDir3 – Directions at which to compute flux at each timestep. Multiple directions can be computed at each timestep. Thus, the arrays are 2d (time,direction). Defined in szCoordSys coordinates

int ae9ap9app_flyinmean_f

(double pppdFluxData);*

Usage: This method computes mean flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Bad values are returned as AE_NaN (defined in AEErrors.h).

Parameters:

pppdFluxData –(return) 3D array (time,energy,direction) of flux data (MeV) Sufficient memory should be allocated by caller, as follows: (# times passed to setFluxEnvironment * # Energy levels * # directions [1 for omni]), storage order: row major t1e1d1, t1e1d2, t1e1d3, t1e2d1, t1e2d2, t1e2d3, t2e1d1...

Return Values:
int – 0 success, else (see AEErrors.h)

int ae9ap9app_flyinperturbedmean_f

(int piScenario, double* pppdFluxData);*

Usage: This method computes perturbed mean flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Perturbed means are a statistical distribution of mean flux based on measurement uncertainty only. Bad values are returned as AE_NaN (defined in AEErrors.h).

Parameters:
piScenario – perturbed mean scenario number (1..999) for repeatability
pppdFluxData –(return) 3D array (time,energy,direction) of flux data (MeV)
Sufficient memory should be allocated by caller, as follows:
(# times passed to setFluxEnvironment * # Energy levels *
directions [1 for omni]), storage order: row major
t1e1d1, t1e1d2, t1e1d3, t1e2d1, t1e2d2, t1e2d3, t2e1d1...

Return values:
int – 0 success, else (see AEErrors.h)

int ae9ap9app_flyinpercentile_f

(int piPercentile, double* pppdFluxData);*

Usage: This method computes percentile flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Bad values are returned as AE_NaN (defined in AEErrors.h). The function ae9ap9::isnan(value) should be used to test for bad values.

Parameters:
piPercentile – percentile (1..99) flux to compute at each time, energy and position
pppdFluxData –(return) 3D array (time,energy,direction) of flux data (MeV)
Sufficient memory should be allocated by caller, as follows:
(# times passed to setFluxEnvironment * # Energy levels *
directions [1 for omni]), storage order: row major
t1e1d1, t1e1d2, t1e1d3, t1e2d1, t1e2d2, t1e2d3, t2e1d1...

Return values:
int – 0 success, else (see AEErrors.h)

int ae9ap9app_flyinscenario_f

(double pdEpoch,*
int piScenario,*
double pppdFluxData,*
bool pbPerturbFluxMap = NULL);*

Usage: This method computes monte carlo flux at each position, time, energy and optionally direction passed in the most recent call to setFluxEnvironment. Monte carlo fluxes are a statistical distribution of flux based on both measurement and temporal uncertainty. Bad values are returned as AE_NaN (defined in AEErrors.h).

Parameters:

pdEpoch – start date/time (in Modified Julian Date) for the scenario
piScenario – monte carlo scenario number (1..999) for repeatability
pppdFluxData –(return) 3D array (time,energy,direction) of flux data (MeV)
Sufficient memory should be allocated by caller, as follows:
(# times passed to setFluxEnvironment * # Energy levels *
directions [1 for omni]), storage order: row major
t1e1d1, t1e1d2, t1e1d3, t1e2d1, t1e2d2, t1e2d3, t2e1d1...
pbPerturbFluxMap – if false, turns off measurement uncertainty mean perturbation. Do Not Use. For testing purposes only.

Return values:

int – 0 success, else (see AEErrors.h)

int ae9ap9app_geterrortext_f

(char pchErrorText, int* piLength);*

Usage: retrieve text associated with a returned error code

Parameters:

pchErrorText – buffer to hold error text
piLength – max length of text (buffer size)

Return values:

int – 0 success, else (see AEErrors.h)

int ae9ap9app_setmodeldatasource_f

(const char pchDataSource, int* piLength);*

Usage: Set the path and file name of the hdf5 format database containing the data of the selected model. (ie: for high energy electrons, pass

"{path}/ AE9V10_runtime_tables.mat") Call this once at startup to initialize the model.

Parameters:
pchDataSource – path and file name of the hdf5 database
piLength – length of the path/file name string passed

Return values:
int – 0 success, else (see AEErrors.h)

int ae9ap9app_setkphineuralnetdatasource_f

(const char pchDataSource,*
int* piLength);

Usage: Set the path and file name of the hdf5 format database containing the K/Phi space neural network used by the selected model. (ie: pass "{path}/ fastPhi_net.mat") Call this once at startup to initialize the model.

Parameters:
pchDataSource – path and file name of the hdf5 neural net database
piLength – length of the path/file name string passed

Return values:
int – 0 success, else (see AEErrors.h)

int ae9ap9app_setkhminneuralnetdatasource_f

(const char pchDataSource,*
int* piLength);

Usage: Set the path and file name of the hdf5 format database containing the K/Hmin space neural network used by the selected model. (ie: pass "{path}/ fast_hmin_net.mat") Call this once at startup to initialize the model.

Parameters:
pchDataSource – path and file name of the hdf5 neural net database
piLength – length of the path/file name string passed

Return values:
int – 0 success, else (see AEErrors.h)

int ae9ap9app_setmagfieldmodeldatasource_f

(const char pchDataSource,*
int* piLength);

Usage: Set the path and file name of the hdf5 format database containing the magnetic field model data used by the selected model. (ie: pass "{path}/ igrfDB.h5") . Call this once at startup to initialize the model.

Parameters:
pchDataSource – path and file name of the hdf5 database
piLength – length of the path/file name string passed

Return values:
int – 0 success, else (see AEErrors.h)

void ae9ap9app_cleanup_f

Usage: Call this to ensure the underlying model gets deallocated. Failure to do so can result in HDF5 console messages on exit.

Parameters: none

Return values: none

int ae9ap9app_getversion_f

(char pchVersionText, int* piLength);*

Usage: Call to obtain version number of Ae9Ap9 library in use

Parameters:
pchVersionText – buffer to hold version number text
piLength – size of buffer

Return values:
int – 0 success, else (see AEErrors.h)

DISTRIBUTION LIST

DTIC/OCP
8725 John J. Kingman Rd, Suite 0944
Ft Belvoir, VA 22060-6218 1 cy

AFRL/RVIL
Kirtland AFB, NM 87117-5776 2 cys

Official Record Copy
AFRL/RVBXR/Dr. Michale Starks 1 cy