IRENE: AE9/AP9/SPM Radiation Environment Model

C 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 C Application Programming Interface
Version 1.58.001

Table of Contents

OVERVIEW **15**

DemoApp and DemoModel Programs 15

APPLICATION-LEVEL C API REFERENCE **17**

Application Class 17

General: *17*

HANDLE AppStartUp 17

int AppShutDown 17

int AppSetExecDir 17

int AppSetWorkDir 18

int AppSetBinDirName 18

int AppSetDelBinDir 18

int AppSetNumProc 18

int AppSetNumFileIo 19

int AppSetWindowsMpiMode 19

int AppSetChunkSize 19

int AppSetTaskDelay 20

int AppGetExecDir 20

int AppGetWorkDir 20

int AppGetBinDirName 21

int AppGetDelBinDir 21

int AppGetNumProc 21

int AppGetNumFileIo 21

int AppGetWindowsMpiMode 21

int AppGetChunkSize 22

int AppGetTaskDelay 22

External Ephemeris Specification: *22*

int AppSetInCoordSys 22

int AppSetInEphemeris 22

int AppClearInEphemeris 23

int AppGetInCoordSys 23

int AppGetInCoordSysUnits 23

int AppGetNumInEphemeris 24

int AppGetInEphemeris 24

Ephemeris Parameter Inputs: *24*

int AppSetPropagator 24

int AppSetSGP4Param 25

int AppSetKeplerUseJ2 25

int AppGetPropagator 25

int AppGetSGP4Mode 25

int AppGetSGP4Datum 26

int AppGetKeplerUseJ2 26

int AppSetTimes 26

int AppGetTimes 26
int AppSetVarTimes....27
int AppGetVarTimes....27
int AppSetTimesList....27
int AppGetNumTimesList....28
int AppGetTimesList....28
int AppClearTimesList....28
int AppSetTLEFile 28
int AppClearTLEFile 29
int AppGetTLEFile....29
int AppSetElementTime....29
int AppSetInclination....29
int AppSetRightAscension....29
int AppSetEccentricity 30
int AppSetArgOfPerigee....30
int AppSetMeanAnomaly....30
int AppSetMeanMotion....30
int AppSetMeanMotion1stDeriv 30
int AppSetMeanMotion2ndDeriv....31
int AppSetBStar....31
int AppSetAltitudeOfApogee 31
int AppSetAltitudeOfPerigee....31
int AppSetLocalTimeOfApogee 31
int AppSetLocalTimeMaxInclination....32
int AppSetTimeOfPerigee 32
int AppSetSemiMajorAxis 32
int AppSetGeosynchLon 32
int AppSetStateVectors 32
int AppSetPositionGEI 33
int AppSetVelocityGEI 33
int AppSetCoordSys....33
int AppGetElementTime 34
int AppGetInclination 34
int AppGetRightAscension 34
int AppGetEccentricity....34
int AppGetArgOfPerigee....35
int AppGetMeanAnomaly....35
int AppGetMeanMotion 35
int AppGetMeanMotion1stDeriv 35
int AppGetMeanMotion2ndDeriv 36
int AppGetBStar 36
int AppGetAltitudeOfApogee....36
int AppGetAltitudeOfPerigee....36
int AppGetLocalTimeOfApogee....37
int AppGetLocalTimeMaxInclination 37
int AppGetTimeOfPerigee....37
int AppGetSemiMajorAxis 37
int AppGetGeosynchLon....38
int AppGetStateVectors....38
int AppGetPositionGEI....38
int AppGetVelocityGEI....38

int AppGetCoordSys 39
int AppGetCoordSysUnits 39
Model Parameter Inputs: *40*
int AppSetModel 40
int AppSetModelDBDir 40
int AppSetModelDBFile 40
int AppSetKPhiDBFile 40
int AppSetKHMinDBFile 41
int AppSetMagfieldDBFile 41
int AppSetDoseModelDBFile 41
int AppSetAdiabatic 41
int AppSetFluxType 42
int AppSetFluxEnergies 42
int AppSetFluxEnergy 42
int AppClearFluxEnergies 42
int AppSetFluxEnergies2 43
int AppSetFluxEnergy2 43
int AppClearFluxEnergies2 43
int AppSetPitchAngle 43
int AppSetPitchAngles 43
int AppClearPitchAngles 44
int AppSetFluxMean 44
int AppSetFluxPercentile 44
int AppSetFluxPercentiles 44
int AppClearFluxPercentiles 45
int AppSetFluxPerturbedScenario 45
int AppSetFluxPerturbedScenarios 45
int AppSetFluxPerturbedScenRange 45
int AppClearFluxPerturbedScenarios 46
int AppSetFluxMonteCarloScenario 46
int AppSetFluxMonteCarloScenarios 46
int AppSetFluxMonteCarloScenRange 46
int AppClearFluxMonteCarloScenarios 47
int AppSetMonteCarloEpochTime 47
int AppSetMonteCarloFluxPerturb 47
int AppSetMonteCarloWorstCase 47
int AppGetModel 48
int AppGetModelDBDir 48
int AppGetModelDBFile 48
int AppGetKPhiDBFile 48
int AppGetKHMinDBFile 49
int AppGetMagfieldDBFile 49
int AppGetDoseModelDBFile 49
int AppGetAdiabatic 50
int AppGetFluxType 50
int AppGetNumFluxEnergies 50
int AppGetFluxEnergies 50
int AppGetNumFluxEnergies2 50
int AppGetFluxEnergies2 51
int AppGetNumPitchAngles 51
int AppGetPitchAngles 51

int AppGetFluxMean 51
int AppGetNumFluxPercentiles 51
int AppGetFluxPercentiles 52
int AppGetNumFluxPerturbedScenarios 52
int AppGetFluxPerturbedScenarios 52
int AppGetNumFluxMonteCarloScenarios 52
int AppGetFluxMonteCarloScenarios 52
int AppGetMonteCarloEpochTime 53
int AppGetMonteCarloFluxPerturb 53
int AppGetMonteCarloWorstCase 53
int AppSetAccumMode 53
int AppClearAccumModes 54
int AppSetAccumInterval 54
int AppSetAccumIntervalSec 54
int AppClearAccumIntervals 54
int AppSetAccumIncrementSec 55
int AppSetAccumIncrementFrac 55
int AppSetReportTimes 55
int AppSetReportTimesSec 55
int AppSetReportAtTime 56
int AppClearReportTimes 56
int AppClearReportAtTime 56
int AppSetFluence 56
int AppGetNumAccumModes 57
int AppGetAccumMode 57
int AppGetAccumModeEntry 57
int AppGetNumAccumIntervals 57
int AppGetAccumIntervalSec 57
int AppGetAccumIntervalSecEntry 58
int AppGetAccumIncrementSec 58
int AppGetAccumIncrementFrac 58
int AppGetNumReportTimes 59
int AppGetReportTimesSec 59
int AppGetNumReportAtTime 59
int AppGetReportAtTime 59
int AppGetFluence 60
int AppSetDoseRate 60
int AppSetDoseAccum 60
int AppSetDoseDepthValues 60
int AppSetDoseDepthUnits 61
int AppSetDoseDepths 61
int AppSetDoseDetector 61
int AppSetDoseGeometry 61
int AppSetDoseNuclearAttenMode 62
int AppSetDoseWithBrems 62
int AppSetUseDoseKernel 62
int AppSetDoseKernelDir 62
int AppSetDoseKernelFile 62
int AppGetDoseRate 63
int AppGetDoseAccum 63
int AppGetNumDoseDepthValues 63

int AppGetDoseDepthValues 63
int AppGetDoseDepthUnits 63
int AppGetDoseDetector 64
int AppGetDoseGeometry 64
int AppGetDoseNuclearAttenMode 64
int AppGetDoseWithBrems 64
int AppGetUseDoseKernel 65
int AppGetDoseKernelDir 65
int AppGetDoseKernelFile 65
int AppSetAggregMedian 65
int AppSetAggregConfLevel 66
int AppSetAggregConfLevels 66
int AppClearAggregConfLevels 66
int AppSetAggregMean 66
int AppGetNumAggregConfLevels 66
int AppGetAggregConfLevels 67
Legacy Model Parameter Inputs: *68*
int AppSetLegActivityLevel 68
int AppSetLegActivityRange 68
int AppSetLeg15DayAvgAp 68
int AppSetLegFixedEpoch 68
int AppSetLegShiftSAA 69
int AppGetLegActivityLevel 69
int AppGetLegActivityRange 69
int AppGetLeg15DayAvgAp 69
int AppGetLegFixedEpoch 70
int AppGetLegShiftSAA 70
int AppSetCamMagfieldModel 70
int AppSetCamDataFilter 70
int AppSetCamPitchAngleBin 70
int AppSetCamSpecies 71
int AppGetCamMagfieldModel 71
int AppGetCamDataFilter 71
int AppGetCamPitchAngleBin 71
int AppGetCamSpecies 72
Model Execution and Results: *73*
int AppRunModel 73
int AppGetEphemeris 74
int AppGetNumDir 74
int AppFlyinMean2D 74
int AppFlyinMean 75
int AppFlyinMeanPlus 75
int AppFlyinPercentile 76
int AppFlyinPercentilePlus 77
int AppFlyinPerturbedMean 78
int AppFlyinPerturbedMeanPlus 78
int AppFlyinMonteCarlo 79
int AppFlyinMonteCarloPlus 80
int AppGetModelData 81
int AppGetAggregData 82
int AppGetAdiabaticCoords 83

int AppGetAdiabaticCoordsPlus 84
int AppResetModelData 85
int AppResetModelRun 85
int AppValidateParameters 86
int AppResetOrbitParameters 86
int AppResetParameters 86
Time Conversion Utilities: *87*
double AppGetGmtSeconds 87
int AppGetDayOfYear 87
double AppGetModifiedJulianDate 87
double AppGetModifiedJulianDateUnix 87
int AppGetDateTime 88
int AppGetHoursMinSec 88
int AppGetMonthDay 88

MODEL-LEVEL C API REFERENCE 91

EPHEMMODEL CLASS 91
General: *91*
HANDLE EphemStartUp 91
EphemShutDown 91
int EphemSetChunkSize 92
int EphemGetChunkSize 92
Model Parameter Inputs: *92*
int EphemSetModelDBDir 92
int EphemSetMagfieldDBFile 93
int EphemSetPropagator 93
int EphemSetSGP4Param 93
int EphemSetKeplerUseJ2 93
int EphemGetModelDBDir 93
int EphemGetMagfieldDBFile 94
int EphemGetPropagator 94
int EphemGetSGP4Mode 94
int EphemGetSGP4WGS 95
int EphemGetKeplerUseJ2 95
int EphemSetTimes 95
int EphemSetVarTimes 95
int EphemSetStartTime 96
int EphemSetEndTime 96
int EphemSetTimeStep 96
int EphemSetVarTimeStep 96
int EphemSetTimesList 97
int EphemGetTimes 97
int EphemGetVarTimes 97
int EphemGetStartTime 98
int EphemGetEndTime 98
int EphemGetTimeStep 98
int EphemGetVarTimeStep 98
int EphemGetNumTimesList 99
int EphemGetTimesList 99
int EphemClearTimesList 99

int EphemSetTLEFile....99
int EphemGetTLEFile....100
int EphemSetTLE....100
int EphemGetNumTLE....100
int EphemGetTLE....100
int EphemResetTLE....101
int EphemSetElementTime....101
int EphemSetInclination....101
int EphemSetRightAscension....101
int EphemSetEccentricity....102
int EphemSetArgOfPerigee....102
int EphemSetMeanAnomaly....102
int EphemSetMeanMotion....102
int EphemSetMeanMotion1stDeriv....102
int EphemSetMeanMotion2ndDeriv....103
int EphemSetBStar....103
int EphemSetAltitudeOfApogee....103
int EphemSetAltitudeOfPerigee....103
int EphemSetLocalTimeOfApogee....103
int EphemSetLocalTimeMaxInclination....104
int EphemSetTimeOfPerigee....104
int EphemSetSemiMajorAxis....104
int EphemSetGeosynchLon....104
int EphemSetStateVectors....104
int EphemSetPositionGEI....105
int EphemSetVelocityGEI....105
int EphemGetElementTime....105
int EphemGetInclination....106
int EphemGetRightAscension....106
int EphemGetEccentricity....106
int EphemGetArgOfPerigee....106
int EphemGetMeanAnomaly....107
int EphemGetMeanMotion....107
Int EphemGetMeanMotion1stDeriv....107
int EphemGetMeanMotion2ndDeriv....107
int EphemGetBStar....108
int EphemGetAltitudeOfApogee....108
int EphemGetAltitudeOfPerigee....108
int EphemGetLocalTimeOfApogee....108
int EphemGetLocalTimeMaxInclination....109
int EphemGetTimeOfPerigee....109
int EphemGetSemiMajorAxis....109
int EphemGetGeosynchLon....109
int EphemGetStateVectors....110
int EphemGetPositionGEI....110
int EphemGetVelocityGEI....110
int EphemResetOrbitParameters....110
int EphemSetMainField....111
int EphemSetExternalField....111
int EphemGetMainField....111
int EphemGetExternalField....111

int EphemSetKpValue 112
int EphemSetKpValues 112
int EphemGetKpValue 112
int EphemGetKpValuesRefTime 113
int EphemGetKpValuesEndTime 113
Model Execution and Results: *113*
int EphemComputeEphemerisGEI 113
int EphemComputeEphemeris 114
int EphemRestartEphemeris 115
int EphemConvertCoordinates 115
int EphemConvertCoordinatesSingle 116
AE9AP9MODEL CLASS 121
General: *121*
HANDLE Ae9Ap9StartUp 121
Ae9Ap9ShutDown 121
Model Parameter Inputs: *121*
int Ae9Ap9SetModel 121
int Ae9Ap9SetModelDBDir 121
int Ae9Ap9SetModelDBFile 122
int Ae9Ap9SetKPhiDBFile 122
int Ae9Ap9SetKHMinDBFile 122
int Ae9Ap9SetMagfieldDBFile 122
int Ae9Ap9LoadModelDB 123
int Ae9Ap9GetModelName 123
int Ae9Ap9GetModelSpecies 123
int Ae9Ap9GetModel 124
int Ae9Ap9GetModelDBDir 124
int Ae9Ap9GetModelDBFile 124
int Ae9Ap9GetKPhiDBFile 124
int Ae9Ap9GetKHMinDBFile 125
int Ae9Ap9GetMagfieldDBFile 125
Model Execution and Results: *125*
int Ae9Ap9SetFluxEnvironmentOmni 126
int Ae9Ap9SetFluxEnvironmentFixPitch 127
int Ae9Ap9SetFluxEnvironmentVarPitch 127
int Ae9Ap9SetFluxEnvironmentDirVec 128
int Ae9Ap9GetPitchAngles 129
int Ae9Ap9FlyinMean or Ae9Ap9ComputeFluxMean 129
int Ae9Ap9FlyinMean 130
int Ae9Ap9FlyinPercentile or Ae9Ap9ComputeFluxPercentile 130
int Ae9Ap9FlyinPerturbedMean or Ae9Ap9ComputeFluxPerturbedMean 130
int Ae9Ap9FlyinScenario or Ae9Ap9ComputeFluxScenario 131
ACCUMMODEL CLASS 133
General: *133*
HANDLE AccumStartUp 133
AccumShutDown 133
Model Parameter Inputs: *133*
int AccumSetTimeInterval 133
int AccumSetTimeIntervalSec 133
int AccumSetTimeIncrement 134

int AccumGetTimeInterval....134
int AccumGetTimeIncrement....134
Model Execution and Results:....135
int AccumLoadBuffer....135
int AccumAddToBuffer....135
int AccumClearBuffer....135
int AccumGetBufferDim....135
int AccumComputeFluence....136
int AccumComputeIntvFluence....136
int AccumAccumIntvFluence....137
int AccumComputeFullFluence....137
int AccumComputeBoxcarFluence....138
int AccumComputeAverageFlux....139
int AccumComputeExponentialFlux....139
int AccumApplyWorstToDate....140
int AccumResetFluence....141
int AccumResetIntvFluence....141
int AccumResetFullFluence....141
int AccumResetBoxcarFluence....141
int AccumResetExponentialFlux....141
int AccumGetFluenceStartTime....141
int AccumGetIntvFluenceStartTime....142
int AccumGetFullFluenceStartTime....142
int AccumGetBoxcarFluenceStartTime....142
int AccumGetLastLength....142
DOSEMODEL CLASS....145
General:....145
HANDLE DoseStartUp....145
DoseShutDown....145
Model Parameter Inputs:....145
int DoseSetModelDBDir....145
int DoseSetModelDBFile....145
int DoseSetSpecies....146
int DoseSetEnergies....146
int DoseSetDepths....146
int DoseSetDetector....146
int DoseSetGeometry....147
int DoseSetNuclearAttenMode....147
int DoseSetWithBrems....147
int DoseGetModelDBDir....147
int DoseGetModelDBFile....147
int DoseGetSpecies....148
int DoseGetNumEnergies....148
int DoseGetNumDepths....148
int DoseGetDetector....148
int DoseGetGeometry....149
int DoseGetNuclearAttenMode....149
int DoseGetWithBrems....149
Model Execution and Results:....149
int DoseComputeFluxDose....149

int DoseComputeFluxDoseRate 150
int DoseComputeFluenceDose 150
DOSEKERNEL CLASS 153
General: *153*
HANDLE DoseKStartUp 153
DoseKShutDown 153
Model Parameter Inputs: *153*
int DoseKSetKernelXmlPath 153
int DoseKSetKernelXmlFile 153
int DoseKSetSpecies 154
int DoseKSetEnergies 154
int DoseKSetDepths 154
int DoseKSetDetector 154
int DoseKSetGeometry 155
int DoseKSetNuclearAttenMode 155
int DoseKSetWithBrems 155
int DoseKGetKernelXmlPath 155
int DoseKGetKernelXmlFile 155
int DoseKGetSpecies 156
int DoseKGetNumEnergies 156
int DoseKGetNumDepths 156
int DoseKGetDetector 156
int DoseKGetGeometry 157
int DoseKGetNuclearAttenMode 157
int DoseKGetWithBrems 157
Model Execution and Results: *157*
int DoseKComputeFluxDose 157
int DoseKComputeFluxDoseRate 158
int DoseKComputeFluenceDose 158
AGGREGMODEL CLASS 161
General: *161*
HANDLE AggregStartUp 161
AggregShutDown 161
Model Parameter Inputs: *161*
int AggregResetAgg 161
int AggregAddScenToAgg 161
int AggregGetAggDimensions 162
int AggregGetNumScenarios 162
Model Execution and Results: *162*
int AggregComputeConfLevel 162
int AggregComputeMedian 163
int AggregComputeMean 163
ADIABATMODEL CLASS 165
General: *165*
HANDLE AdiabatStartUp 165
AdiabatShutDown 165
Model Parameter Inputs: *165*
int AdiabatSetModelDBDir 165
int AdiabatSetKPhiDBFile 165
int AdiabatSetKHMinDBFile 166

int AdiabatSetMagfieldDBFile 166
int AdiabatGetModelDBDir 166
int AdiabatGetKPhiDBFile 167
int AdiabatGetKHMinDBFile 167
int AdiabatGetMagfieldDBFile 167
int AdiabatSetK_Min 168
int AdiabatSetK_Max 168
int AdiabatSetHminMin 168
int AdiabatSetHminMax 168
int AdiabatSetPhiMin 168
int AdiabatSetPhiMax 169
int AdiabatUpdateLimits 169
int AdiabatGetK_Min 169
int AdiabatGetK_Max 169
int AdiabatGetHminMin 169
int AdiabatGetHminMax 170
int AdiabatGetPhiMin 170
int AdiabatGetPhiMax 170
Model Execution and Results: *170*
int AdiabatComputeCoordinateSet 170
int AdiabatComputeCoordinateSetVarPitch 172
int AdiabatCalcDirPitchAngles 173
int AdiabatConvertCoordinates 174
int AdiabatConvertCoordinatesSingle 175
RADENVMODEL CLASS 177
General: *177*
HANDLE RadEnvStartUp 177
RadEnvShutDown 177
Model Parameter Inputs: *177*
int RadEnvSetModel 177
int RadEnvGetModel 177
int RadEnvSetModelDBDir 177
int RadEnvSetModelDBFile 178
int RadEnvSetMagfieldDBFile 178
int RadEnvGetModelDBDir 178
int RadEnvGetModelDBFile 179
int RadEnvGetMagfieldDBFile 179
int RadEnvSetFluxType 179
int RadEnvSetEnergies 179
int RadEnvSetActivityLevel 180
int RadEnvSetActivityRange 180
int RadEnvSet15DayAvgAp 180
int RadEnvSetFixedEpoch 180
int RadEnvSetShiftSAA 181
int RadEnvGetFluxType 181
int RadEnvGetNumEnergies 181
int RadEnvGetActivityLevel 181
int RadEnvGetActivityRange 182
int RadEnvGet15DayAvgAp 182
int RadEnvGetFixedEpoch 182

int RadEnvGetShiftSAA 182
int RadEnvSetCoordSys 182
int RadEnvSetEphemeris 183
int RadEnvGetCoordSys 183
int RadEnvGetCoordSysUnits 183
int RadEnvGetNumEphemeris 184
Model Execution and Results: *184*
int RadEnvComputeFlux 184
CAMMICEMODEL CLASS 185
General: *185*
HANDLE CammiceStartUp 185
CammiceShutDown 185
Model Parameter Inputs: *185*
int CammiceSetModelDBDir 185
int CammiceSetModelDBFile 185
int CammiceSetMagfieldDBFile 186
int CammiceSetMagfieldModel 186
int CammiceSetDataFilter 186
int CammiceSetPitchAngleBin 186
int CammiceSetSpecies 187
int CammiceSetCoordSys 187
int CammiceGetModelDBDir 187
int CammiceGetModelDBFile 187
int CammiceGetMagfieldDBFile 188
int CammiceGetMagfieldModel 188
int CammiceGetDataFilter 188
int CammiceGetPitchAngleBin 188
int CammiceGetSpecies 189
int CammiceGetCoordSys 189
int CammiceGetCoordSysUnits 189
int CammiceSetEphemeris 189
int CammiceGetNumEphemeris 190
Model Execution and Results: *190*
int CammiceComputeFlux 190
DATETIMEUTIL CLASS 191
Model Execution and Results: *191*
double DateTimeGetGmtSeconds 191
int DateTimeGetDayOfYear 191
double DateTimeGetModifiedJulianDate 191
double DateTimeGetModifiedJulianDateUnix 191
int DateTimeGetDateTime 192
int DateTimeGetHoursMinSec 192
int DateTimeGetMonthDay 192

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 language. 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.

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.

User applications in C that utilize the IRENE package API may be built using CMake, as was the various executable and library files of the IRENE package. Please refer to the "Build Instructions" document included in this distribution for the configuration settings used in the CMake build process, and the use of third-party libraries: Boost® (for linear algebra functions and data structures), HDF5® (for internal databases), and Xerces-C++ (XML parser). Windows builds will require the use of Visual Studios 2017. The Intel MPI Library development product will be required for building of multi-threaded 64-bit Windows applications.

More information about the CMake build system can be found at https://cmake.org/documentation/ .

The C version of *DemoApp* application can be built using CMake using the provided configuration files. Make a build directory (such as '~/Irene/<*platform*>/demoApp') [here '~' refers to the path to the 'Irene' installation location, and <*platform*> may be 'linux' or 'win64']. Navigate to that directory, then enter the command:

```
cmake ~/Irene/api_demos/demoApp/C -DIRENE_ROOT=~/Irene/<platform>
```

The optional '`-DCMAKE_INSTALL_PREFIX=<path>`' can be used to specify a location for the 'install' step, if desired. The *DemoAppC* executable will be copied to the "<*path*>/bin" directory. The multi-threaded version (*DemoAppMpiC*) can be also be built by adding '`-DUSE_MPI=YES`' to the *cmake* command.
Then use the commands '`make`' and '`make install`' on Linux. On Windows, use the Visual Studios 'solution' file for the 'build' and 'install' steps, as described in the "Build Instructions" document.

This application performs a variety of model calculations and associated operations. Enter:

```
DemoAppC -x ~/Irene/<platform>/bin -d ~/Irene/modelData
```

Other application argument options are available; enter '`DemoAppC -h`' for the full list.
A 'launcher' may be required for executing the multi-threaded *DemoAppMpi* application:
Linux:

```
mpirun -np 1 DemoAppMpiC -x ~/Irene/<platform>/bin -d ~/Irene/modelData -n 5
```

Windows:

```
mpiexec.exe -np 1 DemoAppMpiC.exe -x ~/Irene/<platform>/bin -d ~/Irene/modelData -n 5
```

The C version of the *DemoModel* application build process is similar. Make a build directory (such as '~/Irene/<*platform*>/demoModel'). Navigate to that directory, then enter the command:

```
cmake ~/Irene/api_demos/demoModel/C -DIRENE_ROOT=~/Irene/<platform>
```

Refer to the "Build Instructions" document for more details.
The optional '`-DCMAKE_INSTALL_PREFIX=<path>`' can be used to specify a location for the 'install' step, if desired. The executable will be copied to the "<*path*>/bin" directory. (Multi-threaded operation is not available in the Model-Level API).
Then use the commands '`make`' and '`make install`' on Linux. On Windows, use the Visual Studios 'solution' file for the 'build' and 'install' steps, as described in the "Build Instructions" document.

This application also performs a variety of model calculations and associated operations. Enter:

```
DemoModelC ~/Irene/modelData
```

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

The specification of a directory path and/or filename may include a system environment variable, ie '`$IRENE_DB/igrfDB.h5`' [Linux style] or '`%IRENE_DB%/igrfDB.h5`' [Windows style], provided the environment variable ('`IRENE_DB`' in this example) has been previously defined.

Application-Level C API Reference

Application Class

Header file: CApplication_c.h

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.

Important: For all routines that return values via pointer arguments, the calling program is responsible for the allocation of the proper amount of memory to contain the returned values. The description of these routines include recommended sizing, or specify other routines to call to determine the required sizing. The calling program is also responsible for the subsequent 'free'ing of the allocated memory.

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:

HANDLE AppStartUp

Usage: Instantiates a new 'Application' object; multiple calls to this will generate separate instances, each of which may be accessed using the unique returned handle value.
Parameters: -none-
Return value: HANDLE - identifier value for the instantiated object

int AppShutDown

(HANDLE zHandle)
Usage: Closes down and removes the 'Application' object specified by the identifer. The object identifier will no longer be valid after this call.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int - 0 = success, otherwise error

The following methods are used for adjusting the internal behavior of the model Apps.
The use of the *AppSetExecDir*() method is <u>required</u>; all others in this section are optional.

int AppSetExecDir

(HANDLE zHandle,
char* szExecDir)

Usage: (***Required***) Specifies the directory path to the executable programs that are needed for performing the prescribed model calculations.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szExecDir - path to installation executables, such as *CmdLineIrene(.exe)*

Return value: int – 0 = success, otherwise error

int AppSetWorkDir

(HANDLE zHandle,
char* szWorkDir)

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 *AppSetNumFileIo*() method may further improve performance.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetBinDirName

(HANDLE zHandle,
char* szBinDirName)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szBinDirName - name of temporary directory to be created

Return value: int – 0 = success, otherwise error

int AppSetDelBinDir

(HANDLE zHandle,
int 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 *AppRunModel*() method. Does not apply to calls to the *AppResetModelRun*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetNumProc

(HANDLE zHandle,
int 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iNumProc – number of processors

Return value: int – 0 = success, otherwise error

int AppSetNumFileIo

(HANDLE zHandle,
int 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iNumFileIo – number of processors

Return value: int – 0 = success, otherwise error

int AppSetWindowsMpiMode

(HANDLE zHandle,
char* szMode)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szMode – 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: int – 0 = success, otherwise error

int AppSetChunkSize

(HANDLE zHandle,
int 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 highly 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 *AppFlyin[]*() and *AppGet[]*() 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetTaskDelay

(HANDLE zHandle,
int 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

---

int AppGetExecDir

(HANDLE zHandle,
char* szExecDir)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szExecDir – pointer to character string (suggested sizing = 256)

Returned parameter:

szExecDir – string containing path to installation executables

Return value: int – 0 = success, otherwise error

int AppGetWorkDir

(HANDLE zHandle,
char* szWorkDir)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szWorkDir – pointer to character string (suggested sizing = 256)
Returned parameter:
szWorkDir – string containing path to location for the temporary directory
Return value: int – 0 = success, otherwise error

int AppGetBinDirName

(HANDLE zHandle,
char* szBinDirName)

Usage: Returns the name for the temporary directory for the intermediate binary files, as specified in the *AppSetBinDirName*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szBinDirName – pointer to character string (suggested sizing = 256)
Returned parameter:
szBinDirName – name for the temporary directory for the intermediate binary files, or 'default'.
Return value: int – 0 = success, otherwise error

int AppGetDelBinDir

(HANDLE zHandle)

Usage: Returns the disposition of the temporary directory and its intermediate binary files, as specified in the *AppSetDelBinDir*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – 0 = false, 1= true

int AppGetNumProc

(HANDLE zHandle)

Usage: Returns the number of processors to use, as specified in the *AppSetNumProc*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – number of processors

int AppGetNumFileIo

(HANDLE zHandle)

Usage: Returns the number of threads to use for the file I/O steps, as specified in the *AppSetNumFileIo*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – number of processors

int AppGetWindowsMpiMode

(HANDLE zHandle,
char* szMode)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szMode – pointer to character string (suggested sizing = 16)
Returned parameter:
szMode – Windows MPI mode string
Return value: int – 0 = success, otherwise error

int AppGetChunkSize

(HANDLE zHandle)
Usage: Returns the current value of the 'chunk' size, as specified in the *AppSetChunkSize*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – number of entries in processing chunk

int AppGetTaskDelay

(HANDLE zHandle)
Usage: Returns the current value of the 'task delay', as specified in the *setTaskDelay*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – 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.

int AppSetInCoordSys

(HANDLE zHandle
char* szCoordSys,
char* szCoordUnits)
Usage: Specifies the coordinate system and units for the position values that are specified by the *AppSetInEphemeris*() method. When not specified, these settings default to 'GEI' and 'Re'. "Re" = radius of the Earth, defined as 6371.2 km.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szCoordSys – 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.
szCoordUnits – coordinate units, 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value
Return value: int – 0 = success, otherwise error

int AppSetInEphemeris

(HANDLE zHandle,
double* pvdTimes,
double* pvdCoords1,

double* pvdCoords2,
double* pvdCoords3,
int iNumTimes,
int iAppend)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

pvdCoords1, *pvdCoords2*, *pvdCoords3* - arrays of position coordinate values, size=*iNumTimes*, associated with times array. These position values are assumed to be in the coordinate system and units specified by *AppSetInCoordSys*().

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.

iNumTimes – number of values in time and coords arrays

iAppend – flag for appending this ephemeris information to any ephemeris information specified in previous call(s) to *AppSetEphemeris*(). If 0 *(false)*, this ephemeris information will replace any existing ephemeris information, if 1 *(true)* it will be added to the end.

Return value: int – 0 = success, otherwise error

int AppClearInEphemeris

(HANDLE zHandle)

Usage: Clears any existing input ephemeris information that was specified in previous calls to *AppSetInEphemeris*().

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

---

int AppGetInCoordSys

(HANDLE zHandle,
char* szCoordSys)

Usage: Returns the coordinate system, specified in the *AppSetInCoordSys*() method for the set of position values as specified in the *AppSetInEphemeris*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szCoordSys – pointer to character string (suggested sizing = 16)

Returned parameter:

szCoordSys – coordinate system string

Return value: int – 0 = success, otherwise error

int AppGetInCoordSysUnits

(HANDLE zHandle,
char* szCoordUnits)

Usage: Returns the coordinate system units, specified in the *AppSetInCoordSys*() method for the set of position values as specified in the *AppSetInEphemeris*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szCoordUnits – pointer to character string (suggested sizing = 16)

Returned parameter:

szCoordUnits – coordinate system units string ('Re' or 'km')

Return value: int – 0 = success, otherwise error

int AppGetNumInEphemeris

(HANDLE zHandle)

Usage: Returns the number of currently defined ephemeris time and position entries.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – number of ephemeris entries, specified in the *AppSetInEphemeris*() method.

int AppGetInEphemeris

(HANDLE zHandle,
double* pvdTimes,
double* pvdCoords1,
double* pvdCoords2,
double* pvdCoords3)

Usage: Returns the input ephemeris time and positions, as specified in the *AppSetInEphemeris*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdTimes – pointer to double array, sized according to *AppGetNumInEphemeris*() results

pvdCoords1, *pvdCoords2*, *pvdCoords3* – pointers to double arrays, sized according to *AppGetNumInEphemeris*() results

Returned Parameters:

pvdTimes – array of time values, in Modified Julian Date form.

pvdCoords1, *pvdCoords2*, *pvdCoords3* – 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 *AppGetInCoordSys*() and *AppGetInCoordSysUnits*() methods.

Returned value: int - Number of records returned (≥0 = success, <0 = error)

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.

int AppSetPropagator

(HANDLE zHandle,
char* szPropSpec)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szPropSpec – propagator model specification; valid values: 'SatEph', 'SGP4' or 'Kepler'.
Return value: int – 0 = success, otherwise error

int AppSetSGP4Param

(HANDLE zHandle,
char* szMode,
char* szWGS)

Usage: Specifies the mode and WGS parameter for the SGP4 orbit propagator, if being used.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szMode – SGP4 propagation mode; valid values: 'Standard' or 'Improved'.
szWGS – World Geodetic System version; valid values: '72old', '72' or '84'.
Return value: int – 0 = success, otherwise error

int AppSetKeplerUseJ2

(HANDLE zHandle,
int iUseJ2)

Usage: Specifies the use of the 'J2' perturbation for the Kepler orbit propagator, if being used.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
iUseJ2 – flag for use of the J2 perturbation feature; *1 (true)* or *0 (false)*
Return value: int – 0 = success, otherwise error

---

int AppGetPropagator

(HANDLE zHandle,
char* szPropSpec)

Usage: Returns the orbit propagator identifier, as specified in the *AppSetPropagator*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szPropSpec – pointer to character string (suggested sizing = 16)
Returned parameter:
szPropSpec – orbit propagator name.
Return value: int – 0 = success, otherwise error

int AppGetSGP4Mode

(HANDLE zHandle
char* szMode)

Usage: Returns the SGP4 propagator mode setting, as specified in the *AppSetSGP4Param*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szMode – pointer to character string (suggested sizing = 16)
Returned parameter:
szMode – SGP4 mode setting string.
Return value: int – 0 = success, otherwise error

int AppGetSGP4Datum

(HANDLE zHandle,
char* szWGS)

Usage: Returns the SGP4 propagator WGS setting, as specified in the *AppSetSGP4Param*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szWGS – pointer to character string (suggested sizing = 16)

Returned parameter:

szWGS – SGP4 WGS setting string.

Return value: int – 0 = success, otherwise error

int AppGetKeplerUseJ2

(HANDLE zHandle)

Usage: Returns the Kepler propagator 'J2' perturbation setting, as specified in the *AppSetKeplerUseJ2*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – *1 (true)* or *0 (false)*.

---

int AppSetTimes

(HANDLE zHandle,
double dStartTime,
double dEndTime,
double dTimeStepSec)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

dTimeStepSec - time step size, in seconds.

Return value: int – 0 = success, otherwise error

int AppGetTimes

(HANDLE zHandle,
double* pdStartTime,
double* pdEndTime,
double* pdTimeStepSec)

Usage: Returns the start and stop times, and fixed time step, for the ephemeris generation.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method.

pdStartTime, *pdEndTime, pdTimeStepSec* – pointers to double variables.

Returned Parameters:

pdStartTime, *pdEndTime* – start and stop time values, in Modified Julian Date form.

pdTimeStepSec – time step size, in seconds.

Return value: int – 0 = success, otherwise error

int AppSetVarTimes

(HANDLE zHandle,
double dStartTime,
double dEndTime,
double dTimeMinStepSec,
double dTimeMaxStepSec,
double dTimeRoundSec)

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 specified 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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: int – 0 = success, otherwise error

int AppGetVarTimes

(HANDLE zHandle,
double* pdStartTime,
double* pdEndTime,
double* pdTimeMinStepSec,
double* pdTimeMaxStepSec,
double* pdTimeRoundSec)

Usage: Returns the start and stop times, and variable time step limits, for the ephemeris generation.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdStartTime, *pdEndTime, pdTimeMinStepSec, pdTimeMaxStepSec, pdTimeRoundSec* – pointers to double variables

Returned Parameters:

pdStartTime, *pdEndTime* – start and stop time values, in Modified Julian Date form

p*dTimeMinStepSec* – lower limit of variable time steps, in seconds

p*dTimeMaxStepSec* – upper limit of variable time steps, in seconds

p*dTimeRoundSec* – rounding of variable time steps, in seconds

Return value: int – 0 = success, otherwise error

int AppSetTimesList

(HANDLE zHandle,
double* pvdTimes,
int iNumTimes)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pvdTimes – array of chronologically ordered time values, in Modified Julian Date form
iNumPitch – number of time values in array
Return value: int – 0 = success, otherwise error

int AppGetNumTimesList

(HANDLE zHandle)

Usage: Returns the number of time entries defined for the ephemeris generation, from the specifications in the *AppSetTimesList*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – number of ephemeris times defined.

int AppGetTimesList

(HANDLE zHandle,
double* pvdTimes)

Usage: Returns the array of time values, in Modified Julian Date form, defined for the ephemeris generation, from the specifications in the *AppSetTimesList*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pvdTimes – pointer to double array, sized according to *AppGetNumTimesList*() results.
Returned Parameters:
pvdTimes – array of time values, in Modified Julian Date form.
Return value: int - Number of values returned (≥0 = success, <0 = error)

int AppClearTimesList

(HANDLE zHandle)

Usage: Clears the time list defined by calls to the *AppSetTimesList*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – 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.

int AppSetTLEFile

(HANDLE zHandle,
char* szTLEFile)

Usage: Specifies the name of the Two-Line Element (TLE) file to use with the selected orbit propagator; this parameter is not needed if a set of orbital element values are being used instead.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szTLEFile – path and filename of TLE file
Return value: int – 0 = success, otherwise error

int AppClearTLEFile

(HANDLE zHandle)
Usage: Unsets the specification of the TLE file.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – 0 = success, otherwise error

int AppGetTLEFile

(HANDLE zHandle,
char* szTLEFile)
Usage: Returns the name of the specified TLE file, as specified in the *AppSetTLEFile*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szTLEFile – pointer to character string (suggested sizing = 256)
Returned parameter:
szTLEFile – path and filename of the TLE file
Return value: int – 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 also requires an associated element time to be specified. See the User's Guide document '*Orbiter Propagation Inputs*' section for more details.

int AppSetElementTime

(HANDLE zHandle,
double dElementTime)
Usage: Specifies the 'epoch' time associated with the set of orbital element values.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dElementTime – element 'epoch' time, in Modified Julian Date form
Return value: int – 0 = success, otherwise error

int AppSetInclination

(HANDLE zHandle,
double dInclination)
Usage: Specifies the orbital element 'Inclination' value.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dInclination – orbit inclination angle, in degrees (0-180)
Return value: int – 0 = success, otherwise error

int AppSetRightAscension

(HANDLE zHandle,
double dRtAscOfAscNode)
Usage: Specifies the orbital element 'Right Ascension of the Ascending Node' value.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method

dRtAscOfAscNode – orbit ascending node position, in degrees (0-360)
Return value: int – 0 = success, otherwise error

int AppSetEccentricity

(HANDLE zHandle,
double dEccentricity)
Usage: Specifies the orbital element 'Eccentricity' value.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dEccentricity – orbit eccentricity value, unitless (0-<1)
Return value: int – 0 = success, otherwise error

int AppSetArgOfPerigee

(HANDLE zHandle,
double dArgOfPerigee)
Usage: Specifies the orbital element 'Argument of Perigee' value.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dArgOfPerigee – orbit perigee position, in degrees (0-360)
Return value: int – 0 = success, otherwise error

int AppSetMeanAnomaly

(HANDLE zHandle,
double dMeanAnomaly)
Usage: Specifies the orbital element 'Mean Anomaly' value.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dMeanAnomaly – orbit mean anomaly value, in degrees (0-360)
Return value: int – 0 = success, otherwise error

int AppSetMeanMotion

(HANDLE zHandle,
double dMeanMotion)
Usage: Specifies the orbital element 'Mean Motion' value.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dMeanMotion – orbit mean motion value, in units of *revolutions per day* (must be >0)
Return value: int – 0 = success, otherwise error

int AppSetMeanMotion1stDeriv

(HANDLE zHandle,
double 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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method

dMeanMotion1stDeriv – first derivative of mean motion, in units of revs per day^2
Return value: int – 0 = success, otherwise error

int AppSetMeanMotion2ndDeriv

(HANDLE zHandle,
double 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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dMeanMotion2ndDeriv – second derivative of mean motion, in units of revs per day^3
Return value: int – 0 = success, otherwise error

int AppSetBStar

(HANDLE zHandle,
double dBStar)
Usage: Specifies the orbital element 'B*' value, for modelling satellite drag effects; this value is only used by the SGP4 propagator.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dBStar – ballistic coefficient value
Return value: int – 0 = success, otherwise error

int AppSetAltitudeOfApogee

(HANDLE zHandle,
double dAltApogee)
Usage: Specifies the orbital element 'Apogee Altitude' value (furthest distance).
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dAltApogee – altitude (in km) above the Earth's surface at the orbit's apogee (>0-74Re)
Return value: int – 0 = success, otherwise error

int AppSetAltitudeOfPerigee

(HANDLE zHandle,
double dAltPerigee)
Usage: Specifies the orbital element 'Perigee Altitude' value (closest distance).
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dAltPerigee – altitude (in km) above the Earth's surface at the orbit's perigee (>0-74Re)
Return value: int – 0 = success, otherwise error

int AppSetLocalTimeOfApogee

(HANDLE zHandle,
double dLocTimeApogee)
Usage: Specifies the local time of the orbit's apogee.
Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method
dLocTimeApogee – local time, in hours (0-24)
Return value: int – 0 = success, otherwise error

int AppSetLocalTimeMaxInclination

(HANDLE zHandle,
double dLocTimeMaxIncl)
Usage: Specifies the local time of the orbit's maximum inclination (ie max latitude).
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dLocTimeMaxIncl – local time, in hours (0-24)
Return value: int – 0 = success, otherwise error

int AppSetTimeOfPerigee

(HANDLE zHandle,
double 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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dTimeOfPerigee – time, in Modified Julian Date form, for orbit perigee
Return value: int – 0 = success, otherwise error

int AppSetSemiMajorAxis

(HANDLE zHandle,
double dSemiMajorAxis)
Usage: Specifies the orbit's semi-major axis length.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dSemiMajorAxis – semi-major axis length (1-75), in units of Re (radius of Earth = 6371.2 km)
Return value: int – 0 = success, otherwise error

int AppSetGeosynchLon

(HANDLE zHandle,
double dGeosynchLon)
Usage: Specifies the geographic East longitude of satellite in a geosynchronous orbit.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dGeosynchLon – East longitude, in degrees (-180 – 360)
Return value: int – 0 = success, otherwise error

int AppSetStateVectors

(HANDLE zHandle,
double* pvdPosition,
double* pvdVelocity)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

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

Return value: int – 0 = success, otherwise error

int AppSetPositionGEI

(HANDLE zHandle,
double dPosX,
double dPosY,
double 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 *AppSetVelocityGEI*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetVelocityGEI

(HANDLE zHandle,
double dVelX,
double dVelY,
double 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 *AppSetPositionGEI*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetCoordSys

(HANDLE zHandle,
char* szCoordSys,
char* szCoordUnits)

Usage: Specifies the coordinate system and units for the position values that will be generated by the propagator specified by *AppSetPropagator*(). These default to 'GEI' and 'Re' when not specified; if the *AppSetInCoordSys*() 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szCoordSys – 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.

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

Return value: int – 0 = success, otherwise error

int AppGetElementTime

(HANDLE zHandle,
double* pdElementTime)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdElementTime – pointer to double variable

Returned parameter:

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

Return value: int – 0 = success, otherwise error

int AppGetInclination

(HANDLE zHandle,
double* pdInclination)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdInclination – pointer to double variable

Returned parameter:

pdInclination – orbit inclination angle, in degrees.

Return value: int – 0 = success, otherwise error

int AppGetRightAscension

(HANDLE zHandle,
double* pdRtAscOfAscNode)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdRtAscOfAscNode – pointer to double variable

Returned parameter:

pdRtAscOfAscNode – orbit ascending node position, in degrees.

Return value: int – 0 = success, otherwise error

int AppGetEccentricity

(HANDLE zHandle,
double* pdEccentricity)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdEccentricity – pointer to double variable

Returned parameter:

pdEccentricity – orbit eccentricity value (unitless).

Return value: int – 0 = success, otherwise error

int AppGetArgOfPerigee

(HANDLE zHandle,
double* pdArgOfPerigee)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdArgOfPerigee – pointer to double variable

Returned parameter:

pdArgOfPerigee – orbit perigee position, in degrees.

Return value: int – 0 = success, otherwise error

int AppGetMeanAnomaly

(HANDLE zHandle,
double* pdMeanAnomaly)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdMeanAnomaly – pointer to double variable

Returned parameter:

pdMeanAnomaly – orbit mean anomaly value, in degrees.

Return value: int – 0 = success, otherwise error

int AppGetMeanMotion

(HANDLE zHandle,
double* pdMeanMotion)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdMeanMotion – pointer to double variable

Returned parameter:

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

Return value: int – 0 = success, otherwise error

int AppGetMeanMotion1stDeriv

(HANDLE zHandle,
double* pdMeanMotion1stDeriv)

Usage: Returns the orbital element 'First Time Derivative of the Mean Motion' value, as specified in the *AppSetMeanMotion1stDeriv*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdMeanMotion1stDeriv – pointer to double variable

Returned parameter:

pdMeanMotion1stDeriv – first derivative of mean motion.
Return value: int – 0 = success, otherwise error

int AppGetMeanMotion2ndDeriv

(HANDLE zHandle,
double* pdMeanMotion2ndDeriv)

Usage: Returns the orbital element 'Second Time Derivative of the Mean Motion' value, as specified in the *AppSetMeanMotion2ndDeriv*() method.

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pdMeanMotion2ndDeriv – pointer to double variable
Returned parameter:
pdMeanMotion2ndDeriv – second derivative of mean motion.
Return value: int – 0 = success, otherwise error

int AppGetBStar

(HANDLE zHandle,
double* pdBStar)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pdBStar – pointer to double variable
Returned parameter:
pdBStar – ballistic coefficient value.
Return value: int – 0 = success, otherwise error

int AppGetAltitudeOfApogee

(HANDLE zHandle,
double* pdAltApogee)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pdAltApogee – pointer to double variable
Returned parameter:
pdAltApogee – orbit apogee altitude, in km.
Return value: int – 0 = success, otherwise error

int AppGetAltitudeOfPerigee

(HANDLE zHandle,
double* pdAltPerigee)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pdAltPerigee – pointer to double variable
Returned parameter:

pdAltPerigee – orbit perigee altitude, in km.
Return value: int – 0 = success, otherwise error

int AppGetLocalTimeOfApogee

(HANDLE zHandle,
double* pdLocTimeApogee)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pdLocTimeApogee – pointer to double variable
Returned parameter:
pdLocTimeApogee – local time of orbit perigee, in hours + fraction.
Return value: int – 0 = success, otherwise error

int AppGetLocalTimeMaxInclination

(HANDLE zHandle,
double* pdLocTimeMaxIncl)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pdLocTimeMaxIncl – pointer to double variable
Returned parameter:
pdLocTimeMaxIncl – local time of orbit maximum inclination, in hours + fraction.
Return value: int – 0 = success, otherwise error

int AppGetTimeOfPerigee

(HANDLE zHandle,
double* pdTimeOfPerigee)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pdTimeOfPerigee – pointer to double variable
Returned parameter:
pdTimeOfPerigee – orbit perigee time value, in Modified Julian Date form
Return value: int – 0 = success, otherwise error

int AppGetSemiMajorAxis

(HANDLE zHandle,
double* pdSemiMajorAxis)

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

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pdSemiMajorAxis – pointer to double variable
Returned parameter:
pdSemiMajorAxis – orbit semi-major axis length, in units of Re.

Return value: int – 0 = success, otherwise error

int AppGetGeosynchLon

(HANDLE zHandle,
double* pdGeosynchLon)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdGeosynchLon – pointer to double variable

Returned parameter:

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

Return value: int – 0 = success, otherwise error

int AppGetStateVectors

(HANDLE zHandle,
double* pvdPosition,
double* pvdVelocity)

Usage: Returns the satellite's position and velocity vector values, as specified in the *AppSetStateVectors*() method, or in the *AppSetPositionGEI*() and *AppSetVelocityGEI*() methods.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdPosition – pointer to double array, size=3

pvdVelocity – pointer to double array, size=3

Returned Parameters:

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

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

Return value: int – 0 = success, otherwise error

int AppGetPositionGEI

(HANDLE zHandle,
double* pdPosX,
double* pdPosY,
double* pdPosZ)

Usage: Returns the satellite's position vector, as specified in either the *AppSetPositionGEI*() or *AppSetStateVectors*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdPosX, pdPosY, pdPosZ – pointers to double variables

Returned Parameters:

pdPosX, pdPosY, pdPosZ – GEI coordinate system satellite position values, in km

Return value: int – 0 = success, otherwise error

int AppGetVelocityGEI

(HANDLE zHandle,
double* pdVelX,
double* pdVelY,

double* pdVelZ)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdVelX, pdVelY, pdVelZ – pointers to double variables

Returned Parameters:

pdVelX, pdVelY, pdVelZ – GEI coordinate system satellite velocity values, in km/sec

Return value: int – 0 = success, otherwise error

int AppGetCoordSys

(HANDLE zHandle,
char* szCoordSys)

Usage: Returns the coordinate system name, as specified in the *AppSetCoordSys*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szCoordSys – pointer to character string (suggested sizing = 16)

Returned parameter:

szCoordSys – coordinate system name.

Return value: int – 0 = success, otherwise error

int AppGetCoordSysUnits

(HANDLE zHandle,
char* szCoordUnits)

Usage: Returns the coordinate system units, as specified in the *AppSetCoordSys*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szCoordUnits – pointer to character string (suggested sizing = 16)

Returned parameter:

szCoordUnits – coordinate system units ('Re' or 'km').

Return value: int – 0 = success, otherwise error

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.

int AppSetModel

(HANDLE zHandle,
char* szModel)

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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szModel – model name: 'AE9', 'AP9', 'PlasmaE', 'PlasmaH', 'PlasmaHe' or 'PlasmaO'
Legacy models: 'AE8', 'AP8', 'CRRESELE', 'CRRESPRO' or 'CAMMICE'

Return value: int – 0 = success, otherwise error

int AppSetModelDBDir

(HANDLE zHandle,
char* szDataDir)

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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szDataDir – directory path for the IRENE database files.

Return value: int – 0 = success, otherwise error

int AppSetModelDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Specifies the name of the database file for flux model calculations. The use of this method is *not needed* when *AppSetModelDBDir*() 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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szDataSource – model database filename, including path

Return value: int – 0 = success, otherwise error

int AppSetKPhiDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Specifies the name of the file for the K/Phi database. The use of this method is *not needed* when *AppSetModelDBDir*() 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataSource – database filename, including path

Return value: int – 0 = success, otherwise error

int AppSetKHMinDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Specifies the name of the file for the K/Hmin database. The use of this method is *not needed* when *AppSetModelDBDir*() 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataSource – database filename, including path

Return value: int – 0 = success, otherwise error

int AppSetMagfieldDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Specifies the name of the file for the magnetic field model database. The use of this method is *not needed* when *AppSetModelDBDir*() 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataSource – magnetic field model database filename, including path

Return value: int – 0 = success, otherwise error

int AppSetDoseModelDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Specifies the name of the file for the dose calculation model database. The use of this method is *not needed* when *AppSetModelDBDir*() and/or *AppSetMagfieldDBFile*() 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataSource – dose calculation model database filename, including path

Return value: int – 0 = success, otherwise error

int AppSetAdiabatic

(HANDLE zHandle,

int iVerdict)

Usage: Specifies whether or not the full adiabatic invariant values are to be calculated and be available during model run. Not applicable for Legacy model runs.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iVerdict – flag for activating (1) or deactivating (0) the adiabatic invariant calculation.

Return value: int – 0 = success, otherwise error

int AppSetFluxType

(HANDLE zHandle,
char* szFluxType)

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 *AppSetFluxEnergies*() [or *AppFluxSetEnergy*()] and *AppSetFluxEnergies2*() [or *AppSetFluxEnergy2*()] methods, respectively.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetFluxEnergies

(HANDLE zHandle,
double* pvdEnergies,
int iNumEnergies)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdEnergies – array of energy values [MeV]; array size is specified by *iNumEnergies*

iNumEnergies – number of values in energy array

Return value: int – 0 = success, otherwise error

int AppSetFluxEnergy

(HANDLE zHandle,
double vdEnergy)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

dEnergy – energy value [MeV]

Return value: int – 0 = success, otherwise error

int AppClearFluxEnergies

(HANDLE zHandle)

Usage: Clears the flux energy list defined by previous calls to the App*SetFluxEnergy*() or *AppSetFluxEnergies*() methods.

Return value: -none-

int AppSetFluxEnergies2

(HANDLE zHandle,
double* pvdEnergies2,
int iNumEnergies)

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 *AppSetFluxEnergies*() or AppSetFluxEnergy() methods.

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method
pvdEnergies2 – array of energy values [MeV]; array size is specified by *iNumEnergies*
iNumEnergies – number of values in energy array

Return value: int – 0 = success, otherwise error

int AppSetFluxEnergy2

(HANDLE zHandle,
double vdEnergy)

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 *AppSetFluxEnergies*() or *AppSetFluxEnergy*() methods.

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method
dEnergy – energy value [MeV]

Return value: int – 0 = success, otherwise error

int AppClearFluxEnergies2

(HANDLE zHandle)

Usage: Clears the *upper bound* flux energy list defined by previous calls to the App*SetFluxEnergy2*() or *AppSetFluxEnergies2*() methods.

Return value: -none-

int AppSetPitchAngle

(HANDLE zHandle,
double 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method
dPitchAngle – pitch angle, in degrees (0-180)

Return value: int – 0 = success, otherwise error

int AppSetPitchAngles

(HANDLE zHandle,

double* pvdPitchAngles,
int iNumPitch)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdPitchAngles – array of pitch angles, size=*iNumPitch*, in degrees (0-180)

iNumPitch – number of values in pitch angle array

Return value: int – 0 = success, otherwise error

int AppClearPitchAngles

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppSetFluxMean

(HANDLE zHandle,
int iVerdict)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iVerdict – flag for activating (1) or deactivating (0) the calculation of mean flux values.

Return value: int – 0 = success, otherwise error

int AppSetFluxPercentile

(HANDLE zHandle,
int 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetFluxPercentiles

(HANDLE zHandle,
int* pviPercent,
int iNumPerc)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method
pviPercent – array of percentile flux values (1-99), size=*iNumPerc*, to be calculated by the model
iNumPerc – number of values in percentile array
Return value: int – 0 = success, otherwise error

int AppClearFluxPercentiles

(HANDLE zHandle)
Usage: Clears the current list of defined flux percentile values.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – 0 = success, otherwise error

int AppSetFluxPerturbedScenario

(HANDLE zHandle,
int 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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
iScenario – scenario number (1-999) of perturbed mean flux values to be calculated by the model
Return value: int – 0 = success, otherwise error

int AppSetFluxPerturbedScenarios

(HANDLE zHandle,
int* pviScenario,
int iNumScen)
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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pviScenario – array of scenario numbers (1-999), size=*iNumScen*, of perturbed mean flux values to be calculated by the model
iNumScen – number of values in scenario array
Return value: int – 0 = success, otherwise error

int AppSetFluxPerturbedScenRange

(HANDLE zHandle,
int iScenarioMin,
int 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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method

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: int – 0 = success, otherwise error

int AppClearFluxPerturbedScenarios

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppSetFluxMonteCarloScenario

(HANDLE zHandle,
int 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetFluxMonteCarloScenarios

(HANDLE zHandle,
int* pviScenario,
int iNumScen)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pviScenario – array of scenario numbers (1-999), size=*iNumScen*, of Monte Carlo flux values to be calculated by the model

iNumScen – number of values in scenario array

Return value: int – 0 = success, otherwise error

int AppSetFluxMonteCarloScenRange

(HANDLE zHandle,
int iScenarioMin,
int 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method
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: int – 0 = success, otherwise error

int AppClearFluxMonteCarloScenarios

(HANDLE zHandle)
Usage: Clears the current list of defined Monte Carlo scenario numbers.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – 0 = success, otherwise error

int AppSetMonteCarloEpochTime

(HANDLE zHandle,
double dEpochTime)
Usage: Specifies the reference time used in the time progression of the Monte Carlo flux calculations.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dEpochTime – Monte Carlo reference time, in Modified Julian Date form
Return value: int – 0 = success, otherwise error

int AppSetMonteCarloFluxPerturb

(HANDLE zHandle,
int iVerdict)
Usage: Enables (*true*) or disables (*false*) 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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
iVerdict – 1 *(true)* or 0 *(false)* for the use of flux perturbations in the Monte Carlo calculations.
Return value: int – 0 = success, otherwise error

int AppSetMonteCarloWorstCase

(HANDLE zHandle,
int iVerdict)
Usage: Enables (*true*) or disables (*false*) 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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
iVerdict – 1 *(true)* or 0 *(false)* for the tracking of the 'maximum-to-date' Monte Carlo results.
Return value: int – 0 = success, otherwise error

int AppGetModel

(HANDLE zHandle,
char* szModel)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szModel – pointer to character string (suggested sizing = 16)

Returned parameter:

szModel – model name.

Return value: int – 0 = success, otherwise error

int AppGetModelDBDir

(HANDLE zHandle,
char* szDataDir)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataDir – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataDir – model database directory.

Return value: int – 0 = success, otherwise error

int AppGetModelDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Returns the name of the database file for flux model calculations. This will be available immediately, when specified using the *AppSetModelDBFile*() method. When the *AppSetModelDBDir*() method is used, the automatically determined filename will be available after a call to the *AppValidateParameters*() or *AppRunModel*() methods.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataSource – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataSource – model database filename.

Return value: int – 0 = success, otherwise error

int AppGetKPhiDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Returns the name of the file for the K/Phi database. This will be available immediately, when specified using the *AppSetKPhiDBFile*() method. When the *AppSetModelDBDir*() method is used, the automatically determined filename will be available after a call to the *AppValidateParameters*() or *AppRunModel*() methods.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataSource – pointer to character string (suggested sizing = 256)

Returned parameter:
szDataSource – K/Phi database filename.
Return value: int – 0 = success, otherwise error

int AppGetKHMinDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Returns the name of the file for the K/Hmin database. This will be available immediately, when specified using the *AppSetKHMinDBFile*() method. When the *AppSetModelDBDir*() method is used, the automatically determined filename will be available after a call to the *AppValidateParameters*() or *AppRunModel*() methods.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szDataSource – pointer to character string (suggested sizing = 256)
Returned parameter:
szDataSource – K/Hmin database filename.
Return value: int – 0 = success, otherwise error

int AppGetMagfieldDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Returns the name of the file for the magnetic field model database. This will be available immediately, when specified using the *AppSetMagfieldDBFile*() method. When the *AppSetModelDBDir*() method is used, the automatically determined filename will be available after a call to the *AppValidateParameters*() or *AppRunModel*() methods.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szDataSource – pointer to character string (suggested sizing = 256)
Returned parameter:
szDataSource – magnetic field model database filename.
Return value: int – 0 = success, otherwise error

int AppGetDoseModelDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Returns the name of the file for the dose calculation model database. This will be available immediately, when specified using the *AppSetDoseModelDBFile*() method. When the *AppSetModelDBDir*() method is used, the automatically determined filename will be available after a call to the *AppValidateParameters*() or *AppRunModel*() methods.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szDataSource – pointer to character string (suggested sizing = 256)
Returned parameter:
szDataSource – dose calculation model database filename.
Return value: int – 0 = success, otherwise error

int AppGetAdiabatic

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: 1 *(true)* or 0 *(false).*

int AppGetFluxType

(HANDLE zHandle,
char* szFluxType)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szFluxType – pointer to character string (suggested sizing = 16)

Returned parameter:

szFluxType – flux type identifier string.

Return value: int – 0 = success, otherwise error

int AppGetNumFluxEnergies

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – number of energy levels.

int AppGetFluxEnergies

(HANDLE zHandle,
double* pvdEnergies)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdEnergies – pointer to double array, sized according to *AppGetNumFluxEnergies*() results.

Returned Parameters:

pvdEnergies – array of energy values, in units of MeV.

Return value: int - Number of records returned (≥0 = success, <0 = error)

int AppGetNumFluxEnergies2

(HANDLE zHandle)

Usage: Returns the number of currently defined upper bounds of the flux energy levels, specified in the *AppSetFluxEnergies2*() or *AppSetFluxEnergy2*() methods.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – number of energy levels.

int AppGetFluxEnergies2

(HANDLE zHandle,
double* pvdEnergies2)

Usage: Returns the upper bounds of the energy bins, for '2PtDiff' flux type, as specified in the *AppSetFluxEnergies2*() or *AppSetFluxEnergy2*() methods.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdEnergies2 – pointer to double array, sized according to *AppGetNumFluxEnergies2*() results.

Returned Parameters:

pvdEnergies2 – array of energy values, in units of MeV.

Return value: int - Number of records returned (≥0 = success, <0 = error)

int AppGetNumPitchAngles

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – number of pitch angles.

int AppGetPitchAngles

(HANDLE zHandle,
double* pvdPitchAngles)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdPitchAngles – pointer to double array, sized according to *AppGetNumPitchAngles*() results.

Returned Parameters:

pvdPitchAngles – array of pitch angles, in degrees.

Return value: int - Number of records returned (≥0 = success, <0 = error)

int AppGetFluxMean

(HANDLE zHandle)

Usage: Returns the current state for the calculation of the 'mean' flux values.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 1 *(true)* or 0 *(false)*, as specified in the *AppSetFluxMean*() method.

int AppGetNumFluxPercentiles

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – number of flux percentiles.

int AppGetFluxPercentiles

(HANDLE zHandle,
int* pviPercent)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pviPercent – pointer to int array, sized according to *AppGetNumFluxPercentiles*() results.

Returned Parameter:

pviPercent – array of percentile flux values.

Return value: int - Number of records returned (≥0 = success, <0 = error)

int AppGetNumFluxPerturbedScenarios

(HANDLE zHandle)

Usage: Returns the number of perturbed mean scenarios defined for the model calculation, as specified in the *AppSetFluxPerturbedScenario*(), *AppSetFluxPerturbedScenarios*(), or *AppSetFluxPerturbedScenRange*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – number of scenarios.

int AppGetFluxPerturbedScenarios

(HANDLE zHandle,
int* pviScenario)

Usage: Returns the array of defined perturbed flux scenario numbers, as specified in the *AppSetFluxPerturbedScenario*(), *AppSetFluxPerturbedScenarios*(), or *AppSetFluxPerturbedScenRange*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pviScenario – pointer to int array, sized according to *AppGetNumFluxPerturbedScenarios*() results.

Returned parameter:

pviScenario – array of scenario numbers.

Return value: int - Number of records returned (≥0 = success, <0 = error)

int AppGetNumFluxMonteCarloScenarios

(HANDLE zHandle)

Usage: Returns the number of Monte Carlo scenarios defined for the model calculation, as specified in the *AppSetFluxMonteCarloScenario*() or *AppSetFluxMonteCarloScenarios*(), or *AppSetFluxMonteCarloScenRange*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – number of scenarios.

int AppGetFluxMonteCarloScenarios

(HANDLE zHandle,
int* pviScenario)

Usage: Returns the array of defined Monte Carlo flux scenario numbers, as specified in the *AppSetFluxMonteCarloScenario*() or *AppSetFluxMonteCarloScenarios*(), or *AppSetFluxMonteCarloScenRange*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pviScenario – pointer to int array, sized according to *AppGetNumFluxMonteCarloScenarios*() results.

Returned parameter:

pviScenario – array of scenario numbers.

Return value: int - Number of records returned (≥0 = success, <0 = error)

int AppGetMonteCarloEpochTime

(HANDLE zHandle,
double* pdMCEpochTime)

Usage: Returns the reference time used in the time progression of the Monte Carlo flux calculations, as specified in the *AppSetMonteCarloEpochTime*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdMCEpochTime – pointer to double variable

Returned parameter:

pdMCEpochTime – Monte Carlo reference time, in Modified Julian Date form.

Return value: int – 0 = success, otherwise error

int AppGetMonteCarloFluxPerturb

(HANDLE zHandle)

Usage: Returns the current Monte Carlo perturbation mode setting, as specified in the *AppSetMonteCarloFluxPerturb*() method

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: 1 *(true)* or 0 *(false)*.

int AppGetMonteCarloWorstCase

(HANDLE zHandle)

Usage: Returns the current state for the tracking of 'maximum-to-date' Monte Carlo results, as specified in the *AppSetMonteCarloWorstCase*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: 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.

int AppSetAccumMode

(HANDLE zHandle,
char* szAccumMode)

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 *AppSetAccumInterval[Sec]*() method.

Please consult the User's Guide for more details about these accumulation modes.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szAccumMode – accumulation mode identifier: 'Cumul'|'Cumulative', 'Intv'|'Interval', 'Full', 'Boxcar' or 'Expon'|'Exponential' [the 'Boxcar' mode requires both interval and increment values to be defined]

Return value: int – 0 = success, otherwise error

int AppClearAccumModes

(HANDLE zHandle)

Usage: Removes all previous accumulation modes specified via the *AppSetAccumMode*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppSetAccumInterval

(HANDLE zHandle,
double dVal)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

dVal – time duration, in units of days+fraction.

Return value: int – 0 = success, otherwise error

int AppSetAccumIntervalSec

(HANDLE zHandle,
double dVal)

Usage: Same as *AppSetAccumInterval*(), except that the duration is specified in seconds. When no intervals are specified, the accumulation interval defaults to 86400 seconds (=1.0 days).

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

dVal – time duration, in units of seconds.

Return value: int – 0 = success, otherwise error

int AppClearAccumIntervals

(HANDLE zHandle)

Usage: Removes all previous accumulation intervals specified via the *AppSetAccumInterval[Sec]*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppSetAccumIncrementSec

(HANDLE zHandle,
double dVal)

Usage: Specifies the time delta for the shift of the 'Boxcar' accumulation mode time windows.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

dVal – 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: int – 0 = success, otherwise error

int AppSetAccumIncrementFrac

(HANDLE zHandle,
double dVal)

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 *AppSetAccumInterval[Sec]*() calls) .

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetReportTimes

(HANDLE zHandle,
double dTimeRef,
double 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 App*SetReportTimesSec*) may be called multiple times to build a sequence of successive reporting periods with different cadences.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

dTimeRef – Reporting *reference* time, in Modified Julian Date form (this time is <u>not</u> 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: int – 0 = success, otherwise error

int AppSetReportTimesSec

(HANDLE zHandle,
double dTimeRef,
double 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 *AppSetReportTimes*) may be called multiple times to build a sequence of successive reporting periods with different cadences.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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: int – 0 = success, otherwise error

int AppSetReportAtTime

(HANDLE zHandle,
double 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 *AppSetReportTimes*() and/or *AppSetReportTimesSec*() methods.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppClearReportTimes

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppClearReportAtTime

(HANDLE zHandle)

Usage: Removes any previously defined 'ReportAt' time specifications.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppSetFluence

(HANDLE zHandle,
int iVerdict)

Usage: Specifies the calculation of fluence values from the flux results of the model run. Use of the *AppSetAccumMode*() and *AppSetAccumInterval[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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iVerdict – flag for activating (1) or deactivating (0) the calculation of fluence values.

Return value: int – 0 = success, otherwise error

---

int AppGetNumAccumModes

(HANDLE zHandle)

Usage: Returns the number of accumulation modes defined, as specified in calls to the *AppSetAccumMode*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppGetAccumMode

(HANDLE zHandle,
char* szAccumMode)

Usage: Returns the *first* accumulation mode, as specified in calls to the *AppSetAccumMode*() method. [If none were specified, the appropriate default mode will be set within a call to the optional *AppValidateParameters*() method, when all parameter specifications have been completed.]

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szAccumMode – pointer to character string (suggested sizing = 32)

Returned parameter:

szAccumMode – accumulation mode; '-none-' if no modes are defined.

Return value: int – 0 = success, otherwise error

int AppGetAccumModeEntry

(HANDLE zHandle,
char* szAccumMode,
int iIdent)

Usage: Returns the accumulation mode entry according to a numeric identifier, based on the *order* of calls to the *AppSetAccumMode*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

szAccumMode – pointer to character string (suggested sizing = 32)

Returned parameter:

szAccumMode – accumulation mode; '-none-' if no modes are defined; 'error' if invalid identifier.

Return value: int – 0 = success, otherwise error

int AppGetNumAccumIntervals

(HANDLE zHandle)

Usage: Returns the number of accumulation intervals defined, as specified in calls to the *AppSetAccumInterval[Sec]*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – number of accumulation interval entries.

int AppGetAccumIntervalSec

(HANDLE zHandle,

double* pdAccumSec)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdAccumSec – pointer to double variable

Returned parameter:

pdAccumSec – interval length, in seconds.

Return value: int – 0 = success, otherwise error.

int AppGetAccumIntervalSecEntry

(HANDLE zHandle,
double* pdAccumSec,
int iIdent)

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 *AppSetAccumInterval[Sec]*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdAccumSec – pointer to double variable

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

Returned parameter:

pdAccumSec – interval length, in seconds.

Return value: int – 0 = success, otherwise error.

int AppGetAccumIncrementSec

(HANDLE zHandle,
double* pdIncrementSec)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdIncrementSec – pointer to double variable

Returned parameter:

pdIncrementSec – time delta, in seconds.

Return value: int – 0 = success, otherwise error

int AppGetAccumIncrementFrac

(HANDLE zHandle,
double* pdIncrementFrac)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pdIncrementFrac – pointer to double variable

Returned parameter:
pdIncrementFrac – interval length fraction.
Return value: int – 0 = success, otherwise error

int AppGetNumReportTimes

(HANDLE zHandle)
Usage: Returns the number of 'Report Time' entries defined, as specified in calls to the *AppSetReportTimes*() and/or *AppSetReportTimesSec*() methods.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – number of 'Report Time' entries.

int AppGetReportTimesSec

(HANDLE zHandle,
double* pvdTimeRef,
double* pvdCadenceSec)
Usage: Returns the arrays of the defined 'Report Time' entries, as specified in calls to the *AppSetReportTimes*() and/or *AppSetReportTimesSec*() methods.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pvdTimeRef – pointer to double array, sized according to *AppGetNumReportTimes*() results.
pvdCadenceSec – pointer to double array, sized according to *AppGetNumReportTimes*() results.
Returned Parameters:
pvdTimeRef – array of defined reference times, in Modified Julian Date form.
pvdCadenceSec – array of associated cadence values, in units of seconds.
Return value: int - Number of records returned (≥0 = success, <0 = error)

int AppGetNumReportAtTime

(HANDLE zHandle)
Usage: Returns the number of 'Report At' entries defined, as specified in calls to the *AppSetReportAtTime*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – number of 'Report At' entries.

int AppGetReportAtTime

(HANDLE zHandle,
double* pvdTimeVal)
Usage: Returns the array of the defined 'Report At' entries, as specified in calls to *AppSetReportAtTime*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pvdTimeVal – pointer to double array, sized according to *AppGetNumReportAtTime*() results.
Returned Parameters:
pvdTimeVal – array of defined report times, in Modified Julian Date form.
Return value: int - Number of records returned (≥0 = success, <0 = error)

int AppGetFluence

(HANDLE zHandle)

Usage: Returns the current state for the calculation of fluence values from the flux, as specified in the *AppSetFluence*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: 1 *(true)* or 0 *(false).*

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'.

int AppSetDoseRate

(HANDLE zHandle,
int iVerdict)

Usage: Specifies the calculation of dose rate values from the flux results of the model run. Use of the *AppSetAccumMode*() and *AppSetAccumInterval[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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iVerdict – flag for activating (1) or deactivating (0) the calculation of dose rate values.

Return value: int – 0 = success, otherwise error

int AppSetDoseAccum

(HANDLE zHandle,
int iVerdict)

Usage: Specifies the calculation of cumulative or accumulated dose values from the flux results of the model run. Use of the *AppSetAccumMode*() and *AppSetAccumInterval[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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iVerdict – flag for activating (1) or deactivating (0) the calculation of dose accum values.

Return value: int – 0 = success, otherwise error

int AppSetDoseDepthValues

(HANDLE zHandle,
double* pvdDepths,
int iNumDepths)

Usage: Specifies the list of aluminum shielding thickness depths, in the units specified by *AppSetDoseDepthUnits*() 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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdDepths – array of depth values, size=*iNumDepths*

iNumDepths – number of values in depth array
Return value: int – 0 = success, otherwise error

int AppSetDoseDepthUnits

(HANDLE zHandle,
char* szDepthUnits)

Usage: Specifies the measurement units associated with the depth values specified using the *AppSetDoseDepthValues*() method.

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szDepthUnits – unit specification: 'millimeters'|'mm', 'mils' or 'gpercm2'
Return value: int – 0 = success, otherwise error

int AppSetDoseDepths

(HANDLE zHandle,
double* pvdDepths,
int iNumDepths,
char* szDepthUnits)

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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pvdDepths – array of depth values, size=*iNumDepths*
iNumDepths – number of values in depth array
szDepthUnits – unit specification: 'millimeters'|'mm', 'mils' or 'gpercm2'
Return value: int – 0 = success, otherwise error

int AppSetDoseDetector

(HANDLE zHandle,
char* szDetector)

Usage: Specifies the dose detector material type that lies behind the aluminum shielding.

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szDetector – material name: 'Aluminum'|'Al', 'Graphite', 'Silicon'|'Si', 'Air', 'Bone', 'Tissue', 'Calcium'|'Ca', 'Gallium'|'Ga', 'Lithium'|'Li', 'Glass'|'SiO2', 'Water'|'H2O'
Return value: int – 0 = success, otherwise error

int AppSetDoseGeometry

(HANDLE zHandle,
char* szGeometry)

Usage: Specifies the geometry of the aluminum shielding in front of (or around) the detector target.

Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szGeometry – configuration name: 'Spherical4pi', 'Spherical2pi', 'FiniteSlab' or 'SemiInfiniteSlab'
Return value: int – 0 = success, otherwise error

int AppSetDoseNuclearAttenMode

(HANDLE zHandle,
char* szNucAttenMode)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szNucAttenMode – attenuation mode: 'None', 'NuclearInteractions' or 'NuclearAndNeutrons'

Return value: int – 0 = success, otherwise error

int AppSetDoseWithBrems

(HANDLE zHandle,
int iVerdict)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iVerdict – flag for including(1) or excluding(0) the bremsstrahlung contributions.

Return value: int – 0 = success, otherwise error

int AppSetUseDoseKernel

(HANDLE zHandle,
int iVerdict)

Usage: Specifies the calculation of the dose values using the kernel-based method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iVerdict – flag for activating (1) or deactivating (0) the use of the dose kernel.

Return value: int – 0 = success, otherwise error

int AppSetDoseKernelDir

(HANDLE zHandle,
char* szDataDir)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataDir – directory path for the dose kernel XML files.

Return value: int – 0 = success, otherwise error

int AppSetDoseKernelFile

(HANDLE zHandle,
char* szDataSource)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataSource – specific Dose kernel XML filename (including path)

Return value: int – 0 = success, otherwise error

---

int AppGetDoseRate

(HANDLE zHandle)

Usage: Returns the current state for the calculation of dose rate values, as specified in the *AppSetDoseRate*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: 1 *(true)* or 0 *(false)*.

int AppGetDoseAccum

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: 1 *(true)* or 0 *(false)*.

int AppGetNumDoseDepthValues

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – number of dose depths.

int AppGetDoseDepthValues

(HANDLE zHandle,
double* pvdDepths)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdDepths – pointer to double array, sized according to *AppGetNumDoseDepthValues*() results.

Returned parameter:

pvdDepths – array of depth values.

Return value: int - Number of records returned (≥0 = success, <0 = error)

int AppGetDoseDepthUnits

(HANDLE zHandle,
char* szDepthUnits)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDepthUnits – pointer to character string (suggested sizing = 16)
Returned parameter:
szDepthUnits – dose depth units.
Return value: int – 0 = success, otherwise error

int AppGetDoseDetector

(HANDLE zHandle,
char* szDetector)

Usage: Returns the dose detector material type that lies behind the aluminum shielding, as specified in the *AppSetDoseDetector*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szDetector – pointer to character string (suggested sizing = 32)
Returned parameter:
szDetector – material type.
Return value: int – 0 = success, otherwise error

int AppGetDoseGeometry

(HANDLE zHandle,
char* szGeometry)

Usage: Returns the geometry of the aluminum shielding in front of (or around) the detector target, as specified in the *AppSetDoseGeometry*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szGeometry – pointer to character string (suggested sizing = 32)
Returned parameter:
szGeometry – dose shielding geometry.
Return value: int – 0 = success, otherwise error

int AppGetDoseNuclearAttenMode

(HANDLE zHandle,
char* szNucAttenMode)

Usage: Returns the 'Nuclear Attenuation' mode used during the dose calculations, as specified in the *AppSetDoseNuclearAttenMode*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szNucAttenMode – pointer to character string (suggested sizing = 32)
Returned parameter:
szNucAttenMode – Dose calculation nuclear attenuation mode.
Return value: int – 0 = success, otherwise error

int AppGetDoseWithBrems

(HANDLE zHandle)

Usage: Returns the current state for the inclusion of the bremsstrahlung contribution in the electron dose values calculated, as specified in the *AppSetDoseWithBrems*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: 1 *(true)* or 0 *(false).*

int AppGetUseDoseKernel

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: 1 *(true)* or 0 *(false).*

int AppGetDoseKernelDir

(HANDLE zHandle,
char* szDataSource)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataSource – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataSource – directory string for the dose kernel XML files.

Return value: int – 0 = success, otherwise error

int AppGetDoseKernelFile

(HANDLE zHandle,
char* szDataSource)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataSource – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataSource – Dose kernel XML filename.

Return value: int – 0 = success, otherwise error

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.

int AppSetAggregMedian

(HANDLE zHandle)

Usage: Adds the 50% value to the list for aggregation confidence level calculations.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppSetAggregConfLevel

(HANDLE zHandle,
int iPercent)

Usage: Adds the specified percent value to the list for aggregation confidence level calculations.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iPercent – percent value to add to list for aggregation confidence level calculations (0-100 valid).

Return value: int – 0 = success, otherwise error

int AppSetAggregConfLevels

(HANDLE zHandle,
int* pviPercent,
int iNumConf)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pviPercent – array of percent values, size=*iNumConf*, for the aggregation confidence level calculations (0-100 valid).

iNumConf – number of values in confidence level array

Return value: int – 0 = success, otherwise error

int AppClearAggregConfLevels

(HANDLE zHandle)

Usage: Clears the accumulated list of percent values for aggregation confidence level calculations.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppSetAggregMean

(HANDLE zHandle)

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.*

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

---

int AppGetNumAggregConfLevels

(HANDLE zHandle)

Usage: Returns the number of confidence levels for the aggregation calculations, as specified in *AppSetAggregConfLevel*(), *AppSetAggregConfLevels*(), or through *AppSetAggregMean*() or *AppSetAggregMedian*() methods.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – number of confidence levels.

int AppGetAggregConfLevels

(HANDLE zHandle,
int* pviPercent)

Usage: Returns the list of confidence level values, as specified in *AppSetAggregConfLevel*(), *AppSetAggregConfLevels*(), or through *AppSetAggregMean*() or *AppSetAggregMedian*() methods.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pviPercent – pointer to int array, sized according to *AppGetNumAggregConfLevels*() results.
Returned parameter:
pviPercent – array of percent values for the aggregation confidence level calculations.
Return value: int - Number of records returned (≥0 = success, <0 = error)

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.

int AppSetLegActivityLevel

(HANDLE zHandle,
char* szActivityLevel)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetLegActivityRange

(HANDLE zHandle,
char* szActivityRange)

Usage: Specifies the geomagnetic activity level parameter for the CRRESELE Legacy model run. Only one of the *AppSetActivityRange*() or *AppSet15DayAvgAp*() methods may be used, otherwise an error is flagged.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szActivityRange – 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: int – 0 = success, otherwise error

int AppSetLeg15DayAvgAp

(HANDLE zHandle,
double d15DayAvgAp)

Usage: Specifies the 15-day average Ap value for the CRRESELE Legacy model run. Only one of the *AppSetActivityRange*() or *AppSet15DayAvgAp*() methods may be used, otherwise an error is flagged.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

int AppSetLegFixedEpoch

(HANDLE zHandle,
int iVerdict)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iVerdict – 1 *(true)* or 0 *(false)*; when *false*, the ephemeris year is used for the magnetic field model.
Return value: int – 0 = success, otherwise error

int AppSetLegShiftSAA

(HANDLE zHandle,
int iVerdict)

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 ***AppSetFixedEpoch*** method is set to 0 *(false)*.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
iVerdict – 1 *(true)* or 0 *(false)*.
Return value: int – 0 = success, otherwise error

---

int AppGetLegActivityLevel

(HANDLE zHandle,
char* szActivityLevel)

Usage: Returns the geomagnetic activity level parameter for the CRRESPRO, AE8 or AP8 Legacy model, as specified in the *AppSetLegActivityLevel*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szActivityLevel – pointer to character string (suggested sizing = 32)
Returned parameter:
szActivityLevel – geomagnetic activity level specification string.
Return value: int – 0 = success, otherwise error

int AppGetLegActivityRange

(HANDLE zHandle,
char* szActivityRange)

Usage: Returns the geomagnetic activity level parameter for the CRRESELE Legacy model, as specified in the *AppSetLegActivityRange*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szActivityRange – pointer to character string (suggested sizing = 32)
Returned parameter:
szActivityRange – geomagnetic activity level specification string.
Return value: int – 0 = success, otherwise error

int AppGetLeg15DayAvgAp

(HANDLE zHandle,
double* pd15DayAvgAp)

Usage: Returns the 15-day average Ap value for the CRRESELE Legacy model, as specified in the *AppSetLeg15DayAvgAp*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
pd15DayAvgAp – pointer to double variable
Returned parameter:

pd15DayAvgAp – 15-day average Ap value.
Return value: int – 0 = success, otherwise error

int AppGetLegFixedEpoch

(HANDLE zHandle)
Usage: Returns the current setting for the use of the model-specific fixed epoch, as specified in the *AppSetLegFixedEpoch*() method.
Return value: int – 1 *(true)* or 0 *(false)*.

int AppGetLegShiftSAA

(HANDLE zHandle)
Usage: Returns the current setting of shifting the SAA from its fixed-epoch location, as specified in the *AppSetLegShiftSAA*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
Return value: int – 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).

int AppSetCamMagfieldModel

(HANDLE zHandle,
char* szMFModel)
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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szMFModel – magnetic field model specification: 'igrf' or 'igrfop'.
Return value: int – 0 = success, otherwise error

int AppSetCamDataFilter

(HANDLE zHandle,
char* szDataFilter)
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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szDataFilter – data filter specification: 'all' or 'filtered' .
Return value: int – 0 = success, otherwise error

int AppSetCamPitchAngleBin

(HANDLE zHandle,
char* szPitchAngleBin)
Usage: Specifies the pitch angle bin for the CAMMICE Legacy model run.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method

szPitchAngleBin – 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: int – 0 = success, otherwise error

int AppSetCamSpecies

(HANDLE zHandle,
char* szSpecies)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

Return value: int – 0 = success, otherwise error

---

int AppGetCamMagfieldModel

(HANDLE zHandle,
char* szMFModel)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szMFModel – pointer to character string (suggested sizing = 16)

Returned parameter:

szMFModel – magnetic field model specification.

Return value: int – 0 = success, otherwise error

int AppGetCamDataFilter

(HANDLE zHandle,
char* szDataFilter)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szDataFilter – pointer to character string (suggested sizing = 16)

Returned parameter:

szDataFilter – data filter specification.

Return value: int – 0 = success, otherwise error

int AppGetCamPitchAngleBin

(HANDLE zHandle,
char* szPitchAngleBin)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

szPitchAngleBin – pointer to character string (suggested sizing = 16)

Returned parameter:
szPitchAngleBin – pitch angle bin identification.
Return value: int – 0 = success, otherwise error

int AppGetCamSpecies

(HANDLE zHandle,
char* szSpecies)

Usage: Returns the particle species for the CAMMICE Legacy model, as specified in the *AppSetCamSpecies*() method.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
szSpecies – pointer to character string (suggested sizing = 16)
Returned parameter:
szSpecies – species identification.
Return value: int – 0 = success, otherwise error

Model Execution and Results:

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

Following a call to the *AppRunModel*() method, the results from the requested calculations are accessible via the various *AppFlyin**() and *AppGet*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 *AppFlyin**() and *AppGet*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 *AppRunModel*() method, the sizing of these data access segments may be adjusted using the *AppSetChunkSize*() method. A call to the *AppResetModelData*() 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 'AppGetModelData' method under the hood, so is accessing the same data 'stream' (based on the data type and percentile/scenario identifier, as well as the accumulation mode and interval identifier).

The *AppFlyin**() and *AppGet*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 may 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 *AppGetCoordSys*() and AppG*etCoordUnit*() 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.

int AppRunModel

(HANDLE zHandle)

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.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppGetEphemeris

(HANDLE zHandle,
double* pvdTimes,
double* pvdCoord1,
double* pvdCoord2,
double* pvdCoord3)

Usage: Returns the ephemeris information, either generated or specified.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdTimes – pointer to double array, sized according to *AppGetChunkSize*() results

pvdCoord1, pvdCoord2, pvdCoord3 – pointers to double arrays, sized according to *AppGetChunkSize*() results

Returned parameters:

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

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of ephemeris position values, in the coordinate system and units previously specified (also accessible from *AppGetCoordSys*() and *AppGetCoordUnit*() 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.

Return value: int – number of ephemeris records returned (≥0 = success, <0 = error)

int AppGetNumDir

(HANDLE zHandle)

Usage: Returns the number of data directions in the generated flux data.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – number of directions: 1 when 'omnidirectional', otherwise, number of defined pitch angles. (when 'omnidirectional', the *AppGetNumPitchAngles*() method returns 0).

int AppFlyinMean2D

(HANDLE zHandle,
double* pvvdFluxData,
char* szAccumMode,
int iAccumIntvIdent)

Usage: Returns the 'Mean' model flux values, for omni-directional model runs. A previous call to the *AppSetFluxMean(1)* method is required for these results to be available for the Ae9/Ap9/SPM models. This method may also be used for accessing flux results from the 'Legacy' models. Flux accumulated average values may be obtained using this method when specifying non-default argument values. The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvvdFluxData – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumEnergies*() results.

szAccuMode – [default="default"] 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 *AppSetAccumMode*() method. See that method description for more information. The "default" translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppFlyinMean

(HANDLE zHandle,
double* pvvvdFluxData,
char* szAccumMode,
int iAccumIntvIdent)

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

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvvdFluxData – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumEnergies*() x *AppGetNumDir*() results.

szAccuMode – [default="default"] 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 *AppSetAccumMode*() method. See that method description for more information. The "default" translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppFlyinMeanPlus

(HANDLE zHandle,
double* pvdTimes
double* pvdCoord1,
double* pvdCoord2,
double* pvdCoord3,
double* pvvdPitchAngles,
double* pvvvdFluxData,
char* szAccumMode,
int iAccumIntvIdent)

Usage: Returns the 'Mean' model flux values, along with the associated ephemeris values and pitch angles. This method can also be used for accessing flux results from the 'Legacy' models. A previous call to the *AppSetFluxMean(1)* method is required for these results to be available for the Ae9/Ap9/SPM models. Flux accumulated average values may be obtained using this method when specifying non-default argument values.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdTimes – pointer to double array, sized according to *AppGetChunkSize*() results

pvdCoord1, pvdCoord2, pvdCoord3 – pointers to double arrays, sized according to *AppGetChunkSize*() results

pvvdPitchAngles – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumDir*() results.

pvvvdFluxData – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumEnergies*() x *AppGetNumDir*() results.

szAccuMode – [default="default"] 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 *AppSetAccumMode*() method. See that method description for more information. The "default" translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

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

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of ephemeris position values, in the coordinate system and units previously specified.

pvvdPitchAngles – 2-dimensional arrray of associated pitch angles; will contain zeros if omni-directional.[time,direction]

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppFlyinPercentile

```
( HANDLE zHandle,
  int iPercentile,
  double* pvvvdFluxData,
  char* szAccumMode,
  int iAccumIntvIdent )
```

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 *AppSetFluxPercentile()* methods. Flux accumulated average values may be obtained using this method when specifying non-default argument values.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

pvvvdFluxData – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumEnergies*() x *AppGetNumDir*() results.

szAccuMode – [default="default"] 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 *AppSetAccumMode*() method. See that method description for more information. The "default" translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppFlyinPercentilePlus

(HANDLE zHandle,
int iPercentile,
double* pvdTimes,
double* pvdCoord1,
double* pvdCoord2,
double* pvdCoord3,
double* pvvdPitchAngles,
double* pvvvdFluxData,
char* szAccumMode,
int iAccumIntvIdent)

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 *AppSetFluxPercentile()* methods. Flux accumulated average values may be obtained using this method when specifying non-default argument values.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

pvdTimes – pointer to double array, sized according to *AppGetChunkSize*() results

pvdCoord1, pvdCoord2, pvdCoord3 – pointers to double arrays, sized according to *AppGetChunkSize*() results

pvvdPitchAngles – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumDir*() results.

pvvvdFluxData – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumEnergies*() x *AppGetNumDir*() results.

szAccuMode – [default="default"] 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 *AppSetAccumMode*() method. See that method description for more information. The "default" translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or

only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

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

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of ephemeris position values, in the coordinate system and units previously specified.

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

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppFlyinPerturbedMean

(HANDLE zHandle,
int iScenario,
double* pvvvdFluxData,
char* szAccumMode,
int iAccumIntvIdent)

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 *AppSetFluxPerturbedScen [Range]()* methods. Flux accumulated average values may be obtained using this method when specifying non-default argument values.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

pvvvdFluxData – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumEnergies*() x *AppGetNumDir*() results.

szAccuMode – [default="default"] 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 *AppSetAccumMode*() method. See that method description for more information. The "default" translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppFlyinPerturbedMeanPlus

(HANDLE zHandle,
int iScenario,
double* pvdTimes,
double* pvdCoord1,
double* pvdCoord2,

double* pvdCoord3,
double* pvvdPitchAngles,
double* pvvvdFluxData,
char* szAccumMode,
int iAccumIntvIdent)

Usage: Returns the 'Perturbed Mean' 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 *AppSetFluxPerturbedScen[Range]()* methods. Flux accumulated average values may be obtained using this method when specifying non-default argument values.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

iFluxAccumAvg – flag for returning flux accumulation average values, if 1 *(true)*. When specified as 0 *(false),* returns flux values at the ephemeris input times and positions.

pvdTimes – pointer to double array, sized according to *AppGetChunkSize*() results

pvdCoord1, pvdCoord2, pvdCoord3 – pointers to double arrays, sized according to *AppGetChunkSize*() results

pvvdPitchAngles – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumDir*() results.

pvvvdFluxData – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumEnergies*() x *AppGetNumDir*() results.

szAccuMode – [default="default"] 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 *AppSetAccumMode*() method. See that method description for more information. The "default" translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

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

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of ephemeris position values, in the coordinate system and units previously specified.

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

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppFlyinMonteCarlo

(HANDLE zHandle,
int iScenario,
double* pvvvdFluxData,
char* szAccumMode,
int iAccumIntvIdent)

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 *AppSetFluxMonteCarloScen[Range]()* methods. Flux accumulated average values may be obtained using this method when specifying non-default argument values.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

pvvvdFluxData – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumEnergies*() x *AppGetNumDir*() results.

szAccuMode – [default="default"] 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 *AppSetAccumMode*() method. See that method description for more information. The "default" translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppFlyinMonteCarloPlus

```
( HANDLE zHandle,
  int iScenario,
  double* pvdTimes,
  double* pvdCoord1,
  double* pvdCoord2,
  double* pvdCoord3,
  double* pvvdPitchAngles,
  double* pvvvdFluxData,
  char* szAccumMode,
  int iAccumIntvIdent )
```

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 *AppSetFluxMonteCarloScen[Range]()* methods. Flux accumulated average values may be obtained using this method when specifying non-default argument values.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

pvdTimes – pointer to double array, sized according to *AppGetChunkSize*() results

pvdCoord1, pvdCoord2, pvdCoord3 – pointers to double arrays, sized according to *AppGetChunkSize*() results

pvvdPitchAngles – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumDir*() results.

pvvvdFluxData – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumEnergies*() x *AppGetNumDir*() results.

szAccuMode – [default="default"] 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 *AppSetAccumMode*() method. See that method description for more information. The "default" translates to 'Cumul'; in this context is the raw (*non*-accumulated) flux values.

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the flux accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

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

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of ephemeris position values, in the coordinate system and units previously specified.

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

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppGetModelData

```
( HANDLE zHandle,
  char* szDataType,
  char* szFluxMode,
  int iCalcVal,
  double* pvdTimes,
  double* pvdCoord1,
  double* pvdCoord2,
  double* pvdCoord3,
  double* pvvdPitchAngles,
  double* pvvvdData,
  char* szAccumMode,
  int iAccumIntvIdent )
```

Usage: Returns the model results from for the specified data type, flux mode and, if applicable, the percentile or scenario identifier, and the accumulation mode and interval identifier. See the following routine for accessing *aggregation* results. The associated ephemeris and pitch angles are also returned.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

szFluxMode – 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.

pvdTimes – pointer to double array, sized according to *AppGetChunkSize*() results

pvdCoord1, pvdCoord2, pvdCoord3 – pointers to double arrays, sized according to *AppGetChunkSize*() results

pvvdPitchAngles – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumDir*() results.

pvvvdData – pointer to double array, sized according to *AppGetChunkSize*() x (either *AppGetNumDoseDepthValues*() for dose results or *AppGetNumEnergies*() for all others) x *AppGetNumDir*() results.

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

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the data accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

pvdTimes – array of ephemeris time values, in Modified Julian Date form; when using an accumulation mode other than 'Cumul', this time specifies the *ending* time of the accumulation interval.

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of ephemeris position values, in the coordinate system and units previously specified.

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

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppGetAggregData

```
( HANDLE zHandle,
  char* szDataType,
  char* szFluxMode,
  int iPercent,
  double* pvdTimes,
  double* pvdCoord1,
  double* pvdCoord2,
  double* pvdCoord3,
  double* pvvdPitchAngles,
  double* pvvvdData,
  char* szAccumMode,
  int iAccumIntvIdent )
```

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.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

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

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

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

pvdTimes – pointer to double array, sized according to *AppGetChunkSize*() results

pvdCoord1, pvdCoord2, pvdCoord3 – pointers to double arrays, sized according to *AppGetChunkSize*() results

pvvdPitchAngles – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumDir*() results.

pvvvdData – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumEnergies*() x *AppGetNumDir*() results.

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

i*AccumIntvIdent* – [default=1] accumulation interval identifier for the data accumulated average values; valid values: 1 through result from *getNumAccumIntervals*() method. Specify 1 for smallest or only accumulation interval. This is dependent on those intervals defined in previous calls to the *setAccumInterval[Sec]*() method.

Returned parameters:

pvdTimes – array of ephemeris time values, in Modified Julian Date form; when using an accumulation mode other than 'Cumul', this time specifies the *ending* time of the accumulation interval.

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of ephemeris position values, in the coordinate system and units previously specified.

pvvdPitchAngles – returned array of associated pitch angles; will be empty if omni-directional.

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppGetAdiabaticCoords

(HANDLE zHandle,
double* pvvdAlpha,
double* pvvdLm,
double* pvvdK,
double* pvvdPhi,
double* pvvdHmin,
double* pvvdLstar,
double* pvdBmin,
double* pvdBlocal,
double* pvdMagLT)

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 *AppSetAdiabatic*(1) method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvvdAlpha, *pvvdLm*, *pvvdK*, *pvvdPhi*, *pvvdHmin*, *pvvdLstar* – pointers to double arrays, sized according to *AppGetChunkSize*() x *AppGetNumDir*() results.

pvdBmin, *pvdBlocal*, *pvdMagLT* – pointers to double arrays, sized according to *AppGetChunkSize*() results.

Returned parameters:

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

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

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

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

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

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

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

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

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

Return value: int – number of records returned (≥0 = success, <0 = error)

int AppGetAdiabaticCoordsPlus

(HANDLE zHandle,
double* pvdTimes,
double* pvdCoord1,
double* pvdCoord2,
double* pvdCoord3,
double* pvvdPitchAngles,
double* pvvdAlpha,
double* pvvdLm,
double* pvvdK,
double* pvvdPhi,
double* pvvdHmin,
double* pvvdLstar,
double* pvdBmin,
double* pvdBlocal,
double* pvdMagLT)

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 *AppSetAdiabatic*(1) method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

pvdTimes – pointer to double array, sized according to *AppGetChunkSize*() results

pvdCoord1, pvdCoord2, pvdCoord3 – pointers to double arrays, sized according to *AppGetChunkSize*() results

pvvdPitchAngles – pointer to double array, sized according to *AppGetChunkSize*() x *AppGetNumDir*() results.

pvvdAlpha, *pvvdLm*, *pvvdK*, *pvvdPhi*, *pvvdHmin*, *pvvdLstar* – pointers to double arrays, sized according to *AppGetChunkSize*() x *AppGetNumDir*() results.

pvdBmin, *pvdBlocal*, *pvdMagLT* – pointers to double arrays, sized according to *AppGetChunkSize*() results.

Returned parameters:

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

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of ephemeris position values, in the coordinate system and units previously specified.

pvvdPitchAngles – 2-dimensional arrray of associated pitch angles; will contain zeros if omni-directional.[time,direction]

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

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

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

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

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

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

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

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

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

Return value: int – number of records returned (≥0 = success, <0 = error)

Utility methods for your convenience

int AppResetModelData

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppResetModelRun

(HANDLE zHandle,
int iDelBinDir,
int iResetParam)

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:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iDelBinDir – flag for the deletion the temporary binary directory containing the most recent model run output files. The directory is removed if 1 *(true)*. When specified as 0 *(false),* it is kept. *Special Note*: the *AppSetDelBinDir* setting only applies to when the Application class object is destroyed when the program completes, or on subsequent calls to *AppRunModel*.

iResetParam – flag for the reset of all previously specified model run input parameters. The input parameters are reset to their initial default values if 1 *(true)*. No change if specified as 0 *(false)*.

Return value: int – 0 = success, otherwise error

int AppValidateParameters

(HANDLE zHandle)

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 *AppRunModel*() method.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success; >0: number of errors detected

int AppResetOrbitParameters

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 0 = success, otherwise error

int AppResetParameters

(HANDLE zHandle)

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

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

Return value: int – 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.

double AppGetGmtSeconds

(HANDLE zHandle,
int iHours,
int iMinutes,
double dSeconds)

Usage: Determines the GMT seconds of day for the input hours, minutes and seconds.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
iHours – hours of day (0-23)
iMinutes – minutes of hour (0-59)
dSeconds – seconds of minute (0-59.999)
Return value: double – GMT seconds of day (0-86399.999)

int AppGetDayOfYear

(HANDLE zHandle,
int iYear,
int iMonth,
int iDay)

Usage: Determines the day number of year for the input year, month and day number.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
iYear – year (1950-2049)
iMonth – month (1-12)
iDay – day of month (1-28|29|30|31)
Return value: int – day number of year (1 – 366)

double AppGetModifiedJulianDate

(HANDLE zHandle,
int iYear,
int iDdd,
double dGmtsec)

Usage: Determines the Modified Julian Date for the input year, day of year and GMT seconds.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
iYear – year (1950-2049)
iDdd – day of year (1-365|366)
dGmtsec – GMT seconds of day (0-86399.999)
Return value: double – Modified Julian Date (33282.0 - 69806.999)

double AppGetModifiedJulianDateUnix

(HANDLE zHandle,
int 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:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
iUnixTime – Unix Time, in seconds from 01 Jan 1970, 0000 GMT; (0 – MaxInt)
Return value: double – Modified Julian Date (40587.0 - 65442.134)

int AppGetDateTime

(HANDLE zHandle,
double dModJulDate,
int *piYear,
int *piDdd,
double *pdGmtsec)

Usage: Determines the year, day of year and GMT seconds for the input Modified Julian Date.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dModJulDate – Modified Julian Date (33282.0 – 69806.999)
p*iYear* – pointer to integer variable
p*iDdd* – pointer to integer variable
p*dGmtsec* – pointer to double variable
Returned parameters:
p*iYear* – year (1950-2049)
p*iDdd* – day of year (1-365|366)
p*dGmtsec* – GMT seconds of day (0-86399.999)
Return value: int – 0 = success, otherwise error

int AppGetHoursMinSec

(HANDLE zHandle,
double dGmtsec,
int* piHours,
int* piMinutes,
double* pdSeconds)

Usage: Determines the hours, minutes and seconds for the input GMT seconds.
Parameters:
zHandle – 'Application' object identifier, as returned from the AppStartUp method
dGmtsec – GMT seconds of day (0-86399.999)
piHours – *pointer* to integer variable
piMinutes – *pointer* to integer variables
pdSeconds – *pointer* to double variable
Returned parameters:
piHours – hours of day (0-23)
piMinutes – minutes of hour (0-59)
pdSeconds – seconds of minute (0-59.999)
Return value: int – 0 = success, otherwise error

int AppGetMonthDay

(HANDLE zHandle,

int iYear,
int iDdd,
int* piMonth,
int* piDay)

Usage: Determines the month and day number for the input year and day of year.

Parameters:

zHandle – 'Application' object identifier, as returned from the AppStartUp method

iYear – year (1950-2049)

iDdd – day of year (1-365|366)

piMonth – *pointer* to integer variable

piDay – *pointer* to integer variable

Returned parameters:

piMonth – month (1-12)

piDay – day of month (1-28|29|30|31)

Return value: int – 0 = success, otherwise error

Model-Level C 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.

Important: For all routines that return values via pointer arguments, the calling program is responsible for the allocation of the proper amount of memory to contain the returned values. The descriptions of these routines include recommended sizing, or specify other routines to call to determine the required sizing. The calling program is also responsible for the subsequent 'free'ing of the allocated memory.

EphemModel Class

Header file: CEphemModel_c.h

This class is the entry point that provides direct programmatic access to the ephemeris generation 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.

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 *EphemComputeEphemeris()* method may be controlled by the sizing specified using the *EphemSetChunkSize()* method. When not specified, the default behavior is to produce ephemeris for the entire period as defined in the *EphemSetTimes()* 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:

HANDLE EphemStartUp

Usage: Instantiates a new 'EphemModel' object; multiple calls to this will generate separate instances, each of which may be accessed using the unique returned handle value.
Parameters: -none-
Return value: HANDLE - identifier value for the instantiated object

EphemShutDown

(HANDLE zHandle)

Usage: Destructor
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
Return values: -none-

int EphemSetChunkSize

(HANDLE zHandle
int iChunkSize)

Usage: Specifies the number of time and position entries that are returned from each call to the *EphemComputeEphemeris*() 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 *EphemSetTimes*() methods will be calculated and returned in a *single* call to the *EphemComputeEphemeris*() 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:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
iChunkSize – number of entries in processing chunk; values lower than 60 are not recommended
Return value: int – 0 = success, otherwise error

int EphemGetChunkSize

(HANDLE zHandle)
Usage: Returns the current value of the 'chunk' size, as described in previous method.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
Return value: int – number of entries in processing chunk, (≥0 = success, <0 = error)

Model Parameter Inputs:

int EphemSetModelDBDir

(HANDLE zHandle,
char* szDataDir)

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:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
szDataDir – directory path for the IRENE database files.
Return value: int – 0 = success, otherwise error

int EphemSetMagfieldDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Specifies the name of the file for the magnetic field model database. The use of this method is *not needed* when *EphemSetModelDBDir*() 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:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szDataSource – magnetic field model database filename, including path

Return value: int – 0 = success, otherwise error

int EphemSetPropagator

(HANDLE zHandle,
char* szPropSpec)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szPropSpec - valid values: 'SatEph', 'SGP4' or 'Kepler'.

Return value: int – 0 = success, otherwise error

int EphemSetSGP4Param

(HANDLE zHandle,
char* szMode,
char* szWGS)

Usage: Specifies the mode and WGS parameter for the SGP4 orbit propagator, if being used.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szMode – SGP4 propagation mode; valid values: 'Standard' or 'Improved'.

szWGS – World Geodetic System version; valid values: '72old', '72' or '84'.

Return value: int – 0 = success, otherwise error

int EphemSetKeplerUseJ2

(HANDLE zHandle,
int iUseJ2)

Usage: Specifies the use of the 'J2' perturbation for the Kepler orbit propagator, if being used.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

iUseJ2 – 1 *(true)* or 0 *(false)*

Return value: int – 0 = success, otherwise error

---

int EphemGetModelDBDir

(HANDLE zHandle,
char* szDataDir)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szDataDir – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataDir – model database directory.

Return value: int – 0 = success, otherwise error

int EphemGetMagfieldDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Returns the name of the file for the magnetic field model database. This will be available immediately, when specified using the *EphemSetMagfieldDBFile*() method. When the *EphemSetModelDBDir*() method is used, the automatically determined filename will be available after a call to the *EphemComputeEphemeris**() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szDataSource – pointer to character string (suggested sizing = 256)

Returned parameters:

szDataSource – magnetic field model database filename.

Return value: int – 0 = success, otherwise error

int EphemGetPropagator

(HANDLE zHandle,
char* szPropSpec)

Usage: Returns the orbit propagator algorithm to use for the ephemeris generation, as specified in the *EphemSetPropagator()* method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szPropSpec – pointer to character string (suggested sizing = 32)

Returned parameters:

szPropSpec – orbit propagator.

Return value: int – 0 = success, otherwise error

int EphemGetSGP4Mode

(HANDLE zHandle,
char* szMode)

Usage: Returns the defined SGP4 orbit propagator mode, as specified in the *EphemSetSGP4param()* method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szMode – pointer to character string (suggested sizing = 32)

Returned parameters:

szMode – SGP4 propagation mode.

Return value: int – 0 = success, otherwise error

int EphemGetSGP4WGS

(HANDLE zHandle,
char* szWGS)

Usage: Returns the defined SGP4 orbit propagator WGS version, as specified in the *EphemSetSGP4param()* method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szWGS – pointer to character string (suggested sizing = 32)

Returned parameters:

szWGS – World Geodetic System version.

Return value: int – 0 = success, otherwise error

int EphemGetKeplerUseJ2

(HANDLE zHandle)

Usage: Returns whether the use of the 'J2' perturbation for the Kepler orbit propagator is enabled, as specified in the *EphemSetKeplerUseJ2*() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

Return value: int – 1 *(true)* or 0 *(false)*, <0 = error

---

int EphemSetTimes

(HANDLE zHandle,
double dStartTime,
double dEndTime,
double dTimeStepSecs)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

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

dTimeStepSecs – time step size, in seconds

Return value: int – 0 = success, otherwise error

int EphemSetVarTimes

(HANDLE zHandle,
double dStartTime,
double dEndTime,
double dTimeMinStepSec,
double dTimeMaxStepSec,
double dTimeRoundSec)

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:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
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: int – 0 = success, otherwise error

int EphemSetStartTime

(HANDLE zHandle,
double 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:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dStartTime – start time value, in Modified Julian Date form
Return value: int – 0 = success, otherwise error

int EphemSetEndTime

(HANDLE zHandle,
double dEndTime)

Usage: Specifies the stop time (*inclusive*), of the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dEndTime – stop time value, in Modified Julian Date form
Return value: int – 0 = success, otherwise error

int EphemSetTimeStep

(HANDLE zHandle,
double 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:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dTimeStepSec – time step size, in seconds
Return value: int – 0 = success, otherwise error

int EphemSetVarTimeStep

(HANDLE zHandle,
double dTimeMinStepSec,
double dTimeMaxStepSec,
double dTimeRoundSec)

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 highly elliptical orbits (ie eccentricity>0.25). See the User's Guide document "Orbit Ephemeris File Description" section for more information.

Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
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: int – 0 = success, otherwise error

int EphemSetTimesList

(HANDLE zHandle,
double* pvdTimes,
int iNumTimes)

Usage: Specifies a set of time values for the ephemeris information to be generated by the orbit propagator from the defined orbital element values or TLE file.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
pvdTimes – array of chronologically ordered time values, in Modified Julian Date form
iNumTimes – number of values in times array
Return value: int – 0 = success, otherwise error

---

int EphemGetTimes

(HANDLE zHandle,
double* pdStartTime,
double* pdEndTime,
double* pdTimeStepSec)

Usage: Returns the start and stop times, and fixed time step, for the ephemeris generation.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
pdStartTime, *pdEndTime, pdTimeStepSec* – pointers to double variables
Returned Parameters:
pdStartTime, *pdEndTime* – start and stop time values, in Modified Julian Date form
p*dTimeStepSec* – time steps, in seconds
Return value: int – 0 = success, otherwise error

int EphemGetVarTimes

(HANDLE zHandle,
double* pdStartTime,
double* pdEndTime,
double* pdTimeMinStepSec,
double* pdTimeMaxStepSec,
double* pdTimeRoundSec)

Usage: Returns the start and stop times, and variable time step limits, for the ephemeris generation.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
pdStartTime, *pdEndTime, pdTimeMinStepSec, pdTimeMaxStepSec, pdTimeRoundSec* – pointers to double variables
Returned Parameters:

pdStartTime, *pdEndTime* – start and stop time values, in Modified Julian Date form
p*dTimeMinStepSec* – lower limit of variable time steps, in seconds
p*dTimeMaxStepSec* – upper limit of variable time steps, in seconds
p*dTimeRoundSec* – rounding of variable time steps, in seconds
Return value: int – 0 = success, otherwise error

int EphemGetStartTime

(HANDLE zHandle,
double* pdStartTime)
Usage: Returns the start for the ephemeris generation.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
pdStartTime – pointer to double variable
Returned Parameters:
pdStartTime – start time value, in Modified Julian Date form
Return value: int – 0 = success, otherwise error

int EphemGetEndTime

(HANDLE zHandle,
double* pdEndTime)
Usage: Returns the end time for the ephemeris generation.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
pdEndTime – pointer to double variable
Returned Parameters:
pdEndTime – end time value, in Modified Julian Date form
Return value: int – 0 = success, otherwise error

int EphemGetTimeStep

(HANDLE zHandle,
double* pdTimeStepSecs)
Usage: Returns the fixed time step for the ephemeris generation.
Parameters:
zHandle – 'EphemModel object identifier, as returned from the EphemStartUp method
pdTimeStepSecs – pointer to double variable
Returned Parameters:
pdTimeStepSecs – time step size, in seconds
Return value: int – 0 = success, otherwise error

int EphemGetVarTimeStep

(HANDLE zHandle,
double* pdTimeMinStepSec,
double* pdTimeMaxStepSec,
double* pdTimeRoundSec)
Usage: Returns the variable time step limits for the ephemeris generation.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdTimeMinStepSec, pdTimeMaxStepSec, pdTimeRoundSec – pointers to double variables
Returned Parameters:
p*dTimeMinStepSec* – lower limit of variable time steps, in seconds
p*dTimeMaxStepSec* – upper limit of variable time steps, in seconds
p*dTimeRoundSec* – rounding of variable time steps, in seconds
Return value: int – 0 = success, otherwise error

int EphemGetNumTimesList

(HANDLE zHandle)

Usage: Returns the number of time entries defined for the ephemeris generation, from the specifications in *EphemSetTimes*() or *EphemSetTimesList*() methods. Note: the number of *variable* timestep times (from *EphemSetVarTimes*() specifications) is available only when the orbital parameters are sufficiently defined.

Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
Return value: int – number of ephemeris times defined; if negative, an error.

int EphemGetTimesList

(HANDLE zHandle,
double* pvdTimes)

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

Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
pvdTimes – pointer to double array, sized according to *EphemGetNumTimesList*() results.
Returned Parameters:
pvdTimes – array of time values, in Modified Julian Date form.
Return value: int - Number of values returned (≥0 = success, <0 = error)

int EphemClearTimesList

(HANDLE zHandle)

Usage: Clears the time list defined by calls to *EphemSetTimes*() or *EphemSetTimesList*() methods, or *EphemSetVarTimes*(), under certain conditions.

Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
Return value: int – 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.

int EphemSetTLEFile

(HANDLE zHandle,
char* szTLEFile)

Usage: Specifies the name of the Two-Line Element (TLE) file to use with the selected orbit propagator; this parameter is not needed if a set of orbital element values are being used instead.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
szTLEFile – path and filename of TLE file
Return value: int – 0 = success, otherwise error

int EphemGetTLEFile

(HANDLE zHandle,
char* szTLEFile)

Usage: Specifies the name of the Two-Line Element (TLE) file to use with the selected orbit propagator; this parameter is not needed if a set of orbital element values are being used instead.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
szTLEFile – pointer to character string (suggested sizing = 256)
Returned parameters:
szTLEFile – path and filename of TLE file
Return value: int – 0 = success, otherwise error

int EphemSetTLE

(HANDLE zHandle,
char* szTLELine1,
char* szTLELine2)

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:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
szTLELine1 – first line of TLE set
szTLELine2 – second line of TLE set
Return value: int – 0 = success, otherwise error

int EphemGetNumTLE

(HANDLE zHandle)

Usage: Returns the number of TLEs that were defined in calls to the EphemSetTLE() method.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
Return value: int – 0 or greater = number of defined TLEs, otherwise error

int EphemGetTLE

(HANDLE zHandle,
int iEntry,
char* szTLELine1,
char* szTLELine2)

Usage: Returns the TLE strings corresponding to the specified entry in the list of TLEs, that were defined in calls to the EphemSetTLE() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

iEntry – index of entry in list of TLEs, counting from 0.

szTLELine1 – pointer to character string (suggested sizing=88)

szTLELine2 – pointer to character string (suggested sizing=88)

Returned parameters:

szTLELine1 – first line of specified TLE set

szTLELine2 – second line of specified TLE set

Return value: int – 0 = success, otherwise error

int EphemResetTLE

(HANDLE zHandle)

Usage: Clears the list of TLEs that were defined in calls to the EphemSetTLE() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

Return value: int – 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 be specified. See the User's Guide document "Orbiter Propagation Inputs" section for more details.

int EphemSetElementTime

(HANDLE zHandle,
double dElementTime)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

dElementTime – time, in Modified Julian Date form

Return value: int – 0 = success, otherwise error

int EphemSetInclination

(HANDLE zHandle,
double dInclination)

Usage: Specifies the orbital element 'Inclination' value.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

dInclination – orbit inclination angle, in degrees (0-180)

Return value: int – 0 = success, otherwise error

int EphemSetRightAscension

(HANDLE zHandle,
double dRtAscOfAscNode)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

dRtAscOfAscNode – orbit ascending node position, in degrees (0-360)
Return value: int – 0 = success, otherwise error

int EphemSetEccentricity

(HANDLE zHandle,
double dEccentricity)
Usage: Specifies the orbital element 'Eccentricity' value.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dEccentricity – orbit eccentricity value, unitless (0-<1)
Return value: int – 0 = success, otherwise error

int EphemSetArgOfPerigee

(HANDLE zHandle,
double dArgOfPerigee)
Usage: Specifies the orbital element 'Argument of Perigee' value.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dArgOfPerigee – orbit perigee position, in degrees (0-360)
Return value: int – 0 = success, otherwise error

int EphemSetMeanAnomaly

(HANDLE zHandle,
double dMeanAnomaly)
Usage: Specifies the orbital element 'Mean Anomaly' value.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dMeanAnomaly – orbit mean anomaly value, in degrees (0-360)
Return value: int – 0 = success, otherwise error

int EphemSetMeanMotion

(HANDLE zHandle,
double dMeanMotion)
Usage: Specifies the orbital element 'Mean Motion' value.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dMeanMotion – orbit mean motion value, in units of revolutions per day (>0)
Return value: int – 0 = success, otherwise error

int EphemSetMeanMotion1stDeriv

(HANDLE zHandle,
double 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:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

dMeanMotion1stDeriv – first derivative of mean motion, in units of revs per day^2 (-10 – 10)
Return value: int – 0 = success, otherwise error

int EphemSetMeanMotion2ndDeriv

(HANDLE zHandle,
double 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:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dMeanMotion2ndDeriv – second derivative of mean motion, in units of revs per day^3 (-1 – 1)
Return value: int – 0 = success, otherwise error

int EphemSetBStar

(HANDLE zHandle,
double dBStar)
Usage: Specifies the orbital element 'B*' value, for modelling satellite drag effects; this value is only used by the SGP4 propagator.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dBStar – ballistic coefficient value (-1 – 1)
Return value: int – 0 = success, otherwise error

int EphemSetAltitudeOfApogee

(HANDLE zHandle,
double dAltApogee)
Usage: Specifies the orbital element 'Apogee Altitude' value (furthest distance).
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dAltApogee – altitude (in km) above the Earth's surface at the orbit's apogee (>0 – 75 Re)
Return value: int – 0 = success, otherwise error

int EphemSetAltitudeOfPerigee

(HANDLE zHandle,
double dAltPerigee)
Usage: Specifies the orbital element 'Perigee Altitude' value (closest distance).
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dAltPerigee – altitude (in km) above the Earth's surface at the orbit's perigee (>0 – 75 Re)
Return value: int – 0 = success, otherwise error

int EphemSetLocalTimeOfApogee

(HANDLE zHandle,
double dLocTimeApogee)
Usage: Specifies the local time of the orbit's apogee.
Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dLocTimeApogee – local time, in hours (0-24)
Return value: int – 0 = success, otherwise error

int EphemSetLocalTimeMaxInclination

(HANDLE zHandle,
double dLocTimeMaxIncl)
Usage: Specifies the local time of the orbit's maximum inclination (ie max latitude).
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dLocTimeMaxIncl – local time, in hours (0-24)
Return value: int – 0 = success, otherwise error

int EphemSetTimeOfPerigee

(HANDLE zHandle,
double 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:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dTimeOfPerigee – time, in Modified Julian Date form, for orbit perigee
Return value: int – 0 = success, otherwise error

int EphemSetSemiMajorAxis

(HANDLE zHandle,
double dSemiMajorAxis)
Usage: Specifies the orbit's semi-major axis length.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dSemiMajorAxis – semi-major axis length (1 – 75), in units of Re (radius of Earth = 6371.2 km)
Return value: int – 0 = success, otherwise error

int EphemSetGeosynchLon

(HANDLE zHandle,
double dGeosynchLon)
Usage: Specifies the geographic longitude of satellite in a geosynchronous orbit
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dGeosynchLon – longitude, in degrees (-180 – 360)
Return value: int – 0 = success, otherwise error

int EphemSetStateVectors

(HANDLE zHandle,
double* pvdPos,
double* pvdVel)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pv*dPos* – array containing the GEI coordinate system satellite position values (X,Y,Z), in km

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

Return value: int – 0 = success, otherwise error

int EphemSetPositionGEI

(HANDLE zHandle,
double dPosX,
double dPosY,
double 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 *EphemSetVelocityGEI*() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

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

Return value: int – 0 = success, otherwise error

int EphemSetVelocityGEI

(HANDLE zHandle,
double dVelX,
double dVelY,
double 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 *EphemSetPositionGEI*() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

dVelX, dVelY, dVelZ– GEI coordinate system satellite velocity values, in km/sec (>0 – 20)

Return value: int – 0 = success, otherwise error

---

int EphemGetElementTime

(HANDLE zHandle,
double pdElementTime*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdElementTime – pointer to double variable

Returned parameter:

pdElementTime – time, in Modified Julian Date form.

Return value: int – 0 = success, otherwise error

int EphemGetInclination

(HANDLE zHandle,
double pdInclination*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdInclination – pointer to double variable

Returned parameter:

pdInclination – orbit inclination angle, in degrees.

Return value: int – 0 = success, otherwise error

int EphemGetRightAscension

(HANDLE zHandle,
double pdRtAscOfAscNode*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdRtAscOfAscNode – pointer to double variable

Returned parameter:

pdRtAscOfAscNode – orbit ascending node position, in degrees.

Return value: int – 0 = success, otherwise error

int EphemGetEccentricity

(HANDLE zHandle,
double pdEccentricity*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdEccentricity – pointer to double variable

Returned parameter:

pdEccentricity – orbit eccentricity value (unitless).

Return value: int – 0 = success, otherwise error

int EphemGetArgOfPerigee

(HANDLE zHandle,
double pdArgOfPerigee*)

Usage: Returned the defined orbital element 'Argument of Perigee' value, as specified in the *EphemSetArgOfPerigee*() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdArgOfPerigee – pointer to double variable

Returned parameter:

pdArgOfPerigee – orbit perigee position, in degrees.

Return value: int – 0 = success, otherwise error

int EphemGetMeanAnomaly

(HANDLE zHandle,
double pdMeanAnomaly*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdMeanAnomaly – pointer to double variable

Returned parameter:

pdMeanAnomaly – orbit mean anomaly value, in degrees

Return value: int – 0 = success, otherwise error

int EphemGetMeanMotion

(HANDLE zHandle,
double pdMeanMotion*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdMeanMotion – pointer to double variable

Returned parameter:

pdMeanMotion – orbit mean motion value, in units of revolutions per day.

Return value: int – 0 = success, otherwise error

Int EphemGetMeanMotion1stDeriv

(HANDLE zHandle,
double pdMeanMotion1stDeriv*)

Usage: Returns the defined orbital element 'First Time Derivative of the Mean Motion' value, as specified in the *EphemSetMeanMotion1stDeriv*() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdMeanMotion1stDeriv – pointer to double variable

Returned parameter:

pdMeanMotion1stDeriv – first derivative of mean motion, in units of revs per day^2.

Return value: int – 0 = success, otherwise error

int EphemGetMeanMotion2ndDeriv

(HANDLE zHandle,
double pdMeanMotion2ndDeriv*)

Usage: Returns the defined orbital element 'Second Time Derivative of the Mean Motion' value, as specified in the *EphemSetMeanMotion2ndDeriv*() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdMeanMotion2ndDeriv – pointer to double variable

Returned parameter:

pdMeanMotion2ndDeriv – second derivative of mean motion, in units of revs per day^3.

Return value: int – 0 = success, otherwise error

int EphemGetBStar

(HANDLE zHandle,
double pdBStar*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdBStar – pointer to double variable

Returned parameter:

pdBStar – ballistic coefficient value.

Return value: int – 0 = success, otherwise error

int EphemGetAltitudeOfApogee

(HANDLE zHandle,
double pdAltApogee*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdAltApogee – pointer to double variable

Returned parameter:

pdAltApogee – altitude (in km) above the Earth's surface at the orbit's apogee.

Return value: int – 0 = success, otherwise error

int EphemGetAltitudeOfPerigee

(HANDLE zHandle,
double pdAltPerigee*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdAltPerigee – pointer to double variable

Returned parameter:

pdAltPerigee – altitude (in km) above the Earth's surface at the orbit's perigee.

Return value: int – 0 = success, otherwise error

int EphemGetLocalTimeOfApogee

(HANDLE zHandle,
double pdLocTimeApogee*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdLocTimeApogee – pointer to double variable

Returned parameter:

pdLocTimeApogee – local time, in hours + fraction.

Return value: int – 0 = success, otherwise error

int EphemGetLocalTimeMaxInclination

(HANDLE zHandle,
double pdLocTimeMaxIncl*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdLocTimeMaxIncl – pointer to double variable

Returned parameter:

pdLocTimeMaxIncl – local time, in hours + fraction.

Return value: int – 0 = success, otherwise error

int EphemGetTimeOfPerigee

(HANDLE zHandle,
double pdTimeOfPerigee*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdTimeOfPerigee – pointer to double variable

Returned parameter:

pdTimeOfPerigee – time of perigee, in Modified Julian Date form.

Return value: int – 0 = success, otherwise error

int EphemGetSemiMajorAxis

(HANDLE zHandle,
double pdSemiMajorAxis*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdSemiMajorAxis – pointer to double variable

Returned parameter:

pdSemiMajorAxis – semi-major axis length, in units of Re.

Return value: int – 0 = success, otherwise error

int EphemGetGeosynchLon

(HANDLE zHandle,
double pdGeosynchLon*)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdGeosynchLon – pointer to double variable

Returned parameter:

pdGeosynchLon – longitude, in degrees.

Return value: int – 0 = success, otherwise error

int EphemGetStateVectors

(HANDLE zHandle,
double* pvdPos,
double* pvdVel)

Usage: Returns the satellite's position and velocity vector values, as specified in the *EphemSetStateVectors*() method, or in the *EphemSetPositionGEI*() and *EphemSetVelocityGEI*() methods.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
pvdPos – pointer to double array, size=3
pvdVel – pointer to double array, size=3

Returned Parameters:

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

Return value: int – 0 = success, otherwise error

int EphemGetPositionGEI

(HANDLE zHandle,
double* pdX,
double* pdY,
double* pdZ)

Usage: Returns the satellite's position vector, as specified in either the *EphemSetPositionGEI*() or *EphemSetStateVectors*() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
pdPosX, pdPosY, pdPosZ – pointers to double variables

Returned Parameters:

pdPosX, pdPosY, pdPosZ – GEI coordinate system satellite position values, in km

Return value: int – 0 = success, otherwise error

int EphemGetVelocityGEI

(HANDLE zHandle,
double* pdVelX,
double* pdVelY,
double* pdVelZ)

Usage: Returns the satellite's velocity vector, as specified in either the *EphemSetVelocityGEI*() or *EphemSetStateVectors*() method.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
pdVelX, pdVelY, pdVelZ – pointers to double variables

Return value: int – 0 = success, otherwise error

int EphemResetOrbitParameters

(HANDLE zHandle)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

Return value: int – 0 = success, otherwise error

int EphemSetMainField

(HANDLE zHandle,
char* szMainField)

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:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

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

Return value: int – 0 = success, otherwise error

int EphemSetExternalField

(HANDLE zHandle,
char* szExternalField)

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 *EphemSetKpValue*() or *EphemSetKpValues*() methods.

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

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

Return value: int – 0 = success, otherwise error

int EphemGetMainField

(HANDLE zHandle,
char* szMainField)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szMainField– pointer to character string (suggested sizing = 16)

Returned parameter:

szMainField – 'main' magnetic field model identifier.

Return value: int – 0 = success, otherwise error

int EphemGetExternalField

(HANDLE zHandle,
char* szExternalField)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szExternalField– pointer to character string (suggested sizing = 16)
Returned parameter:
szExternalField – 'external' magnetic field model identifier.
Return value: int – 0 = success, otherwise error

int EphemSetKpValue

(HANDLE zHandle,
double 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 *EphemSetKpValues*() method is cleared.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dKpVal – Kp Index value. Valid range = 0.0 – 9.0.
Return value: int – 0 = success, otherwise error

int EphemSetKpValues

(HANDLE zHandle,
double dRefTime,
double* pvdKpVals,
int iNumEntries)

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 *EphemSetKpValue*() method.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
dRefTime – reference time value, in Modified Julian Date form; must be at 0000 GMT of day.
pv*dKpVals* – array of time history of three-hour Kp index values. Valid range = 0.0 – 9.0.
iNumEntries – number of Kp values in array
Return value: int – 0 = success, otherwise error

int EphemGetKpValue

(HANDLE zHandle,
double* pdKpVal)

Usage: Returns the current defined Kp value. This may be the constant value specified in the *EphemSetKpValue*() method, or the appropriate entry from the time history value defined in the *EphemSetKpValues*() method, according to the most recently used calculation time value.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
pdKpVal – pointer to double variable
Returned parameter:
pdKpVal – current Kp index value.
Return value: int – 0 = success, otherwise error

int EphemGetKpValuesRefTime

(HANDLE zHandle,
double* pdRefTime)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdRefTime – pointer to double variable

Returned parameter:

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

Return value: int – 0 = success, otherwise error

int EphemGetKpValuesEndTime

(HANDLE zHandle,
double* pdEndTime)

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pdEndTime – pointer to double variable

Returned parameter:

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

Return value: int – 0 = success, otherwise error

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.

int EphemComputeEphemerisGEI

(HANDLE zHandle,
double* pvdTimes,
double* pvdXGEI,
double* pvdYGEI,
double* pvdZGEI,
double* pvdXDotGEI,
double* pvdYDotGEI,
double* pvdZDotGEI)

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 *EphemComputeEphemeris*() method is specified using the *EphemSetChunkSize*() method. If this 'chunk' size is not specified, or set to '0', the ephemeris for the entire time period, defined by the *EphemSetTimes*(), *EphemSetVarTimes*() or *EphemSetTimesList*() methods, is returned in a single call; in this case, the sizing of the return parameters would be dependent on the specifications used in those methods. If a great number of ephemeris entries are requested in a single call, the amount of memory required 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:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pvdTimes, pvdXGEI, pvdYGEI, pvdZGEI, pvdXDotGEI, pvdYDotGEI, pvdZDotGEI – pointers to double arrays, sized according to *EphemGetChunkSize*() results

Returned parameters:

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

pvdXGEI, pvdYGEI, pvdZGEI – arrays of ephemeris position values, in the 'GEI' coordinate system and units of 'km'

pvdXDotGEI, pvdYDotGEI, pvdZDotGEI – arrays of ephemeris velocity values, in the 'GEI' coordinate system and units of 'km/sec'

Return value: int – Number of ephemeris records returned (≥0 = success, <0 = error)

int EphemComputeEphemeris

```
( HANDLE zHandle,
  char* szCoordSys,
  char* szCoordUnits,
  double* pvdTimes,
  double* pvdCoord1,
  double* pvdCoord2,
  double* pvdCoord3 )
```

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 *EphemComputeEphemeris*() method is specified using the *EphemSetChunkSize*() method. If this 'chunk' size is not specified, or set to '0', the ephemeris for the entire time period, defined by the *EphemSetTimes*(), *EphemSetVarTimes*() or *EphemSetTimesList* () methods, is returned in a single call; in this case, the sizing of the return parameters would be dependent on the specifications used in those methods. If a great number of ephemeris entries are requested in a single call, the amount of memory required 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:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

pvdTimes – pointer to double array, sized according to *EphemGetChunkSize*() results

szCoordSys – 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.

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

pvdTimes, pvdCoord1, pvdCoord2, pvdCoord3 – pointers to double arrays, sized according to *EphemGetChunkSize*() results

Returned parameters:

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

pvdCoord1, pvdCoord2, pvdCoord3 – 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.
Return value: int – Number of ephemeris records returned (≥0 = success, <0 = error)

int EphemRestartEphemeris

(HANDLE zHandle)

Usage: When the 'chunk' size, specified using the *EphemSetChunkSize*() method, is larger than 0, this method explicitly resets the *EphemComputeEphemeris*() 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.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
Return value: int – 0 = success, otherwise error

int EphemConvertCoordinates

(HANDLE zHandle,
char* szCoordSys,
char* szCoordUnits,
double* pvdTimes,
double* pvdCoord1,
double* pvdCoord2,
double* pvdCoord3,
int iNumTimes,
char* szNewCoordSys,
char* szNewCoordUnits,
double* pvdNewCoord1,
double* pvdNewCoord2,
double* pvdNewCoord3)

Usage: Converts the set of input times, position coordinates from one coordinate system to another.
Parameters:
zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method
szCoordSys – 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.
szCoordUnits – 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.
pvdTimes – array of time values, in Modified Julian Date form
pvdCoord1, pvdCoord2, pvdCoord3 – arrays of position values, in the coordinate system and units specified by the *szCoordSys* and *szCoordUnit* 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.
iNumTimes – number of values in times array and coord arrays
szNewCoordSys – '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.
szNewCoordUnits – 'new' units: 'km' or 'Re'; 'GDZ' is set to always use 'km' for the altitude value.

pvdNewCoord1, pvdNewCoord2, pvdNewCoord3 – pointers to double arrays, of size *iNumTimes*

Returned parameters:

pvdNewCoord1, pvdNewCoord2, pvdNewCoord3 – arrays of position values, in the 'new' coordinate system and units specified by the *szNewCoordSys* and *szNewCoordUnit* parameter values.

Return value: int – 0 = success, otherwise error

int EphemConvertCoordinatesSingle

```
( HANDLE zHandle,
  char* szCoordSys,
  char* szCoordUnits,
  double dTime,
  double dCoord1,
  double dCoord2,
  double dCoord3,
  char* szNewCoordSys,
  char* szNewCoordUnits,
  double* pdNewCoord1,
  double* pdNewCoord2,
  double* pdNewCoord3 )
```

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

Parameters:

zHandle – 'EphemModel' object identifier, as returned from the EphemStartUp method

szCoordSys – 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.

szCoordUnits – '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 *szCoordSys* and *szCoordUnit* 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.

szNewCoordSys – '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.

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

pdNewCoord1, pdNewCoord2, pdNewCoord3 – pointers to double variables

Returned parameters:

pdNewCoord1, pdNewCoord2, pdNewCoord3 – position values, in the 'new' coordinate system and units specified by the *szNewCoordSys* and *szNewCoordUnit* parameter values.

Return value: int – 0 = success, otherwise error

```
int EphemComputeBfield
        ( HANDLE zHandle,
          char* szCoordSys,
          char* szCoordUnits,
          double* pvdTimes,
          double* pvdCoord1,
          double* pvdCoord2,
```

double* pvdCoord3,
int iNumTimes,
double* pvvBVecGeo,
double* pvdBMag,
double* pvdBMin,
double* pvdLm)

Usage: Calculates several magnetic field parameters for the specified time and position inputs. Results will be affected by specifications for the 'main' and 'external' magnetic field model, using the *EphemSetMainField*() and *EphemSetExternalField*() methods, respectively. When the 'Tsyg89' external field model is being used, an additional specification of the Kp index value will be required, using the *EphemSetKpValue*() or *EphemSetKpValues*() 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:

szCoordSys – 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.

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

pvdTimes – array of time values, in Modified Julian Date form

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of position values, in the coordinate system and units specified by the *szCoordSys* and *szCoordUnit* 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.

iNumTimes – number of values in times array and coord arrays

pvvBVecGeo – pointer to double array, of size *iNumTimes* × 3

pvdBMag – pointer to double array, of size *iNumTimes*

pvdBMin – pointer to double array, of size *iNumTimes*

pvdLm – pointer to double array, of size *iNumTimes*

Returned parameters:

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

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

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

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

Return value: int – 0 = success, otherwise error

int EphemComputeBfieldSingle
(HANDLE zHandle,
char* szCoordSys,
char* szCoordUnits,
double dTime,
double dCoord1,
double dCoord2,
double dCoord3,
double* pvdBVecGeo,
double* pdBMag,

double* pdBMin,
double* pdLm)

Usage: Calculates several magnetic field parameters for the specified time and position input. Results will be affected by specifications for the 'main' and 'external' magnetic field model, using the *EphemSetMainField*() and *EphemSetExternalField*() methods, respectively. When the 'Tsyg89' external field model is being used, an additional specification of the Kp index value will be required, using the *EphemSetKpValue*() or *EphemSetKpValues*() 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:

szCoordSys – 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.

szCoordUnits – '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 *szCoordSys* and *szCoordUnit* 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.

pvdBVecGeo – pointer to double array, of size 3

pdBMag – pointer to double variable

pdBMin – pointer to double variable

pdLm – pointer to double variable

Returned parameters:

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

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

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

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

Return value: int – 0 = success, otherwise error

int EphemComputeInvariants
(HANDLE zHandle,
char* szCoordSys,
char* szCoordUnits,
double* pvdTimes,
double* pvdCoord1,
double* pvdCoord2,
double* pvdCoord3,
int iNumTimes,
double* pvdPitchAngles,
int iNumPitchAngles,
double* pvdBMin,
double* pvvdBMinPosGeo,
double* pvvdBVecGeo,
double* pvvdLm,

double* pvvdI)

Usage: Calculates magnetic field parameters as a function of pitch angle for the specified time and position inputs. Results will be affected by specifications for the 'main' and 'external' magnetic field model, using the *EphemSetMainField*() and *EphemSetExternalField*() methods, respectively. When the 'Tsyg89' external field model is being used, an additional specification of the Kp index value will be required, using the *EphemSetKpValue*() or *EphemSetKpValues*() 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:

szCoordSys – 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.

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

pvdTimes – array of time values, in Modified Julian Date form

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of position values, in the coordinate system and units specified by the *szCoordSys* and *szCoordUnit* 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.

iNumTimes – number of values in times array and coord arrays

pvdPitchAngles – array of pitch angle values, in degrees (0-90). Must be in *descending* order.

iNumPitchAngles – number of values in pitch angle array

pvdBMin – pointer to double array, of size *iNumTimes*

pvvBMinPosGeo – pointer to double array, of size *iNumTimes* × 3

pvvBVecGeo – pointer to double array, of size *iNumTimes* × 3

pvvdLm – pointer to double array, of size *iNumTimes* × *iNumPitchAngles*

pvvdI – pointer to double array, of size *iNumTimes* × *iNumPitchAngles*

Returned parameters:

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

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

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

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

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

Return value: int – 0 = success, otherwise error

```
int EphemComputeInvariantsSingle
        ( HANDLE zHandle,
          char* szCoordSys,
          char* szCoordUnits,
          double dTime,
          double dCoord1,
          double dCoord2,
          double dCoord3,
```

double* pvdPitchAngles,
int iNumPitchAngles,
double* pdBMin,
double* pvdBMinPosGeo,
double* pvdBVecGeo,
double* pvdLm,
double* pvdI)

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 *EphemSetMainField*() and *EphemSetExternalField*() methods, respectively. When the 'Tsyg89' external field model is being used, an additional specification of the Kp index value will be required, using the *EphemSetKpValue*() or *EphemSetKpValues*() 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:

szCoordSys – 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.

szCoordUnits – '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 *szCoordSys* and *szCoordUnit* 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.

pvdPitchAngles – array of pitch angle values, in degrees (0-90). Must be in *descending* order.

iNumPitchAngles – number of values in pitch angle array

pdBMin – pointer to double variable

pvdBMinPosGeo – pointer to double array, of size 3

pvdBVecGeo – pointer to double array, of size 3

pvdLm – pointer to double array, of size *iNumPitchAngles*

pvdI – pointer to double array, of size *iNumPitchAngles*

Returned parameters:

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

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

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

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

pvdI – array of "I" values [unitless], associated with the time+position specified, for each of the pitch angles specified.

Return value: int – 0 = success, otherwise error

Ae9Ap9Model Class

Header file: Ae9Ap9Model_c.h

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:

HANDLE Ae9Ap9StartUp

Usage: Instantiates a new 'Ae9Ap9Model' object; multiple calls to this will generate separate instances, each of which may be accessed using the unique returned handle value.
Parameters: -none-
Return value: HANDLE - identifier value for the instantiated object

Ae9Ap9ShutDown

(HANDLE zHandle)
Usage: Destructor
Parameters:
zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method
Return values: -none-

Model Parameter Inputs:

int Ae9Ap9SetModel

(HANDLE zHandle,
char* szModele)
Usage: Specifies the name of the flux model to be used in the calculations. Note the 'Plasma' model names now includes the species type.
Parameters:
zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method
szModel – model name: 'AE9', 'AP9', 'PlasmaE', 'PlasmaH', 'PlasmaHe' or 'PlasmaO'
Return value: int – 0 = success, otherwise error

int Ae9Ap9SetModelDBDir

(HANDLE zHandle,
char* szDataDir)
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:

zHandle – 'Ae9Ap9' object identifier, as returned from the Ae9Ap9StartUp method
szDataDir – directory path for the IRENE database files.
Return value: int – 0 = success, otherwise error

int Ae9Ap9SetModelDBFile

(HANDLE zHandle,
char* szModelDBFile)

Usage: Specifies the name of the database file for flux model calculations. The use of this method is *not needed* when *Ae9Ap9SetModelDBDir*() 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 *Ae9Ap9LoadModelDB*() or *Ae9Ap9SetFluxEnvironment*() methods, the specified model database ***cannot*** be changed. For best results, use separate model objects for different model species.

Parameters:
zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method
szModelDBFile – model database filename, including path
Return value: int – 0 = success, otherwise error

int Ae9Ap9SetKPhiDBFile

(HANDLE zHandle,
char* szKPhiDBFile)

Usage: Specifies the name of the file for the K/Phi database. The use of this method is *not needed* when *Ae9Ap9SetModelDBDir*() 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:
zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method
szKPhiDBFile – database filename, including path
Return value: int – 0 = success, otherwise error

int Ae9Ap9SetKHMinDBFile

(HANDLE zHandle,
char* szKHMinDBFile)

Usage: Specifies the name of the file for the K/Hmin database. The use of this method is *not needed* when *Ae9Ap9SetModelDBDir*() 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:
zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method
szKHMinDBFile – database filename, including path
Return value: int – 0 = success, otherwise error

int Ae9Ap9SetMagfieldDBFile

(HANDLE zHandle,
char* szMagfieldDBFile)

Usage: Specifies the name of the file for the magnetic field model database. The use of this method is *not needed* when *Ae9Ap9SetModelDBDir*() 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:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

szMagfieldDBFile – magnetic field model database filename, including path

Return value: int – 0 = success, otherwise error

int Ae9Ap9LoadModelDB

(HANDLE zHandle)

Usage: Performs the initial loading of information from the model database file specified in the *Ae9Ap9SetModelDBFile*() method. Use of this method is optional, as it is called internally on the initial call to the *Ae9Ap9SetFluxEnvironment*() method. However, it is required if the *Ae9Ap9GetModel[Name|Species]*() methods are called before the initial call to *Ae9Ap9SetFluxEnvironment*().

Once initialized, these model databases specifications ***cannot*** be changed.

Parameters:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

Return value: int – 0 = success, otherwise error

---

int Ae9Ap9GetModelName

(HANDLE zHandle,
char* szModelName)

Usage: Returns the name of the model described by the model database file specified in the previous call to *Ae9Ap9SetModelDBFile*() method. This returned model name is available only after a call to either the *Ae9Ap9LoadModelDB*() or *Ae9Ap9SetFluxEnvironment*() methods .

Parameters:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

szModelName – pointer to character string (suggested sizing = 8)

Returned parameter:

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

Return value: int – 0 = success, otherwise error

int Ae9Ap9GetModelSpecies

(HANDLE zHandle,
char* szModelSpecies)

Usage: Returns the name of the particle species of the model database file specified in the previous call to *Ae9Ap9SetModelDBFile*() method. This returned species name is available only after a call to either the *Ae9Ap9LoadModelDB*() or *Ae9Ap9SetFluxEnvironment*() methods .

Parameters:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

szModelSpecies – pointer to character string (suggested sizing = 8)

Returned parameter:

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

Return vaue: int – 0 = success, otherwise error

int Ae9Ap9GetModel

(HANDLE zHandle,
char* szModel)

Usage: Returns the name of the flux model, as specified in the *Ae9Ap9SetModel*() method.

Parameters:

zHandle – 'Ae9Ap9' object identifier, as returned from the Ae9Ap9StartUp method

szModel – pointer to character string (suggested sizing = 16)

Returned parameter:

szModel – model name.

Return value: int – 0 = success, otherwise error

int Ae9Ap9GetModelDBDir

(HANDLE zHandle,
char* szDataDir)

Usage: Returns the directory name containing the collection of IRENE model database files that was specified in a previous call to the *Ae9Ap9SetModelDBDir*() method; otherwise, blank.

Parameters:

zHandle – 'Ae9Ap9' object identifier, as returned from the Ae9Ap9StartUp method

szDataDir – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataDir – model database directory.

Return value: int – 0 = success, otherwise error

int Ae9Ap9GetModelDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Returns the name of the database file for flux model calculations. This will be available immediately, when specified using the *Ae9Ap9SetModelDBFile*() method. When the *Ae9Ap9SetModelDBDir*() method is used, the automatically determined filename will be available after a call to either the *Ae9Ap9LoadModelDB*() or *Ae9Ap9SetFluxEnvironment**() methods.

Parameters:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

szDataSource – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataSource – model database filename.

Return value: int – 0 = success, otherwise error

int Ae9Ap9GetKPhiDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Returns the name of the file for the K/Phi database. This will be available immediately, when specified using the *Ae9Ap9SetKPhiDBFile*() method. When the *Ae9Ap9SetModelDBDir*() method is used, the automatically determined filename will be available after a call to either the *Ae9Ap9LoadModelDB*() or *Ae9Ap9SetFluxEnvironment**() methods.

Parameters:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

szDataSource – pointer to character string (suggested sizing = 256)

Returned parameter:
szDataSource – K/Phi database filename.
Return value: int – 0 = success, otherwise error

int Ae9Ap9GetKHMinDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Returns the name of the file for the K/Hmin database. This will be available immediately, when specified using the *Ae9Ap9SetKHMinDBFile*() method. When the *Ae9Ap9SetModelDBDir*() method is used, the automatically determined filename will be available after a call to either the *Ae9Ap9LoadModelDB*() or *Ae9Ap9SetFluxEnvironment**() methods.
Parameters:
zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method
szDataSource – pointer to character string (suggested sizing = 256)
Returned parameter:
szDataSource – K/Hmin database filename.
Return value: int – 0 = success, otherwise error

int Ae9Ap9GetMagfieldDBFile

(HANDLE zHandle,
char* szDataSource)

Usage: Returns the name of the file for the magnetic field model database. This will be available immediately, when specified using the *Ae9Ap9SetMagfieldDBFile*() method. When the *Ae9Ap9SetModelDBDir*() method is used, the automatically determined filename will be available after a call to either the *Ae9Ap9LoadModelDB*() or *Ae9Ap9SetFluxEnvironment**() methods.
Parameters:
zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method
szDataSource – pointer to character string (suggested sizing = 256)
Returned parameter:
szDataSource – magnetic field model database filename.
Return value: int – 0 = success, otherwise error

Model Execution and Results:

The *Ae9Ap9SetFluxEnvironment**() methods are used to specify the ephemeris (time and position) and particle energies for the flux calculation of the Ae9Ap9 model. The various versions of this method provide different ways to define the flux particle direction(s) – omnidirection, fixed (over time) or variable pitch angles, or explicit direction vectors. The various *Ae9Ap9Flyin**() (or *Ae9Ap9ComputeFlux**()) methods return the respective types of flux values using the most recently defined 'flux environment' specifications.
The returned flux values are in units of [#/cm2/sec] (for integral) or [#/cm2/sec/MeV] (for differential).

For best model performance, it is recommended that the amount of ephemeris information being supplied as input to the *Ae9Ap9SetFluxEnvironment**() methods be moderated. These methods perform 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 *Ae9Ap9Flyin**() method(s) will return fluxes for the current ephemeris segment.

The ephemeris information produced by the *EphemComputeEphemeris*() can be segmented through the use of the associated *EphemSetChunkSize*() method. The segmentation of ephemeris information from other sources will need to be accomplished by the calling process.

int Ae9Ap9SetFluxEnvironmentOmni

(HANDLE zHandle,
char* szFluxType,
double* pvdEnergies,
double* pvdEnergies2,
double* pvdTimes,
char* szCoordSys,
char* szCoordUnits,
double* pvdCoords1,
double* pvdCoords2,
double* pvdCoords3
int iNumTimes,
int iNumEnergies)

Usage: Specifies the flux type, energies [MeV] 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:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

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

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

pvdEnergies2 – ignored unless flux type is '2ptDiff'; array of energy values [MeV], specifying upper bounds of energy bins

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

szCoordSys – 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.

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

pvdCoords1, *pvdCoords2*, *pvdCoords3* – 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 szCoordSys and szCoordUnits 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.

iNumTimes – number of times in time array

iNumEnergies – number of energies in energy array

Return value: int – 0 = success, otherwise error

int Ae9Ap9SetFluxEnvironmentFixPitch

(HANDLE zHandle,
char* szFluxType,
double* pvdEnergies,
double* pvdEnergies2,
double* pvdTimes,
char* szCoordSys,
char* szCoordUnits,
double* pvdCoords1,
double* pvdCoords2,
double* pvdCoords3,
double* pvdPitchAngles,
int iNumTimes,
int iNumEnergies,
int iNumDir)

Usage: Specifies the flux type, energies [MeV] 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:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

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

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

pvdEnergies2 – ignored unless flux type is '2ptDiff'; array of energy values [MeV], specifying upper bounds of energy bins

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

szCoordSys – 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.

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

pvdCoords1, *pvdCoords2*, *pvdCoords3* – 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 szCoordSys and szCoordUnits 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.

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

iNumTimes – number of times in time array

iNumEnergies – number of energies in energy array

iNumDir – number of pitch angles in directions array

Return value: int – 0 = success, otherwise error

int Ae9Ap9SetFluxEnvironmentVarPitch

(HANDLE zHandle,
char* szFluxType,
double* pvdEnergies,
double* pvdEnergies2,

double* pvdTimes,
char* szCoordSys,
char* szCoordUnits,
double* pvdCoords1,
double* pvdCoords2,
double* pvdCoords3,
double* pvdFluxDir1,
double* pvdFluxDir2,
double* pvdFluxDir3
int iNumTimes,
int iNumEnergies,
int iNumDir)

Usage: Specifies the flux type, energies [MeV] 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:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

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

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

pvdEnergies2 – ignored unless flux type is '2ptDiff'; array of energy values [MeV], specifying upper bounds of energy bins

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

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

szCoordUnits – 'km' or 'Re'

pvdCoords1, *pvdCoords2*, *pvdCoords3* – 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 szCoordSys and szCoordUnits parameters. Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

pvdFluxDir1, *pvdFluxDir2*, *pvdFluxDir3* – arrays of direction array values associated with times and position arrays. These direction values must be in the same Cartesian coordinate system and units as the position arrays.

iNumTimes – number of times in time array

iNumEnergies – number of energies in energy array

iNumDir – number of pitch angles in directions array

Return value: int – 0 = success, otherwise error

int Ae9Ap9SetFluxEnvironmentDirVec

(HANDLE zHandle,
char* szFluxType,
double* pvdEnergies,
double* pvdEnergies2,
double* pvdTimes,
char* szCoordSys,

char* szCoordUnits,
double* pvdCoords1,
double* pvdCoords2,
double* pvdCoords3,
double* pvvdPitchAngles
int iNumTimes,
int iNumEnergies,
int iNumDir)

Usage: Specifies the flux type, energies [MeV] 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:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

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

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

pvdEnergies2 – ignored unless flux type is '2ptDiff'; array of energy values [MeV], specifying upper bounds of energy bins

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

szCoordSys – 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.

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

pvdCoords1, *pvdCoords2*, *pvdCoords3* – arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system specified by the szCoordSys and szCoordUnits parameters. This may be a unit vector. Please consult the User's Guide document, "Supported Coordinate Systems" for more details.

pv*vdPitchAngles* – 2-dimensional array of pitch angles, in degrees (0-180). [time,pitch angles]

iNumTimes – number of times in time array

iNumEnergies – number of energies in energy array

iNumDir – number of pitch angles in directions array

Return value: int – 0 = success, otherwise error

int Ae9Ap9GetPitchAngles

(HANDLE zHandle,
double* pvvdPitchAngles)

Usage: Returns the set of uni-directional pitch angles used in the model calculations, corresponding to the direction arrays input to the fourth form of the *Ae9Ap9SetFluxEnvironment*() method.

Parameters:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

pv*vdPitchAngles* – 2-dimensional array of pitch angles, in degrees (0-180). [time,direction]

Return value: int – 0 = success, otherwise error

int Ae9Ap9FlyinMean or ***Ae9Ap9ComputeFluxMean***

(HANDLE zHandle,
double* pvvvdFluxData)

Usage: Returns the 'Mean' model flux at the times, positions, energies (and possibly directions) specified in the most recent call to the *Ae9Ap9SetFluxEnvironment*() method.
This same method may also be called as *AppComputeFluxMean*(), with same exact arguments.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

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

Return value: int – 0 = success, otherwise error

int Ae9Ap9FlyinMean

(HANDLE zHandle,
double* pvvdFluxData)

Usage: Returns the 'Mean' model flux at the times, positions, energies specified in the most recent call to the *Ae9Ap9SetFluxEnvironment*() method. This method may only be used when calculating *omnidirectional* fluxes.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

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

Return value: int – 0 = success, otherwise error

int Ae9Ap9FlyinPercentile or ***Ae9Ap9ComputeFluxPercentile***

(HANDLE zHandle,
int iPercentile,
double* pvvvdFluxData)

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 the *Ae9Ap9SetFluxEnvironment*() method.
This same method may also be called as *AppComputeFluxPercentile*(), with same exact arguments.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method

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

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

Return value: int – 0 = success, otherwise error

int Ae9Ap9FlyinPerturbedMean or ***Ae9Ap9ComputeFluxPerturbedMean***

(HANDLE zHandle,
int iScenario,
double* pvvvdFluxData)

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 the *Ae9Ap9SetFluxEnvironment*() method.
This same method may also be called as *AppComputeFluxPerturbedMean*(), with same exact arguments.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).

Parameters:

zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method
iScenario – scenario number of the Perturbed Mean flux values to be returned.
pvvvdFluxData – returned 3-dimensional array of the flux values for the specified scenario number. [time,energy,direction]
Return value: int – 0 = success, otherwise error

int Ae9Ap9FlyinScenario or ***Ae9Ap9ComputeFluxScenario***

(HANDLE zHandle,
const double &dEpochTime,
int iScenario,
double* pvvvdFluxData,
int iPerturbFluxMap)

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 the *Ae9Ap9SetFluxEnvironment*() method.
This same method may also be called as *AppComputeFluxScenario*(), with same exact arguments.
The returned flux values are in units of [#/cm2/sec] (integral) or [#/cm2/sec/MeV] (differential).
Parameters:
zHandle – 'Ae9Ap9Model' object identifier, as returned from the Ae9Ap9StartUp method
dEpochTime – Monte Carlo reference time, in Modified Julian Date form
iScenario – scenario number of the Monte Carlo flux values to be returned.
pvvvdFluxData – returned 3-dimensional array of the flux values for the specified scenario number. [time,energy,direction]
iPerturbFluxMap – flag, needed only for developmental testing, for controlling application of flux perturbations within calculations. 0 *(false)*, or 1 *(true)* (recommended).
Return value: int – 0 = success, otherwise error

AccumModel Class

Header file: CAccumModel_c.h

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 *AppCompute**() 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:

HANDLE AccumStartUp

Usage: Instantiates a new 'AccumModel' object; multiple calls to this will generate separate instances, each of which may be accessed using the unique returned handle value.
Parameters: -none-
Return value: HANDLE - identifier value for the instantiated object

AccumShutDown

(HANDLE zHandle)
Usage: Destructor
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
Return values: -none-

Model Parameter Inputs:

int AccumSetTimeInterval

(HANDLE zHandle,
double dTimeInterval)
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 *AccumSetTimeIntervalSec*()) is used only with the *AccumComputeIntvFluence*(), *AccumComputeBoxcarFluence*() and *AccumComputeExponentialFlux*() methods. If not specified, the interval duration defaults to 1 day (86400 seconds).
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
dTimeInterval – time duration, in units of days+fraction *[must be greater than smallest timestep of ephemeris]*
Return value: int – 0 = success, otherwise error

int AccumSetTimeIntervalSec

(HANDLE zHandle,
double dTimeIntervalSec)
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 *AccumSetTimeInterval*()) is used only with the *AccumComputeIntvFluence*(), *AccumComputeBoxcarFluence*() and

AccumComputeExponentialFlux() methods. If not specified, the interval duration defaults to 86400 seconds (1 day).

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

dTimeIntervalSec – time duration, in seconds *[must be greater than smallest timestep of ephemeris]*

Return value: int – 0 = success, otherwise error

int AccumSetTimeIncrement

(HANDLE zHandle,
double dTimeIncrement)

Usage: Specifies the time delta for the shift of the 'Boxcar' accumulation mode time windows. This defined increment is used only with the *AccumComputeBoxcarFluence*() method.

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

dTimeIncrement – 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 *AccumSetTimeInterval*() method.

Return value: int – 0 = success, otherwise error

int AccumGetTimeInterval

(HANDLE zHandle,
double* pdTimeInterval)

Usage: Returns the time interval of the accumulation data results, as specified in the *AccumSetTimeInterval*() or *AccumSetTimeIntervalSec*() functions.

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

pdTimeInterval – pointer to double variable

Returned parameter:

pdTimeInterval – time duration, in units of days+fraction.

Return value: int – 0 = success, otherwise error

int AccumGetTimeIncrement

(HANDLE zHandle,
double* pdTimeIncrement)

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

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

pdTimeIncrement – pointer to double variable

Returned parameter:

pdTimeIncrement – time delta, in seconds, for the increment of time between the start of adjacent Boxcar time windows.

Return value: int – 0 = success, otherwise error

Model Execution and Results:

int AccumLoadBuffer

(HANDLE zHandle,
double* pvdTimes,
double* pvvvdData,
int iNumTimes,
int iNumValues,
int iNumDir)

Usage: Loads the sets of input time-tagged data into the accumulation buffer, replacing any previous buffer contents.

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

pvdTimes – array of time values, in Modified Julian Date form, size=*iNumTimes*

pvvvdData – 3-dimensional array of data values to be loaded into buffer, size=*iNumTimes* x *iNumValues* x *iNumDir*. [time,energy,direction]

iNumTimes – number of values in time array, and number of values in time dimension of data array

iNumValues – number of values in energy dimension of data array

iNumDir – number of values in direction dimension of data array

Return value: int – 0 = success, otherwise error

int AccumAddToBuffer

(HANDLE zHandle,
double dTime,
double* pvvdData,
int iNumValues,
int iNumDir)

Usage: Adds a single set of input time-tagged data to the current contents of the accumulation buffer.

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

dTime – time value, in Modified Julian Date form

pvvdData – 2-dimensional array of data values to be loaded into buffer, size=*iNumValues* x *iNumDir*. [energy,direction]

iNumValues – number of values in energy dimension of data array

iNumDir – number of values in direction dimension of data array

Return value: int – 0 = success, otherwise error

int AccumClearBuffer

(HANDLE zHandle)

Usage: Clears the current contents of the accumulation buffer.

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

Return value: int – 0 = success, otherwise error

int AccumGetBufferDim

(HANDLE zHandle,

int* piNumTimes,
int* piNumValues,
int* piNumDir)

Usage: Returns the data dimensions of the current contents of the accumulation buffer.

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

piNumTimes – pointer to int variable

piNumValues – pointer to int variable

piNumDir – pointer to int variable

Returned parameters:

piNumTimes – number of values in time dimension of data array

piNumValues – number of values in energy dimension of data array

piNumDir – number of values in direction dimension of data array

Return value: int – 0 = success, otherwise error

int AccumComputeFluence

(HANDLE zHandle,
double* pvdFluenceTimes,
double* pvvvdFluence)

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 <u>no</u> effect on these calculations.*

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

pvdFluenceTimes – pointer to double array, sized according to iNumTimes from *AccumGetBufferDim*() results

pvvvdFluence – pointer to double array, sized according to *AccumGetBufferDim*() results multiplied together (iNumTimes x iNumValues x iNumDir)

Returned parameters:

pvdFluenceTimes – array of times, in Modified Julian Date form

pvvvdFluence – 3-dimensional array of the calculated fluence values [time,energy,direction]

Return value: int – 0 or greater = number of data records returned in arrays; otherwise error

int AccumComputeIntvFluence

(HANDLE zHandle,
double* pvdFluenceTimes,
double* pvvvdFluence,
int* pviIntIndices,
int iReturnPartial)

Usage: Processes the current contents of the data buffer, then returns the time-integrated data results from any *<u>completed</u>* accumulation intervals (whose duration is previously specified in calls to the *AccumSetTimeInterval[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 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:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

pvdFluenceTimes, pviIntIndices – pointers to double arrays, sized according to iNumTimes from *AccumGetBufferDim*() results

pvvvdFluence – pointer to double array, sized according to *AccumGetBufferDim*() results multiplied together (iNumTimes x iNumValues x iNumDir)

iReturnPartial – 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, otherwise set to 0 *(false)*.

Returned parameters:

pvdFluenceTimes – array of times, in Modified Julian Date form

pvvvdFluence – 3-dimensional array of the calculated fluence values for zero or more completed accumulation intervals. [time,energy,direction]

pviIntIndices – returned 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.

Return value: int – 0 or greater = number of data records returned in arrays; otherwise error

int AccumAccumIntvFluence

(HANDLE zHandle,
double* pvdFluenceTimes,
double* pvvvdFluence,
int iNumTimes,
int iNumValues,
int iNumDir,
double* pvvvdFluenceIntvAccum,
int iAccumReset)

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 *AccumComputeIntvFluence*() method.

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

pvdFluenceTimes – array of times, in Modified Julian Date form

pvvvdFluence – 3-dimensional vector of the previously calculated fluence values. [time,energy,direction]

iNumTimes – number of values in time array, and number of values in time dimension of data array

iNumValues – number of values in energy dimension of data array

iNumDir – number of values in direction dimension of data array

pvvvdFluenceIntvAccum – pointer to double array of same size as pvvvdFluence

iAccumReset – flag for forcing a reset of the internal fluence accumulation data. 1 *(true)* or 0 *(false)*.

Returned parameters:

pvvvdFluenceIntvAccum – 3-dimensional array of the corresponding *cumulative* fluence values since the initial call to this routine, or a reset was indicated.

Return value: int – 0 or greater = number of data records returned in arrays; otherwise error

int AccumComputeFullFluence

(HANDLE zHandle,

double* pvdFluenceTimes,
double* pvvvdFluence,
int iReturnFinal)

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:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

pvdFluenceTimes – pointer to double array, sized according to iNumTimes from *AccumGetBufferDim*() results

pvvvdFluence – pointer to double array, sized according to *AccumGetBufferDim*() results multiplied together (iNumTimes x iNumValues x iNumDir)

iReturnFinal – flag for returning the single set of cumulative fluence values. Set to 1 *(true)* only after *all* data has been processed, otherwise set to 0 *(false)*.

Returned parameters:

pvdFluenceTimes – array of times, in Modified Julian Date form

pvvvdFluence – 3-dimensional 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]

Return value: int – 0 or 1 = number of data records returned in arrays; otherwise error

int AccumComputeBoxcarFluence

(HANDLE zHandle,
double* pvdFluenceTimes,
double* pvvvdFluence,
int* pviIntIndices,
int iReturnPartial)

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 *AccumSetTimeInterval[Sec]*() methods; the time spacing between the start times of subsequent boxcar windows is specified with the *AccumSetTimeIncrement*() 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 *AccumComputeAverageFlux*() 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:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

pvdFluenceTimes, pviIntIndices – pointers to double arrays, sized according to iNumTimes from *AccumGetBufferDim*() results

pvvvdFluence – pointer to double array, sized according to *AccumGetBufferDim*() results multiplied together (iNumTimes x iNumValues x iNumDir)

iReturnPartial – 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, otherwise set to 0 *(false)*. Also causes the boxcar accumulation information to be reset.

Returned parameters:

pvdFluenceTimes – array of times, in Modified Julian Date form

pvvvdFluence – 3-dimensional array of the calculated fluence values for zero or more completed boxcar accumulation intervals. [time,energy,direction]

pviIntIndices – 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.

Return value: int – 0 or greater = number of data records returned in arrays; otherwise error

int AccumComputeAverageFlux

(HANDLE zHandle,
double* pvdFluenceTimes,
double* pvvvdFluence,
double dIntervalSec,
int iNumTimes,
int iNumValues,
int iNumDir,
double* pvvvdFluxAvg)

Usage: Computes the average flux rate over fixed-length intervals, based on the input fluence and interval time.

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

pvdFluenceTimes – array of times, in Modified Julian Date form

pvvvdFluence – 3-dimensional array of the calculated fluence values, associated with the interval end times specified in pvdFluenceTimes. [time,energy,direction]

dIntervalSec – duration of time interval, in seconds, used in the calculation of the fluence values. Specify the value used in the *AccumSetTimeInterval[Sec]*() method for 'interval'- and 'boxcar'-type accumulations; specify '0' for use with cumulative fluences (no accumulation); specify '-1' for 'full' accumulation fluence.

iNumTimes – number of values in time array, and number of values in time dimension of data array

iNumValues – number of values in energy dimension of data array

iNumDir – number of values in direction dimension of data array

pvvvdFluxAvg – pointer to double array, of same size as pvvvdFluence

Returned parameter:

pvvvdFluxAvg – 3-dimensional array of the calculated flux average values [time,energy,direction]

Return value: int – 0 or greater: number of sets of flux averages returned; otherwise error

int AccumComputeExponentialFlux

(HANDLE zHandle,
double* pvdExpFluxTimes,
double* pvvvdExpFlux,
int* pviIntIndices,
int iFinal)

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 call to the *AccumSetTimeInterval[Sec]*() method.
The *applyWorstToDate*() method can be used to further process these results, when the appropriate 'maximum' array of values is maintained.

Parameters:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

pvdExpFluxTimes, pviIntIndices – pointers to double arrays, sized according to iNumTimes from *AccumGetBufferDim*() results

pvvvdExpFlux – pointer to double array, sized according to *AccumGetBufferDim*() results multiplied together (iNumTimes x iNumValues x iNumDir)

Returned parameters:

vdExpFluxTimes – array of times, in Modified Julian Date form

pvvvdExpFlux – 3-dimensional array of the calculated exponential average flux values for zero or more input data entries. [time,energy,direction]

pviIntIndices – 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.

iFinal – flag for process completion. Set to 0 while actively processing data; when all data has been processed, set to 1 on the final call to this routine.

Return value: int – 0 or greater = number of data records returned in arrays; otherwise error

int AccumApplyWorstToDate

```
( HANDLE zHandle,
  int iNumTimes,
  int iNumValues,
  int iNumDir,
  double* pvvvdData,
  double* pvvdMaxData,
  double* pvvvdDataWorst )
```

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:

zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method

iNumTimes – number of values in time array, and number of values in time dimension of data array

iNumValues – number of values in energy dimension of data array

iNumDir – number of values in direction dimension of data array

pvvvdData – 3-dimensional array of data, of no specific type. [time,energy|depth,direction]

pvvdMaxData – input and returned 2-dimensional array of the (current and previously) determined maximum data values. [energy|depth,direction]. A "reset" of these maximum values is implied when an empty pvvvdMaxData array is passed as *input*.

Returned parameters:

pvvdMaxData – 2-dimensional array of the current determined maximum data values. [energy|depth,direction]

pvvvdDataWorst – 3-dimensional array of the corresponding 'worst to date' of the input data values. [time,energy|depth,direction]
Return value: int – 0 = success, otherwise error

int AccumResetFluence

(HANDLE zHandle)
Usage: clears the internally stored timestep-based cumulative fluence accumulation data. Subsequent calls to AccumCompute*Fluence*() method will start a new set of fluence accumulation.
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
Return value: int – 0 = success, otherwise error

int AccumResetIntvFluence

(HANDLE zHandle)
Usage: clears the internally stored interval-based cumulative fluence accumulation data. Subsequent calls to AccumCompute*IntvFluence*() method will start a new set of fluence accumulation.
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
Return value: int – 0 = success, otherwise error

int AccumResetFullFluence

(HANDLE zHandle)
Usage: clears the internally stored full fluence accumulation data. Subsequent calls to AccumCompute*FullFluence*() method will start a new set of fluence accumulation.
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
Return value: int – 0 = success, otherwise error

int AccumResetBoxcarFluence

(HANDLE zHandle)
Usage: clears the internally stored boxcar fluence accumulation data. Subsequent calls to AccumCompute*BoxcarFluence*() method will start a new set of fluence accumulation.
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
Return value: int – 0 = success, otherwise error

int AccumResetExponentialFlux

(HANDLE zHandle)
Usage: clears the internally stored exponential flux average data. Subsequent calls to AccumCompute*ExponentialFlux*() method will start a new set of exponential flux average calculations.
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
Return value: int – 0 = success, otherwise error

int AccumGetFluenceStartTime

(HANDLE zHandle,

double* pdStartTime)
Usage: Returns the starting time for the cumulative fluence accumulation data.
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
pdStartTime – pointer to double variable
Returned parameter:
pdStartTime – fluence accumulation data start time, in MJD form.
Return value: int – 0 = success, otherwise error

int AccumGetIntvFluenceStartTime

(HANDLE zHandle,
double* pdStartTime)
Usage: Returns the starting time for the current interval of fluence accumulation data.
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
pdStartTime – pointer to double variable
Returned parameter:
pdStartTime – fluence accumulation data start time, in MJD form.
Return value: int – 0 = success, otherwise error

int AccumGetFullFluenceStartTime

(HANDLE zHandle,
double* pdStartTime)
Usage: Returns the starting time for the full fluence accumulation data.
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
pdStartTime – pointer to double variable
Returned parameter:
pdStartTime – full fluence accumulation data start time, in MJD form.
Return value: int – 0 = success, otherwise error

int AccumGetBoxcarFluenceStartTime

(HANDLE zHandle,
double* pdStartTime)
Usage: Returns the starting time for the oldest active boxcar interval of fluence accumulation data.
Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
pdStartTime – pointer to double variable
Returned parameter:
pdStartTime – boxcar fluence accumulation data start time, in MJD form.
Return value: int – 0 = success, otherwise error

int AccumGetLastLength

(HANDLE zHandle,
double* pdLastLength)
Usage: Returns the last interval time duration of the most recent *AccumCompute*Fluence*|*Flux*() method call in which the 'bReturnPartial' input parameter was set to 1 *(true)*.

Parameters:
zHandle – 'AccumModel' object identifier, as returned from the AccumStartUp method
pdLastLength – pointer to double variable
Returned parameter:
pdLastLength– last interval time duration, in seconds. -1.0 if no partial interval occurred.
Return value: int – 0 = success, otherwise error

DoseModel Class

Header file: CDoseModel_c.h

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:

HANDLE DoseStartUp

Usage: Instantiates a new 'DoseModel' object; multiple calls to this will generate separate instances, each of which may be accessed using the unique returned handle value.
Parameters: -none-
Return value: HANDLE - identifier value for the instantiated object

DoseShutDown

(HANDLE zHandle)
Usage: Destructor
Parameters:
zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method
Return values: -none-

Model Parameter Inputs:

int DoseSetModelDBDir

(HANDLE zHandle,
char* szDataDir)
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:
zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method
szDataDir – directory path for the IRENE database files.
Return value: int – 0 = success, otherwise error

int DoseSetModelDBFile

(HANDLE zHandle,
char* szModelDBFile)
Usage: Specifies the name of the file for the dose calculation model database. The use of this method is *not needed* when *DoseSetModelDBDir*() 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:
zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method
szModelDBFile – model database filename, including path
Return value: int – 0 = success, otherwise error

int DoseSetSpecies

(HANDLE zHandle,
char* szSpecies)

Usage: Specifies the particle species for the ShieldDose2 model calculations.

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

szSpecies – species identification: 'H+'|'protons' or 'e-'|'electrons'.

Return value: int – 0 = success, otherwise error

int DoseSetEnergies

(HANDLE zHandle,
double* pvdEnergies,
int iNumEnergies,
char* szEnergyUnits)

Usage: Specifies the set of energies (and units) of the flux values that are to be input to the *DoseComputeFluxDose[Rate]*() methods.

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

pvdEnergies – array of energy values

iNumEnergies – number of values in energy array

szEnergyUnits – energy units: 'eV', 'keV', 'MeV' or 'GeV'

Return value: int – 0 = success, otherwise error

int DoseSetDepths

(HANDLE zHandle,
double* pvdDepths,
int iNumDepths,
char* szDepthUnits)

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:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

pvdDepths – array of depth values; in ascending order, no duplicates

iNumDepths – number of values in depth array

szDepthUnits – unit specification: 'millimeters'|'mm', 'mils' or 'gpercm2'

Return value: int – 0 = success, otherwise error

int DoseSetDetector

(HANDLE zHandle,
char* szDetector)

Usage: Specifies the dose detector material type, behind (or inside) the aluminum shielding.

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

szDetector – 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: int – 0 = success, otherwise error

int DoseSetGeometry

(HANDLE zHandle,
char* szGeometry)

Usage: Specifies the geometry of the aluminum shielding in front of (or around) the detector target.

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

szGeometry – configuration name: 'Spherical4pi', 'Spherical2pi', 'FiniteSlab' or 'SemiInfiniteSlab'

Return value: int – 0 = success, otherwise error

int DoseSetNuclearAttenMode

(HANDLE zHandle,
char* szNucAttenMode)

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

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

szNucAttenMode – attenuation mode: 'None', 'NuclearInteractions' or 'NuclearAndNeutrons'

Return value: int – 0 = success, otherwise error

int DoseSetWithBrems

(HANDLE zHandle,
int iVerdict)

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:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

iVerdict – flag for including(1) or excluding(0) the bremsstrahlung contributions.

Return value: int – 0 = success, otherwise error

---

int DoseGetModelDBDir

(HANDLE zHandle,
char* szDataDir)

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

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

szDataDir – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataDir – model database directory.

Return value: int – 0 = success, otherwise error

int DoseGetModelDBFile

(HANDLE zHandle,
char* szModelDBFile)

Usage: Returns the name of the database file used for the ShieldDose2 model calculations. This will be available immediately, when specified using the *DoseSetModelDBFile*() method. When the *DoseSetModelDBDir*() method is used, the automatically determined filename will be available after a call to one of the *DoseCompute**() methods.

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

szModelDBFile – pointer to character string (suggested sizing = 256)

Returned parameters:

szModelDBFile – model database filename, including path.

Return value: int – 0 = success, otherwise error

int DoseGetSpecies

(HANDLE zHandle,
char* szSpecies)

Usage: Returns the particle species for the ShieldDose2 model calculations, as specified in the DoseSetSpecies() method.

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

szSpecies – pointer to character string (suggested sizing = 16)

Returned parameters:

szSpecies – species identification.

Return value: int – 0 = success, otherwise error

int DoseGetNumEnergies

(HANDLE zHandle)

Usage: Returns the number of currently defined energies, as specified in the *DoseSetEnergies*() method.

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

Return value: int – number of energies.

int DoseGetNumDepths

(HANDLE zHandle)

Usage: Returns the number of currently defined dose depths, as specified in the *DoseSetDepths*() method.

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

Return value: int – number of depths.

int DoseGetDetector

(HANDLE zHandle,
char* szDetector)

Usage: Returns the dose detector material type, behind (or inside) the aluminum shielding, as specified in the *DoseSetDetector*() method.

Parameters:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

szDetector – pointer to character string (suggested sizing = 32)

Returned parameters:
szDetector – material name.
Return value: int – 0 = success, otherwise error

int DoseGetGeometry

(HANDLE zHandle,
char* szGeometry)

Usage: Returns the geometry of the aluminum shielding in front of (or around) the detector target, as specified in the *DoseSetGeometry*() method.
Parameters:
zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method
szGeometry – pointer to character string (suggested sizing = 32)
Returned parameters:
szGeometry – geometry configuration name.
Return value: int – 0 = success, otherwise error

int DoseGetNuclearAttenMode

(HANDLE zHandle,
char* szNucAttenMode)

Usage: Returns the 'Nuclear Attenuation' mode used during the ShieldDose2 model calculations, as specified in the *DoseSetNuclearAttenMode*() method.
Parameters:
zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method
szNucAttenMode – pointer to character string (suggested sizing = 32)
Returned parameters:
szNucAttenMode – attenuation mode.
Return value: int – 0 = success, otherwise error

int DoseGetWithBrems

(HANDLE zHandle)

Usage: Returns the current state for the inclusion of the bremsstrahlung contribution in the electron dose values calculated, as specified in the *DoseSetWithBrems*() method.
Parameters:
zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method
Return value: 1 *(true)* or 0 *(false).*

Model Execution and Results:

int DoseComputeFluxDose

(HANDLE zHandle,
double* pvdFluxData,
int iNumEnergies,
double* pvdDoseData)

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:
zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

pvdFluxData – array of omni-directional differential flux values, over the energy levels previously specified via the *DoseSetEnergies*() 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)

iNumEnergies – number of values in pvdFluxData array

pvdDoseData – pointer to double array, sized according to *DoseGetNumDepths*() results.

Returned parameter:

pvdDoseData – array of dose model results, over the shielding depths previously specified via the *DoseSetDepths*() 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].

Return value: int – 0 = success, otherwise error

int DoseComputeFluxDoseRate

(HANDLE zHandle,
double* pvvdFluxData,
int iNumTimes,
int iNumEnergies,
double* pvvvdDoseRate)

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:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

pvvdFluxData – 2-dimensional array of omni-directional differential flux values, over the energy levels previously specified via the *DoseSetEnergies*() method, for one or more times.[time,energy]
These input flux values are expected to be in units of #/cm^2/sec/MeV

iNumTimes – length of time dimension in pvdFluxData array

iNumEnergies – length of energy dimension in pvdFluxData array

pvvvdDoseData – pointer to double array, sized according to *iNumTimes* x *DoseGetNumDepths*() results.

Returned parameter:

pvvvdDoseData – 2-dimensional array of dose rates [rads/sec], over the shielding depths previously specified via the *DoseSetDepths*() method. [time,depths]

Return value: int – 0 = success, otherwise error

int DoseComputeFluenceDose

(HANDLE zHandle,
double* pvvvdFluenceData,
int iNumTimes,
int iNumEnergies,
double* pvvvdDoseVal)

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:

zHandle – 'DoseModel' object identifier, as returned from the DoseStartUp method

pvvvdFluenceData – 3-dimensional array of omni-directional differential fluence values, over the energy levels previously specified via the *DoseSetEnergies*() 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

iNumTimes – length of time dimension in pvdFluxData array

iNumEnergies – length of energy dimension in pvdFluxData array

pvvvdDoseVal – pointer to double array, sized according to *iNumTimes* x *DoseGetNumDepths*() results.

Returned parameter:

pvvvdDoseVal – 3-dimensional array of dose values [rads], for a single direction, over the shielding depths previously specified via the *DoseSetDepths*() method. [time,depths,direction]

Return value: int – 0 = success, otherwise error

DoseKernel Class

Header file: CDoseKernel_c.h

This class is the entry point that provides direct programmatic access to 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:

HANDLE DoseKStartUp

Usage: Instantiates a new 'DoseKernel' object; multiple calls to this will generate separate instances, each of which may be accessed using the unique returned handle value.
Parameters: -none-
Return value: HANDLE - identifier value for the instantiated object

DoseKShutDown

(HANDLE zHandle)
Usage: Destructor
Parameters:
zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method
Return values: -none-

Model Parameter Inputs:

int DoseKSetKernelXmlPath

(HANDLE zHandle,
char* szKernelXmlPath)
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:
zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method
szKernelXmlPath – dose kernel xml file collection path
Return value: int – 0 = success, otherwise error

int DoseKSetKernelXmlFile

(HANDLE zHandle,
char* szKernelXmlFile)
Usage: Specifies the name of the file for the kernel-based dose calculation method.
The dose xml file specified *must* correspond to the specified geometry and detector material.
Parameters:
zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method
szKernelXmlFile – dose kernel xml filename, including path
Return value: int – 0 = success, otherwise error

int DoseKSetSpecies

(HANDLE zHandle,
char* szSpecies)

Usage: Specifies the particle species for the ShieldDose2 model calculations.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

szSpecies – species identification: 'H+'|'protons' or 'e-'|'electrons'.

Return value: int – 0 = success, otherwise error

int DoseKSetEnergies

(HANDLE zHandle,
double* pvdEnergies,
int iNumEnergies,
char* szEnergyUnits)

Usage: Specifies the set of energies and units of the flux values that are to be input to the *DoseKComputeFluxDose[Rate]*() methods.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

pvdEnergies – array of energy values

iNumEnergies – number of values in energy array

szEnergyUnits – energy units: 'eV', 'keV', 'MeV' or 'GeV'

Return value: int – 0 = success, otherwise error

int DoseKSetDepths

(HANDLE zHandle,
double* pvdDepths,
int iNumDepths,
char* szDepthUnits)

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:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

pvdDepths – array of depth values; in ascending order, no duplicates

iNumDepths – number of values in depth array

szDepthUnits – unit specification: 'millimeters'|'mm', 'mils' or 'gpercm2'

Return value: int – 0 = success, otherwise error

int DoseKSetDetector

(HANDLE zHandle,
char* szDetector)

Usage: Specifies the dose detector material type, behind (or inside) the aluminum shielding.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

szDetector – 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: int – 0 = success, otherwise error

int DoseKSetGeometry

(HANDLE zHandle,
char* szGeometry)

Usage: Specifies the geometry of the aluminum shielding in front of (or around) the detector target.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

szGeometry – configuration name: 'Spherical4pi', 'Spherical2pi', 'FiniteSlab' or 'SemiInfiniteSlab'

Return value: int – 0 = success, otherwise error

int DoseKSetNuclearAttenMode

(HANDLE zHandle,
char* szNucAttenMode)

Usage: Specifies the 'Nuclear Attenuation' mode of the kernel dose calculations.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

szNucAttenMode – attenuation mode: 'None'

(other modes of 'NuclearInteractions' and 'NuclearAndNeutrons' are currently unsupported)

Return value: int – 0 = success, otherwise error

int DoseKSetWithBrems

(HANDLE zHandle,
int iVerdict)

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:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

iVerdict – flag for including(1) or excluding(0) the bremsstrahlung contributions.

Return value: int – 0 = success, otherwise error

---

int DoseKGetKernelXmlPath

(HANDLE zHandle,
char* szKernelXmlPath)

Usage: Returns the path for the dose kernel xml files used for dose calculations, as specified in the *DoseKSetKernelXmlPath*() method.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

szKernelXmlPath – pointer to character string (suggested sizing = 256)

Returned parameters:

szKernelXmlPath – dose kernel xml path.

Return value: int – 0 = success, otherwise error

int DoseKGetKernelXmlFile

(HANDLE zHandle,

char* szKernelXmlFile)

Usage: Returns the name of the dose kernel xml file used for dose calculations, as specified in the *DoseKSetKernelXmlFile*() method.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

szKernelXmlFile – pointer to character string (suggested sizing = 256)

Returned parameters:

szKernelXmlFile – dose kernel xml filename.

Return value: int – 0 = success, otherwise error

int DoseKGetSpecies

(HANDLE zHandle,
char* szSpecies)

Usage: Returns the particle species for the ShieldDose2 model calculations, as specified in the DoseKSetSpecies() method.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

szSpecies – pointer to character string (suggested sizing = 16)

Returned parameters:

szSpecies – species identification.

Return value: int – 0 = success, otherwise error

int DoseKGetNumEnergies

(HANDLE zHandle)

Usage: Returns the number of currently defined energies, specified in the *DoseKSetEnergies*() method.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

Return value: int – number of energies.

int DoseKGetNumDepths

(HANDLE zHandle)

Usage: Returns the number of currently defined dose depths, specified in the *DoseKSetDepths*() method.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

Return value: int – number of depths.

int DoseKGetDetector

(HANDLE zHandle,
char* szDetector)

Usage: Returns the dose detector material type, behind (or inside) the aluminum shielding, as specified in the *DoseKSetDetector*() method.

Parameters:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

szDetector – pointer to character string (suggested sizing = 32)

Returned parameters:

szDetector – material name.
Return value: int – 0 = success, otherwise error

int DoseKGetGeometry

(HANDLE zHandle,
char* szGeometry)

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

Parameters:
zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method
szGeometry – pointer to character string (suggested sizing = 16)

Returned parameters:
szGeometry – geometry configuration name.

Return value: int – 0 = success, otherwise error

int DoseKGetNuclearAttenMode

(HANDLE zHandle,
char* szNucAttenMode)

Usage: Returns the 'Nuclear Attenuation' mode of the kernel dose calculations, as specified in the *DoseKSetNuclearAttenMode*() method.

Parameters:
zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method
szNucAttenMode – pointer to character string (suggested sizing = 32)

Returned parameters:
szNucAttenMode – attenuation mode.

Return value: int – 0 = success, otherwise error

int DoseKGetWithBrems

(HANDLE zHandle)

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

Parameters:
zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

Return value: 1 *(true)* or 0 *(false).*

Model Execution and Results:

int DoseKComputeFluxDose

(HANDLE zHandle,
double* pvdFluxData,
int iNumEnergies,
double* pvdDoseData)

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:
zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

pvdFluxData – array of omni-directional differential flux values, over the energy levels previously specified via the *DoseKSetEnergies*() 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)

iNumEnergies – number of values in pvdFluxData array

pvdDoseData – pointer to double array, sized according to *DoseKGetNumDepths*() results.

Returned parameter:

pvdDoseData – array of dose model results, over the shielding depths previously specified via the *DoseKSetDepths*() 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].

Return value: int – 0 = success, otherwise error

int DoseKComputeFluxDoseRate

(HANDLE zHandle,
double* pvvdFluxData,
int iNumTimes,
int iNumEnergies,
double* pvvvdDoseRate)

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:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

pvvdFluxData – 2-dimensional array of omni-directional differential flux values, over the energy levels previously specified via the *DoseKSetEnergies*() method, for one or more times.[time,energy]
These input flux values are expected to be in units of #/cm^2/sec/MeV

iNumTimes – length of time dimension in pvdFluxData array

iNumEnergies – length of energy dimension in pvdFluxData array

pvvvdDoseData – pointer to double array, sized according to *iNumTimes* x *DoseKGetNumDepths*() results.

Returned parameter:

pvvvdDoseData – 2-dimensional array of dose rates [rads/sec], over the shielding depths previously specified via the *DoseKSetDepths*() method. [time,depths]

Return value: int – 0 = success, otherwise error

int DoseKComputeFluenceDose

(HANDLE zHandle,
double* pvvvdFluenceData,
int iNumTimes,
int iNumEnergies,
double* pvvvdDoseVal)

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:

zHandle – 'DoseKernel' object identifier, as returned from the DoseKStartUp method

pvvvdFluenceData – 3-dimensional array of omni-directional differential fluence values, over the energy levels previously specified via the *DoseKSetEnergies*() 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

iNumTimes – length of time dimension in pvdFluxData array

iNumEnergies – length of energy dimension in pvdFluxData array

pvvvdDoseVal – pointer to double array, sized according to *iNumTimes* x *DoseKGetNumDepths*() results.

Returned parameter:

pvvvdDoseVal – 3-dimensional array of dose values [rads], for a single direction, over the shielding depths previously specified via the *DoseKSetDepths*() method. [time,depths,direction]

Return value: int – 0 = success, otherwise error

AggregModel Class

Header file: CAggregModel_c.h

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 AggregA*ddScenToAgg*() method, the confidence levels may be calculated using the various *AggregCompute**() 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 'Aggregulation and Aggregation Inputs' section of the User's Guide document for more details.

General:

HANDLE AggregStartUp

Usage: Instantiates a new 'AggregModel' object; multiple calls to this will generate separate instances, each of which may be accessed using the unique returned handle value.
Parameters: -none-
Return value: HANDLE - identifier value for the instantiated object

AggregShutDown

(HANDLE zHandle)
Usage: Destructor
Parameters:
zHandle – 'AggregModel' object identifier, as returned from the AggregStartUp method
Return values: -none-

Model Parameter Inputs:

int AggregResetAgg

(HANDLE zHandle)
Usage: Clears all time-tagged data from the scenario data aggregation.
Parameters:
zHandle – 'AggregModel' object identifier, as returned from the AggregStartUp method
Return value: int – 0 = success, otherwise error

int AggregAddScenToAgg

(HANDLE zHandle,
double* pvdScenTimes,
double* pvvvdData,
int iNumTimes,
int iNumValues,
int iNumDir)

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 *AggregResetAgg*() method.

Parameters:

zHandle – 'AggregModel' object identifier, as returned from the AggregStartUp method

pvdScenTimes – array of time values, in Modified Julian Date form

pvvvdData – 3-dimensional array of scenario data values to be loaded into the aggregation. [time,energy|depth,direction]

iNumTimes – length of pvdScenTimes array and time dimension in pvvvdData array

iNumValues – length of energy or depth dimension in pvvvdData array

iNumDir – length of direction dimension in pvvvdData array

Return value: int – 0 = success, otherwise error

int AggregGetAggDimensions

(HANDLE zHandle,
int* piNumTimes,
int* piNumValues,
int* piNumDir)

Usage: Returns the dimensions of the currently defined aggregation data entries, as specified in *AggregAddScenToAgg*() calls.

Parameters:

zHandle – 'AggregModel' object identifier, as returned from the AggregStartUp method

piNumTimes, piNumValues, *piNumDir* – pointers to int variables

Returned paremeters:

piNumTimes – length of times dimension in current aggregation data

piNumValues – length of energy or depth dimension in current aggregation data

piNumDir – length of direction dimension in current aggregation data

Return value: int – 0 = success, otherwise error

int AggregGetNumScenarios

(HANDLE zHandle)

Usage: Returns the number of currently defined aggregation scenario entries, specified in the *AggregAddScenToAgg*() method.

Parameters:

zHandle – 'AggregModel' object identifier, as returned from the AggregStartUp method

Return value: int – number of aggregation scenarios.

Model Execution and Results:

int AggregComputeConfLevel

(HANDLE zHandle,
int iPercent,
double* pvdPercTimes,
double* pvvvdPercData)

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:

zHandle – 'AggregModel' object identifier, as returned from the AggregStartUp method

iPercent – input confidence level percent value (0-100)

pvdPercTimes – pointer to double array, sized according to iNumTimes from *AggregGetAggDimensions*() results

pvvvdPercData – pointer to double array, sized according to iNumTimes x iNumValues x iNumDir from *AggregGetAggDimensions*() results

Returned parameters:

pvdPercTimes – array of times, in Modified Julian Date form

pvvvdPercData – 3-dimensional array of the calculated confidence level values [time,energy|depth,direction]

Return value: int – 0 = success, otherwise error

int AggregComputeMedian

(HANDLE zHandle,
double* pvdPercTimes,
double* pvvvdPercData)

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.

Parameters:

zHandle – 'AggregModel' object identifier, as returned from the AggregStartUp method

pvdPercTimes – pointer to double array, sized according to iNumTimes from *AggregGetAggDimensions*() results

pvvvdPercData – pointer to double array, sized according to iNumTimes x iNumValues x iNumDir from *AggregGetAggDimensions*() results

Returned parameters:

pvdPercTimes – array of times, in Modified Julian Date form

pvvvdPercData – 3-dimensional array of the calculated aggregation median values [time,energy|depth,direction]

Return value: int – 0 = success, otherwise error

int AggregComputeMean

(HANDLE zHandle,
double* pvdPercTimes,
double* pvvvdPercData)

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.

Parameters:

zHandle – 'AggregModel' object identifier, as returned from the AggregStartUp method

pvdPercTimes – pointer to double array, sized according to iNumTimes from *AggregGetAggDimensions*() results

pvvvdPercData – pointer to double array, sized according to iNumTimes x iNumValues x iNumDir from *AggregGetAggDimensions*() results

Returned parameters:

pvdPercTimes – array of times, in Modified Julian Date form

pvvvdPercData – 3-dimensional array of the calculated aggregation mean values [time,energy|depth,direction]

Return value: int – 0 = success, otherwise error

AdiabatModel Class

Header file: CAdiabatModel_c.h

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 Adiabatendix D and E of the User's Guide document.

General:

HANDLE AdiabatStartUp

Usage: Instantiates a new 'AdiabatModel' object; multiple calls to this will generate separate instances, each of which may be accessed using the unique returned handle value.
Parameters: -none-
Return value: HANDLE - identifier value for the instantiated object

AdiabatShutDown

(HANDLE zHandle)
Usage: Destructor
Parameters:
zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method
Return values: -none-

Model Parameter Inputs:

int AdiabatSetModelDBDir

(HANDLE zHandle,
char* szDataDir)
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:
zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method
szDataDir – directory path for the IRENE database files.
Return value: int – 0 = success, otherwise error

int AdiabatSetKPhiDBFile

(HANDLE zHandle,
char* szKPhiDBFile)

Usage: Specifies the name of the file for the K/Phi database. The use of this method is *not needed* when *AdiabatSetModelDBDir*() 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:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

szKPhiDBFile – database filename, including path.

Return value: int – 0 = success, otherwise error

int AdiabatSetKHMinDBFile

(HANDLE zHandle,
char* szKHMinDBFile)

Usage: Specifies the name of the file for the K/Hmin database. The use of this method is *not needed* when *AdiabatSetModelDBDir*() 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:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

szKHMinDBFile – database filename, including path.

Return value: int – 0 = success, otherwise error

int AdiabatSetMagfieldDBFile

(HANDLE zHandle,
char* szMagfieldDBFile)

Usage: Specifies the name of the file for the magnetic field model database. The use of this method is *not needed* when *AdiabatSetModelDBDir*() 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:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

szMagfieldDBFile – magnetic field model database filename, including path.

Return value: int – 0 = success, otherwise error

---

int AdiabatGetModelDBDir

(HANDLE zHandle,
char* szDataDir)

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

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

szDataDir – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataDir – model database directory.

Return value: int – 0 = success, otherwise error

int AdiabatGetKPhiDBFile

(HANDLE zHandle,
char* szKPhiDBFile)

Usage: Returns the name of the file for the K/Phi database. This will be available immediately, when specified using the *AdiabatSetKPhiDBFile*() method. When the *AdiabatSetModelDBDir*() method is used, the automatically determined filename will be available after a call to one of the *AdiabatComputeCoordinate**() methods.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

szKPhiDBFile – pointer to character string (suggested sizing = 256)

Returned parameter:

szKPhiDBFile – database filename, including path.

Return value: int – 0 = success, otherwise error

int AdiabatGetKHMinDBFile

(HANDLE zHandle,
char* szKHMinDBFile)

Usage: Returns the name of the file for the K/Hmin database. This will be available immediately, when specified using the *AdiabatSetKHMinDBFile*() method. When the *AdiabatSetModelDBDir*() method is used, the automatically determined filename will be available after a call to one of the *AdiabatComputeCoordinate**() methods.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

szKHMinDBFile – pointer to character string (suggested sizing = 256)

Returned parameter:

szKHMinDBFile – database filename, including path.

Return value: int – 0 = success, otherwise error

int AdiabatGetMagfieldDBFile

(HANDLE zHandle,
char* szMagfieldDBFile)

Usage: Specifies the name of the file for the magnetic field model database. This will be available immediately, when specified using the *AdiabatSetMagfieldDBFile*() method. When the *AdiabatSetModelDBDir*() method is used, the automatically determined filename will be available after a call to one of the *AdiabatComputeCoordinate**() methods.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

szMagfieldDBFile – pointer to character string (suggested sizing = 256)

Returned parameter:

szMagfieldDBFile – magnetic field model database filename, including path.

Return value: int – 0 = success, otherwise error

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.

int AdiabatSetK_Min

(HANDLE zHandle,
double dK_Min)

Usage: Specifies the minimum 'K' value returned from *AdiabatComputeCoordinateSet*() methods.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

dK_Min – 'K' adiabatic value minimum (0.0 if not specified)

Return value: int – 0 = success, otherwise error

int AdiabatSetK_Max

(HANDLE zHandle,
double dK_Max)

Usage: Specifies the maximum 'K' value returned from *AdiabatComputeCoordinateSet*() methods.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

dK_Min – 'K' adiabatic value maximum (25.0 if not specified)

Return value: int – 0 = success, otherwise error

int AdiabatSetHminMin

(HANDLE zHandle,
double dHminMin)

Usage: Specifies the minimum 'Hmin' value, altitude in km, returned from *AdiabatComputeCoordinateSet*() methods.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

dHminMin – 'Hmin' adiabatic value minimum (-1500.0 if not specified)

Return value: int – 0 = success, otherwise error

int AdiabatSetHminMax

(HANDLE zHandle,
double dHminMax)

Usage: Specifies the maximum 'Hmin' value, altitude in km, returned from *AdiabatComputeCoordinateSet*() methods.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

dHminMin – 'Hmin' adiabatic value maximum (50000.0 if not specified)

Return value: int – 0 = success, otherwise error

int AdiabatSetPhiMin

(HANDLE zHandle,
double dPhiMin)

Usage: Specifies the minimum 'Phi' value returned from *AdiabatComputeCoordinateSet*() methods.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

dPhiMin – 'Phi' adiabatic value minimum (0.125 if not specified)

Return value: int – 0 = success, otherwise error

int AdiabatSetPhiMax

(HANDLE zHandle,
double dPhiMax)

Usage: Specifies the maximum 'Phi' value returned from *AdiabatComputeCoordinateSet*() methods.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

dPhiMin – 'Phi' adiabatic value maximum (2.5 if not specified)

Return value: int – 0 = success, otherwise error

int AdiabatUpdateLimits

Usage: Implements any changes to the K, Hmin or Phi limit specifications, but requires that the database files have already been specified via AdiabatSetKPhiDBFile(), AdiabatSetKHMinDBFile() and AdiabatSetMagfieldDBFile() methods. Use of this method is needed only if any of these limits are changed *after* the initial call to one of the *AdiabatComputeCoordinateSet*() methods.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

Return value: int – 0 = success, otherwise error

---

int AdiabatGetK_Min

(HANDLE zHandle,
double* pdK_Min)

Usage: Returns the minimum 'K' value, as specified in the *AdiabatSetK_Min*() method.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

pdK_Min – pointer to double variable

Returned parameter:

pdK_Min – 'K' adiabatic value minimum (0.0 if not specified)

Return value: int – 0 = success, otherwise error

int AdiabatGetK_Max

(HANDLE zHandle,
double* pdK_Max)

Usage: Returns the maximum 'K' value, as specified in the *AdiabatSetK_Max*() method.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

pdK_Max – pointer to double variable

Returned parameter:

pdK_Max – 'K' adiabatic value maximum (25.0 if not specified)

Return value: int – 0 = success, otherwise error

int AdiabatGetHminMin

(HANDLE zHandle,
double* pdHminMin)

Usage: Returns the minimum 'Hmin' value, altitude in km, as specified in the AdiabatSetHminMin() method.

Parameters:
zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method
pdHminMin – pointer to double variable
Returned parameter:
pdHminMin – 'Hmin' adiabatic value minimum (-1500.0 if not specified)
Return value: int – 0 = success, otherwise error

int AdiabatGetHminMax

(HANDLE zHandle,
double* pdHminMax)

Usage: Returns the maximum 'Hmin' value, altitude in km, as specified in the *AdiabatSetHminMax*() method.
Parameters:
zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method
pdHminMax – pointer to double variable
Returned parameter:
pdHminMax – 'Hmin' adiabatic value maximum (50000.0 if not specified)
Return value: int – 0 = success, otherwise error

int AdiabatGetPhiMin

(HANDLE zHandle,
double* pdPhiMin)

Usage: Returns the minimum 'Phi' value, as specified in the *AdiabatSetPhiMin*() method.
Parameters:
zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method
pdPhiMin – pointer to double variable
Returned parameter:
pdPhiMin – 'Phi' adiabatic value minimum (0.125 if not specified)
Return value: int – 0 = success, otherwise error

int AdiabatGetPhiMax

(HANDLE zHandle,
double* pdPhiMax)

Usage: Returns the maximum 'Phi' value, as specified in the *AdiabatSetPhiMax*() method.
Parameters:
zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method
pdPhiMax – pointer to double variable
Returned parameter:
pdPhiMax – 'Phi' adiabatic value maximum (2.5 if not specified)
Return value: int – 0 = success, otherwise error

Model Execution and Results:

int AdiabatComputeCoordinateSet

(HANDLE zHandle,
char* szCoordSys,

char* szCoordUnits,
double* pvdTimes,
double* pvdCoord1,
double* pvdCoord2,
double* pvdCoord3,
int iNumTimes,
double* pvdPitchAngles,
int iNumPitchAngles,
double* pvvdAlpha,
double* pvvdLm,
double* pvvdK,
double* pvvdPhi,
double* pvvdHmin,
double* pvvdLstar,
double* pvdBmin,
double* pvdBlocal,
double* pvdMagLT,
double* pvvdB,
double* pvvdI)

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:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

szCoordSys – 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.

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

pvdTimes – array of time values, in Modified Julian Date form

pvdCoord1, pvdCoord2, pvdCoord3 –arrays of position values, in the coordinate system and units specified by the *szCoordSys* and *szCoordUnit* 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.

iNumTimes – number of values in the time and position arrays

pvdPitchAngles – 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.

iNumPitchAngles – number of pitch angles

pvvdAlpha, *pvvdLm*, *pvvdK*, *pvvdPhi*, *pvvdHmin*, *pvvdLstar* – pointers to double arrays, sized according to *iNumTimes* x *iNumPitchAngles*.

pvdBmin, *pvdBlocal*, *pvdMagLT* – pointers to double arrays, sized according to *iNumTimes*.

Returned parameters:

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

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

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

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

pvvdHmin – 2-dimensional array of adiabatic invariant ‘Hmin’ associated with the pitch angles at the ephemeris locations. [time,direction]

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

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

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

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

pvvdB – 2-dimensional array of the local magnetic field vector components at the ephemeris locations. [time,components]

pvvdI – 2-dimensional array of the local magnetic field current “I” value associated with the pitch angles at the ephemeris locations. [time,direction]

Return value: int – 0 = success, otherwise error

int AdiabatComputeCoordinateSetVarPitch

```
( HANDLE zHandle,
  char* szCoordSys,
  char* szCoordUnits,
  double* pvdTimes,
  double* pvdCoord1,
  double* pvdCoord2,
  double* pvdCoord3,
  int iNumTimes,
  double* pvvdPitchAngles,
  int iNumPitchAngles,
  double* pvvdAlpha,
  double* pvvdLm,
  double* pvvdK,
  double* pvvdPhi,
  double* pvvdHmin,
  double* pvvdLstar,
  double* pvdBmin,
  double* pvdBlocal,
  double* pvdMagLT,
  double* pvvdB,
  double* pvvdI )
```

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:

zHandle – ‘AdiabatModel’ object identifier, as returned from the AdiabatStartUp method

szCoordSys – 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.

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

pvdTimes – array of time values, in Modified Julian Date form

pvdCoord1, pvdCoord2, pvdCoord3 – arrays of position values, in the coordinate system and units specified by the *szCoordSys* and *szCoordUnit* 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.

iNumTimes – number of values in the time and position arrays

pvvdPitchAngles – 2-dimensional 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]

iNumPitchAngles – number of pitch angles for each time

pvvdAlpha, *pvvdLm*, *pvvdK*, *pvvdPhi*, *pvvdHmin*, *pvvdLstar* – pointers to double arrays, sized according to *iNumTimes* x *iNumPitchAngles*.

pvdBmin, *pvdBlocal*, *pvdMagLT* – pointers to double arrays, sized according to *iNumTimes*.

Returned parameters:

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

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

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

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

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

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

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

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

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

pvvdB – 2-dimensional array of the local magnetic field vector components at the ephemeris locations. [time,components]

pvvdI – 2-dimensional array of the local magnetic field current "I" value associated with the pitch angles at the ephemeris locations. [time,direction]

Return value: int – 0 = success, otherwise error

int AdiabatCalcDirPitchAngles

```
( HANDLE zHandle,
  char* szCoordSys,
  char* szCoordUnits,
  double* pvdTimes,
  double* pvdCoordX,
  double* pvdCoordY,
  double* pvdCoordZ,
  double* pvvdDirX,
  double* pvvdDirY,
```

double* pvvdDirZ,
int iNumTimes,
int iNumDirs,
double* pvvdPitchAngles)

Usage: Calculates the pitch angles corresponding to the input times, position and direction arrays.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

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

szCoordUnits – coordinate units, 'km' or 'Re'

pvdTimes – array of time values, in Modified Julian Date form

pvdCoordX, pvdCoordY, pvdCoordZ – arrays of position values, in the Cartesian coordinate system and units specified by the *szCoordSys* and *szCoordUnit* parameter values.

pvvdDirX, pvvdDirY, pvvdDirZ – 2-dimensional arrays of direction values, in the Cartesian coordinate system specified by the *szCoordSys* parameter value. These direction values may be full magnitude or unit arrays. [time,direction]

iNumTimes – number of values in the time and position arrays

iNumDirs – number of direction vectors for each time

pvvdPitchAngles – pointer to double array, sized according to *iNumTimes* x *iNumDirs.*

Returned parameter:

pvvdPitchAngles – 2-dimensional array of pitch angles corresponding to the input time, position and direction information. [time,direction]

Return value: int – 0 = success, otherwise error

int AdiabatConvertCoordinates

(HANDLE zHandle,
char* szCoordSys,
char* szCoordUnits,
double* pvdTimes,
double* pvdCoord1,
double* pvdCoord2,
double* pvdCoord3,
int iNumTimes,
char* szNewCoordSys,
char* szNewCoordUnits,
double* pvdNewCoord1,
double* pvdNewCoord2,
double* pvdNewCoord3)

Usage: Converts the set of input times, position coordinates from one coordinate system to another.

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

szCoordSys – 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.

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

pvdTimes – array of time values, in Modified Julian Date form

pvdCoord1, pvdCoord2, pvdCoord3 –arrays of position values, in the coordinate system and units specified by the *szCoordSys* and *szCoordUnit* 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.

iNumTimes – number of values in the time and position arrays

szNewCoordSys – '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.

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

pvdNewCoord1, pvdNewCoord2, pvdNewCoord3 – pointers to double arrays, each sized according to *iNumTimes.*

Returned parameters:

pvdNewCoord1, pvdNewCoord2, pvdNewCoord3 – arrays of position values, in the 'new' coordinate system and units specified by the *szNewCoordSys* and *szNewCoordUnit* parameter values.

Return value: int – 0 = success, otherwise error

int AdiabatConvertCoordinatesSingle

```
( HANDLE zHandle,
  char* szCoordSys,
  char* szCoordUnits,
  double dTime,
  double dCoord1,
  double dCoord2,
  double dCoord3,
  char* szNewCoordSys,
  char* szNewCoordUnits,
  double* pdNewCoord1,
  double* pdNewCoord2,
  double* pdNewCoord3 )
```

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

Parameters:

zHandle – 'AdiabatModel' object identifier, as returned from the AdiabatStartUp method

szCoordSys – 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.

szCoordUnits – '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 *szCoordSys* and *szCoordUnit* 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.

szNewCoordSys – '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.

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

dNewCoord1, dNewCoord2, dNewCoord3 – pointers to double variables

Returned parameters:

dNewCoord1, dNewCoord2, dNewCoord3 – position values, in the 'new' coordinate system and units specified by the *szNewCoordSys* and *szNewCoordUnit* parameter values.

Return value: int – 0 = success, otherwise error

RadEnvModel Class

Header file: CRadEnvModel_c.h

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:

HANDLE RadEnvStartUp

Usage: Instantiates a new 'RadEnvModel' object; multiple calls to this will generate separate instances, each of which may be accessed using the unique returned handle value.
Parameters: -none-
Return value: HANDLE - identifier value for the instantiated object

RadEnvShutDown

(HANDLE zHandle)
Usage: Destructor
Parameters:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
Return values: -none-

Model Parameter Inputs:

int RadEnvSetModel

(HANDLE zHandle,
char* szModel)
Usage: Specifies the name of the flux model to be used in the calculations.
Parameters:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
szModel – model name: 'AE8', 'AP8', 'CRRESELE' or 'CRRESPRO'.
Return value: int – 0 = success, otherwise error

int RadEnvGetModel

(HANDLE zHandle,
char* szModel)
Usage: Returns the name of the flux model being used, as specified in the *RadEnvSetModel*() method.
Parameters:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
szModel – pointer to character string (suggested sizing = 16)
Returned parameter:
szModel – model name.
Return value: int – 0 = success, otherwise error

int RadEnvSetModelDBDir

(HANDLE zHandle,

char* szDataDir)

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:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

szDataDir – directory path for the IRENE database files.

Return value: int – 0 = success, otherwise error

int RadEnvSetModelDBFile

(HANDLE zHandle,
char* szModelDBFile)

Usage: Specifies the name of the database file for legacy flux model calculations. The use of this method is *not needed* when *RadEnvSetModelDBDir*() 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:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

szModelDBFile – model database filename, including path.

Return value: int – 0 = success, otherwise error

int RadEnvSetMagfieldDBFile

(HANDLE zHandle,
char* szMagfieldDBFile)

Usage: Specifies the name of the file for the magnetic field model database. The use of this method is *not needed* when *RadEnvSetModelDBDir*() 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:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

szMagfieldDBFile – magnetic field model database filename, including path.

Return value: int – 0 = success, otherwise error

---

int RadEnvGetModelDBDir

(HANDLE zHandle,
char* szDataDir)

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

Parameters:

zHandle – 'RadEnv' object identifier, as returned from the RadEnvStartUp method

szDataDir – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataDir – model database directory.

Return value: int – 0 = success, otherwise error

int RadEnvGetModelDBFile

(HANDLE zHandle,
char* szModelDBFile)

Usage: Returns the name of the database file for legacy flux model calculations. This will be available immediately, when specified using the *RadEnvSetModelDBFile*() method. When the *AdiabatSetModelDBDir*() method is used, the automatically determined filename will be available after a call the *RadEnvComputeFlux*() method.

Parameters:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

szModelDBFile – pointer to character string (suggested sizing = 256)

Returned parameter:

szModelDBFile – model database filename, including path.

Return value: int – 0 = success, otherwise error

int RadEnvGetMagfieldDBFile

(HANDLE zHandle,
char* szMagfieldDBFile)

Usage: Returns the name of the file for the magnetic field model database. This will be available immediately, when specified using the *RadEnvSetMagfieldDBFile*() method. When the *RadEnvSetModelDBDir*() method is used, the automatically determined filename will be available after a call to the *RadEnvComputeFlux*() method.

Parameters:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

szMagfieldDBFile – pointer to character string (suggested sizing = 256)

Returned parameter:

szMagfieldDBFile – magnetic field model database filename, including path.

Return value: int – 0 = success, otherwise error

---

int RadEnvSetFluxType

(HANDLE zHandle,
char* szFluxType)

Usage: Specifies the type of flux values to be calculated by the model.

Parameters:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

szFluxType – flux type identifier: '1PtDiff'|'Differential'|'Diff' or 'Integral'

Return value: int – 0 = success, otherwise error

int RadEnvSetEnergies

(HANDLE zHandle,
double* pvdEnergies,
int iNumEnergies)

Usage: Specifies the set energies [MeV] at which the flux values are calculated by the selected model. Please consult User's Guide, RadEnvendix A for valid ranges/values, which depend on the model selected.

Parameters:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
pvdEnergies – array of energy values, in units of MeV
iNumEnergies – number of values in pvdEnergies array
Return value: int – 0 = success, otherwise error

int RadEnvSetActivityLevel

(HANDLE zHandle,
char* szActivityLevel)
Usage: Specifies the geomagnetic activity level parameter for the CRRESPRO, AE8 or AP8 models.
Parameters:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
szActivityLevel – geomagnetic activity level specification:
for CRRESPRO model, 'active' or 'quiet';
for AE8 or AP8 model, 'min' or 'max'.
Return value: int – 0 = success, otherwise error

int RadEnvSetActivityRange

(HANDLE zHandle,
char* szActivityRange)
Usage: Specifies the geomagnetic activity level parameter for the CRRESELE model. Only one of the *RadEnvSetActivityRange*() or *RadEnvSet15DayAvgAp*() methods may be used, otherwise an error is flagged.
Parameters:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
szActivityRange – 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: int – 0 = success, otherwise error

int RadEnvSet15DayAvgAp

(HANDLE zHandle,
double d15DayAvgAp)
Usage: Specifies the 15-day average Ap value for the CRRESELE model. Only one of the *RadEnvSetActivityRange*() or *RadEnvSet15DayAvgAp*() methods may be used, otherwise an error is flagged.
Parameters:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
d15DayAvgAp – 15-day average Ap value.
Return value: int – 0 = success, otherwise error

int RadEnvSetFixedEpoch

(HANDLE zHandle,
int iVerdict)
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:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

iVerdict – 1 *(true)* or 0 *(false)*; when *false*, the ephemeris year is used for the magnetic field model.
Return value: int – 0 = success, otherwise error

int RadEnvSetShiftSAA

(HANDLE zHandle,
int iVerdict)

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 ***RadEnvSetFixedEpoch*** method is set to 0 *(false)*.

Parameters:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
iVerdict – 1 *(true)* or 0 *(false)*.
Return value: int – 0 = success, otherwise error

---

int RadEnvGetFluxType

(HANDLE zHandle,
char* szFluxType)

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

Parameters:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
szFluxType – pointer to character string (suggested sizing = 32)
Returned parameter:
szFluxType – flux type identifier.
Return value: int – 0 = success, otherwise error

int RadEnvGetNumEnergies

(HANDLE zHandle)

Usage: Returns the number of currently defined energies.

Parameters:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
Return value: int – number of ephemeris entries, specified in the *RadEnvSetEnergies*() method.

int RadEnvGetActivityLevel

(HANDLE zHandle,
char* szActivityLevel)

Usage: Returns the geomagnetic activity level parameter, as specified in the *RadEnvSetActivityLevel*() method.

Parameters:
zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method
szActivityLevel – pointer to character string (suggested sizing = 16)
Returned parameter:
szActivityLevel – geomagnetic activity level specification.
Return value: int – 0 = success, otherwise error

int RadEnvGetActivityRange

(HANDLE zHandle,
char* szActivityRange)

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

Parameters:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

szActivityRange – pointer to character string (suggested sizing = 16)

Returned parameter:

szActivityRange – geomagnetic activity level specification, in terms of Ap values.

Return value: int – 0 = success, otherwise error

int RadEnvGet15DayAvgAp

(HANDLE zHandle,
double* pd15DayAvgAp)

Usage: Returns the 15-day average Ap value for the CRRESELE model, as specified in the *RadEnvSet15DayAvgAp*() method.

Parameters:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

p*d15DayAvgAp* – pointer to double variable

Returned parameter:

p*d15DayAvgAp* – 15-day average Ap value.

Return value: int – 0 = success, otherwise error

int RadEnvGetFixedEpoch

(HANDLE zHandle)

Usage: Returns the use of the model-specific fixed epoch (year) for the magnetic field model in the flux calculations, as specified in the *RadEnvSetFixedEpoch*() method.

Parameters:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

Return value: int – 1 *(true)* or 0 *(false)*, otherwise error

int RadEnvGetShiftSAA

(HANDLE zHandle)

Usage: Returns whether SAA shifting of epoch time has been enabled, as specified in the *RadEnvSetShiftSAA*() method.

Parameters:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

Return value: int – 1 *(true)* or 0 *(false)*, otherwise error

---

int RadEnvSetCoordSys

(HANDLE zHandle
char* szCoordSys,
char* szCoordUnits)

Usage: Specifies the coordinate system and units for the position values that are specified by the

RadEnvSetEphemeris() method. When not specified, these settings default to 'GEI' and 'Re'.
“Re” = radius of the Earth, defined as 6371.2 km.

Parameters:

zHandle – ‘RadEnvModel’ object identifier, as returned from the RadEnvStartUp method

szCoordSys – 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.

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

Return value: int – 0 = success, otherwise error

int RadEnvSetEphemeris

(HANDLE zHandle,
double* pvdTimes,
double* pvdCoords1,
double* pvdCoords2,
double* pvdCoords3,
int iNumTimes)

Usage: Specifies the ephemeris time and positions to be used for the model calculations.

Parameters:

zHandle – ‘RadEnvModel’ object identifier, as returned from the RadEnvStartUp method

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

pvdCoords1, *pvdCoords2*, *pvdCoords3* – arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system and units specified by *RadEnvSetCoordSys*().
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.

iNumTimes – number of values in all input arrays

Return value: int – 0 = success, otherwise error

---

int RadEnvGetCoordSys

(HANDLE zHandle
char* szCoordSys)

Usage: Returns the coordinate system for position values, as specified in the *RadEnvSetCoordSys*() method.

Parameters:

zHandle – ‘RadEnvModel’ object identifier, as returned from the RadEnvStartUp method

szCoordSys – pointer to character string (suggested sizing = 16)

Returned parameter:

szCoordSys – coordinate system identifier

Return value: int – 0 = success, otherwise error

int RadEnvGetCoordSysUnits

(HANDLE zHandle
char* szCoordUnits)

Usage: Returns the coordinate system units for position values, as specified in the

RadEnvSetCoordSys() method.

Parameters:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

szCoordUnits– pointer to character string (suggested sizing = 16)

Returned parameter:

szCoordUnits– coordinate system units

Return value: int – 0 = success, otherwise error

int RadEnvGetNumEphemeris

(HANDLE zHandle)

Usage: Returns the number of currently defined ephemeris time and position entries, as specified in the *RadEnvSetEphemeris*() method.

Parameters:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

Return value: int – number of ephemeris entries.

Model Execution and Results:

int RadEnvComputeFlux

(HANDLE zHandle,
double* pvvdFluxData)

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:

zHandle – 'RadEnvModel' object identifier, as returned from the RadEnvStartUp method

pvvdFluxData – pointer to double array, sized according to *RadEnvGetNumEphemeris*() results X *RadEnvGetNumEnergies*() results.

Returned parameter:

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

Return value: int – 0 = success, otherwise error

CammiceModel Class

Header file: CCammiceModel_c.h

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:

HANDLE CammiceStartUp

Usage: Instantiates a new 'CammiceModel' object; multiple calls to this will generate separate instances, each of which may be accessed using the unique returned handle value.
Parameters: -none-
Return value: HANDLE - identifier value for the instantiated object

CammiceShutDown

(HANDLE zHandle)
Usage: Destructor
Parameters:
zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method
Return values: -none-

Model Parameter Inputs:

int CammiceSetModelDBDir

(HANDLE zHandle,
char* szDataDir)
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:
zHandle – 'CammiceModel' object identifier, as returned from the CammiceStartUp method
szDataDir – directory path for the IRENE database files.
Return value: int – 0 = success, otherwise error

int CammiceSetModelDBFile

(HANDLE zHandle,
char* szModelDBFile)
Usage: Specifies the name of the database file for legacy flux model calculations. The use of this method is *not needed* when *CammiceSetModelDBDir*() 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:
zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method
szModelDBFile – model database filename, including path

Return value: int – 0 = success, otherwise error

int CammiceSetMagfieldDBFile

(HANDLE zHandle,
char* szMagfieldDBFile)

Usage: Specifies the name of the file for the magnetic field model database. The use of this method is *not needed* when *CammiceSetModelDBDir*() 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:

zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method

szMagfieldDBFile – magnetic field model database filename, including path

Return value: int – 0 = success, otherwise error

int CammiceSetMagfieldModel

(HANDLE zHandle,
char* szMFModel)

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:

zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method

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

Return value: int – 0 = success, otherwise error

int CammiceSetDataFilter

(HANDLE zHandle,
char* szDataFilter)

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:

zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method

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

Return value: int – 0 = success, otherwise error

int CammiceSetPitchAngleBin

(HANDLE zHandle,
char* szPitchAngleBin)

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

Parameters:

zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method

szPitchAngleBin – 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: int – 0 = success, otherwise error

int CammiceSetSpecies

(HANDLE zHandle,
char* szSpecies)

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

Parameters:

zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method

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

Return value: int – 0 = success, otherwise error

int CammiceSetCoordSys

(HANDLE zHandle
char* szCoordSys,
char* szCoordUnits)

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

Parameters:

zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method

szCoordSys – 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.

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

Return value: int – 0 = success, otherwise error

---

int CammiceGetModelDBDir

(HANDLE zHandle,
char* szDataDir)

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

Parameters:

zHandle – 'CammiceModel' object identifier, as returned from the CammiceStartUp method

szDataDir – pointer to character string (suggested sizing = 256)

Returned parameter:

szDataDir – model database directory.

Return value: int – 0 = success, otherwise error

int CammiceGetModelDBFile

(HANDLE zHandle,
char* szModelDBFile)

Usage: Returns the name of the database file for legacy flux model calculations. This will be available immediately, when specified using the *CammiceSetModelDBFile*() method. When the *CammiceSetModelDBDir*() method is used, the automatically determined filename will be available after a call to the *CammiceComputeFlux*() method.

Parameters:

zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method

szModelDBFile – pointer to character string (suggested sizing = 256)

Returned parameter:
szModelDBFile – model database filename, including path.
Return value: int – 0 = success, otherwise error

int CammiceGetMagfieldDBFile

(HANDLE zHandle,
char* szMagfieldDBFile)

Usage: Returns the name of the file for the magnetic field model database. This will be available immediately, when specified using the *CammiceSetMagfieldDBFile*() method. When the *CammiceSetModelDBDir*() method is used, the automatically determined filename will be available after a call to the *CammiceComputeFlux*() method.

Parameters:
zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method
szMagfieldDBFile – pointer to character string (suggested sizing = 256)
Returned parameter:
szMagfieldDBFile – magnetic field model database filename, including path.
Return value: int – 0 = success, otherwise error

int CammiceGetMagfieldModel

(HANDLE zHandle,
char* szMFModel)

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

Parameters:
zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method
szMFModel – pointer to character string (suggested sizing = 16)
Returned parameter:
szMFModel – magnetic field model specification.
Return value: int – 0 = success, otherwise error

int CammiceGetDataFilter

(HANDLE zHandle,
char* szDataFilter)

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

Parameters:
zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method
szDataFilter – pointer to character string (suggested sizing = 16)
Returned parameter:
szDataFilter – data filter specification.
Return value: int – 0 = success, otherwise error

int CammiceGetPitchAngleBin

(HANDLE zHandle,
char* szPitchAngleBin)

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

Parameters:
zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method
szPitchAngleBin – pointer to character string (suggested sizing = 16)
Returned parameter:
szPitchAngleBin – pitch angle bin identification.
Return value: int – 0 = success, otherwise error

int CammiceGetSpecies

(HANDLE zHandle,
char* szSpecies)

Usage: Returns the (single) particle species for the CAMMICE model run, as specified in the *CammiceSetSpecies*() method.
Parameters:
zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method
szSpecies – pointer to character string (suggested sizing = 16)
Returned parameter:
szSpecies – species identification.
Return value: int – 0 = success, otherwise error

int CammiceGetCoordSys

(HANDLE zHandle
char* szCoordSys)

Usage: Returns the coordinate system being used, as specified in the CammiceSetCoordSys() method.
Parameters:
zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method
szCoordSys – pointer to character string (suggested sizing = 16)
Returned parameter:
szCoordSys – coordinate system identifier.
Return value: int – 0 = success, otherwise error

int CammiceGetCoordSysUnits

(HANDLE zHandle
char* szCoordUnits)

Usage: Returns the coordinate system units being used, as specified in the *CammiceSetCoordSys*() method.
Parameters:
zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method
szCoordUnits – pointer to character string (suggested sizing = 16)
Returned parameter:
szCoordUnits – coordinate system units.
Return value: int – 0 = success, otherwise error

---

int CammiceSetEphemeris

(HANDLE zHandle,
double* pvdTimes,

double* pvdCoords1,
double* pvdCoords2,
double* pvdCoords3,
int iNumTimes)

Usage: Specifies the ephemeris time and positions to be used for the model calculations.

Parameters:

zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method

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

pvdCoords1, *pvdCoords2*, *pvdCoords3* – arrays of position coordinate values associated with times array. These position values are assumed to be in the coordinate system and units specified by *CammiceSetCoordSys()*.

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.

iNumTimes – number of values in pvdTimes and pvdCoords* arrays

Return value: int – 0 = success, otherwise error

int CammiceGetNumEphemeris

(HANDLE zHandle)

Usage: Returns the number of currently defined ephemeris time and position entries.

Parameters:

zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method

Return value: int – number of ephemeris entries, specified in the *CammiceSetEphemeris*() method.

Model Execution and Results:

int CammiceComputeFlux

(HANDLE zHandle,
double* pvvdFluxData)

Usage: Returns the mean model flux from the CAMMICE model, based on the various model parameter inputs, for the 12 fixed energy bins.

Parameters:

zHandle – '*CammiceModel*' object identifier, as returned from the *CammiceStartUp* method

pvvdFluxData – pointer to double array, sized according to *CammiceGetNumEphemeris*() results x 12.

Returned parameter:

pvvdFluxData – returned 2-dimensional array of the mean flux values. [time, energy]

Return value: int – 0 = success, otherwise error

DateTimeUtil Class

Header file: CDateTimeUtil_c.h

This class is the entry point that provides direct programmatic access to date and time conversion utilities.

Model Execution and Results:

double DateTimeGetGmtSeconds

(int iHours,
int iMinutes,
double 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: double – GMT seconds of day

int DateTimeGetDayOfYear

(int iYear,
int iMonth,
int 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: int – day number of year

double DateTimeGetModifiedJulianDate

(int iYear,
int iDdd,
double 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: double – Modified Julian Date (33282.0 - 69806.999)

double DateTimeGetModifiedJulianDateUnix

(int 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: double – Modified Julian Date (40587.0 - 65442.134)

int DateTimeGetDateTime

(double dModJulDate,
int* piYear,
int* piDdd,
double* pdGmtsec)

Usage: Determines the year, day of year and GMT seconds for the input Modified Julian Date.
Parameters:
dModJulDate – Modified Julian Date value (33282.0 - 69806.999)
piYear – *pointer* to returned year (1950-2049)
piDdd – *pointer* to returned day of year (1-365|366)
pdGmtsec – *pointer* to returned GMT seconds of day (0-86399.999)
Return value: int – 0 = success, otherwise error

int DateTimeGetHoursMinSec

(double dGmtsec,
int* piHours,
int* piMinutes,
double* pdSeconds)

Usage: Determines the hours, minutes and seconds for the input GMT seconds.
Parameters:
dGmtsec – GMT seconds of day (0-86399.999)
piHours – *pointer* to returned hours of day (0-23)
piMinutes – *pointer* to returned minutes of hour (0-59)
pdSeconds – *pointer* to returned seconds of minute (0-59.999)
Return value: int – 0 = success, otherwise error

int DateTimeGetMonthDay

(int iYear,
int iDdd,
int* piMonth,
int* piDay)

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)
piMonth – *pointer* to returned month (1-12)
piDay – *pointer* to returned day of month (1-28|29|30|31)
Return value: int – 0 = success, otherwise error

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