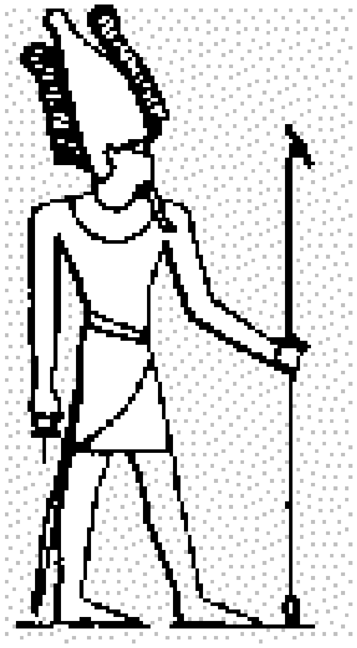

Odin-OSIRIS Level 1 Services

REFERENCE MANUAL

Version:	2.47
Date:	October 26, 2012
Author:	Nick Lloyd.

Page Index

Page Index 2
Odin-OSIRIS Level 1 API Reference 5
  Initialization Methods 5
  Attitude Database Methods 5
  Attitude Helper Methods 5
  Odin STW Conversion methods 5
  Measurement Database methods 6
  CDB Processing Functions 6
  Structures and Enumerations 6
OSIRIS Level 1 API Functions 7
  AngleToOrbitalPlane 8
  AngleToTangentHorizon 9
  BlankOutRadiationHitPixels 10
  CDBGen_Molecule_CrossSection 11
  CDBGen_OS_DarkCurrent 13
  CDBGen_OS_FlatFieldResponsivity 14
  CDBGen_OS_InternalScatter 16
  CDBGen_OS_PointSpread 17
  CDBGen_OS_PolarizedResponsivity 18
  CDBGen_OS_ReferenceSpectrum 20
  CDBGen_OS_Wavelength 22
  CDBGen_OS_XsectionFlatField 24
  CFToECI 25
  ECIPlanetaryBody 27
  ECIToGEO 28
  ECIToGeodetic 29
  ECItoOrbitalPlane 30
  GeodeticToECI 31
  GeodeticToGEO 32
  GEOToECI 33
  GEOToGeodetic 34
  GetAltitudeError 35
  GetCFUnitVector 37
  GetDataBetweenTimes 38
  GetFOVSize 40
  GetInstrumentXandYECI 41
  GetLevel1Version 43
  GetOdinPosition 44
  GetOdinVelocity 45
  GetOperatingMode [Not Properly Implemented] 46
  GetOrbitNumber 47
  GetOrbitStartAndEndTimes 48
  GetOsirisEcmwf 49
  GetOSSlitCurvature 50
  GetPixelCFUnitVector (Deprecated) 51
  GetPixelCFUnitVectorExt 52
  GetPlanetInFOV 54
  GetPlanetInInstrumentFOV 56
  GetScanData 58

GetScanDiagnostics 60
GetScanInfo 61
GetScanNumber 62
GetScanStartAndEndTimes 63
GetScienceProgram [Not properly Implemented] 64
GetSolarAngles 66
GetStarsInFOV 68
GetStarsInInstrumentFOV 70
GetUVPSF 72
InitializeLevel1Services 74
LoadMjdOS 75
LoadOrbitOS 76
LoadScanOS 78
LocateOrbitFile 79
LOSEntrancePoints 81
LOSTangentPoint 83
OrbitalPlanetoECI 84
OSIRISAvgTemperature 85
OSSolarAngles 86
OSTangentPoint 87
ReleaseScanData 88
StwLocateResetEpoch 89
StwToUtc 90
StwUsesFixedResetEpoch 91
UninitializeLevel1Services 92
OSIRIS LEVEL 1 API Structures 93
Spectrograph and IR Data Product History: Audit Field Values 94
Base Data Types 95
CFVECTOR, Spacecraft Control Frame 96
ECIVECTOR, Earth Centered Inertial Reference Frame 97
GEOVECTOR, Geographic Earth Centered Reference Frame 98
enum ECI_REFERENCE_FRAME 99
enum ODIN_INSTRUMENT 100
enum ODIN_POINTING_FRAME 101
enum PLANETARY_BODY 103
GEODETIC_COORD 104
IR_L1 105
Modified Julian Date 108
OS_L0 109
OS_L1 113
ODIN_SCAN_DIAGNOSTICS 118
ODIN_SCAN_ENTRY 120
OSIRIS_ECMWF 122
QUATERNION 124
ONYX_VERSION_STRUCT 125
OSIRIS LEVEL 1 API INFORMATION 126
Required Environment Variables 127
Software Installation 128
Software Usage 129
Odin/OSIRIS Level 1 Database Overview 130
Ephemeris Calculations 132
Source Code Control 134
Areas still requiring definition or clarification 135

Document Update History ... 136
Version 1.4 ... 136
Version 1.5 ... 136
Version 1.6 ... 136
Version 1.7 ... 136
Version 1.8 ... 136
Version 1.9 ... 137
Version 2.0 ... 137
Version 2.1 ... 137
Version 2.2 ... 137
Version 2.3 ... 137
Version 2.4 ... 137
Version 2.5 ... 137
Version 2.6 ... 137
Version 2.7 ... 138
Version 2.8 ... 138
Version 2.9 ... 138
Version 2.10 ... 138
Version 2.11 ... 138
Version 2.12 ... 138
Version 2.13 ... 138
Version 2.14 ... 138
Version 2.15 ... 138
Version 2.16 ... 138
Version 2.17 ... 138
Version 2.18 ... 139
Version 2.19 ... 139
Version 2.20 ... 139
Version 2.21 ... 139

Odin-OSIRIS Level 1 API Reference

Initialization Methods

GetLevel1Version	Fetch the version string of this code
InitializeLevel1Services	Initialize the OSIRIS level 1 Services
UninitializeLevel1Services	Uninitialize the OSIRIS level 1 Services

Attitude Database Methods

AngleToOrbitalPlane	Inclination of a vector to orbital plane.
AngleToTangentHorizon	Inclination of vector to tangent horizon
ECIPlanetaryBody	Get ECI position of a planetary body (Sun, Moon…)
GetCFUnitVector	Get instrument unit vector in Odin Control Frame
GetFOVSize	Get the size of an instrument’s Field of View
GetInstrumentXandYECI	Get instrument X and Y coordinate in ECI frame
GetOdinPosition	Get ECI location of Odin
GetOdinVelocity	Get ECI velocity of Odin
GetOrbitNumber	Get the satellite orbit number at a given UTC
GetOrbitStartAndEndTimes	Get the start and end times of a specific orbit
GetPixelCFUnitVector	Get pixel’s pointing vector in ODIN control Frame
GetPlanetInInstrumentFOV	See if a planet/moon is in an instrument’s field of view
GetPlanetInFOV	See if a planet/moon is in the field of view
GetScanNumber	Get the satellite scan number at any instant
GetScanInfo	Get information about a particular scan
GetScanStartAndEndTimes	Get the start and end time of a specific scan
GetSolarAngles	Get the appropriate SZA, SAA and SSA
GetStarsInFOV	Get the list of stars in arbitrary field of view
GetStarsInInstrumentFOV	Get stars in the instrument field of view
LOSEntrancePoints	Calculate LOS intersection points at given height
LOSTangentPoint	Calculate line-of-sight tangent point

Attitude Helper Methods

CFToECI	Convert a CF vector to an ECI vector
GetAttitudeError	Get altitude error in attitude solution.
ECIToGEO	Convert ECI to Geographic
ECIToGeodetic	Convert ECI to Geodetic coordinates
ECItoOrbitalPlane	Convert ECI cords to orbital plane coords
GeodeticToECI	Convert Geodetic coordinates to ECI coordinates
GeodeticToGEO	Convert Geodetic to Geographic
GEOToECI	Convert Geographic to ECI
GEOToGeodetic	Convert Geographic to Geodetic coordinates
OrbitalPlanetoECI	Convert orbital plane coords to ECI

Odin STW Conversion methods

StwLocateResetEpoch	Convert STW using a STW epoch derived from a given date.
StwToUtc	Convert the Odin STW to UTC
StwUsesFixedResetEpoch	Convert STW using a user-specified Odin STW epoch

Measurement Database methods

GetDataBetweenTimes	Get all level 1 data between two times
GetOperatingMode	Get an instrument's operational mode at a given time
GetScanData	Get all level 1 data for an altitude scan
GetOsirisEcmwf	Get the Ecmwf data for a requested mjd
GetScienceProgram	Get the Odin science program at a given time
LocateOrbitFile	Locates an orbit file on the users machine
ReleaseScanData	Release the collection acquired with **GetScanData** or **GetDataBetweenTimes**

CDB Processing Functions

CDBGen_Molecule_CrossSection	Get cross-section data for a molecule
CDBGen_OS_DarkCurrent	Get/apply Dark Current correction for OS spectra
CDBGen_OS_FlatFieldResponsivity	Get/apply non-polarized responsivity to OS spectra
CDBGEN_OS_InternalScatter	Get the value of gamma for internal scattering.
CDBGen_OS_PointSpread	Get the point spread function of each pixel
CDBGen_OS_PolarizedResponsivity	Get polarized responsivity for OS spectra
CDBGen_OS_ReferenceSpectrum	Get the OS reference spectrum
CDBGen_OS_Wavelength	Get wavelength of each pixel in OS spectra
CDBGEN_OS_XsectionFlatField	Get non-polarized responsivity for spatial column on CCD

Structures and Enumerations

Base Data Types	Description of the base data types
CFVECTOR	Spacecraft Control Frame coordinate
ECIVECTOR	Earth Centered Inertial Coordinate
enum ECI_REFERENCE_FRAME	Identifies different velocity reference frames
enum ODIN_INSTRUMENT	The list of instruments on Odin
enum ODIN_POINTING_FRAME	The list of pointing directions on Odin
enum PLANETARY_BODY	Planetary bodies in solar system
GEODETIC_COORD	A Geodetic coordinate
IR_L1	The OSIRIS IR level 1 data structure
Modified Julian Date	Modified Julian Date description
OS_L0	The OSIRIS OS level 0 data structure.
OS_L1	The OSIRIS OS level 1 data structure.
QUATERNION	Spacecraft rotation quaternion
ONYX_VERSION_STRUCT	Software versioning structure

OSIRIS Level 1 API Functions

AngleToOrbitalPlane

Retrieves the angle of *vector* to the ODIN orbital plane at the instant given by *mjd*.

HRESULT AngleToOrbitalPlane(
double *mjd*,
const ECIVECTOR* *vector,*
double* *angle*
);

```
IDL> angle  =  L1->ANGLETOORBITALPLANE( mjd, ecivector)
```

```
MAT> [angle] = AngleToOrbitalPlane(L1, mjd, vector );
```

Parameters

mjd
  The instant, expressed as a Modified Julian Date, at which the orbital plane should be determined.

vector
  The *vector* which is inclined to the orbital plane. It must be expressed in ECI.

angle
  Returns the inclination of vector to the orbital plane in degrees.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

AngleToTangentHorizon

Calculates the angle between *vector* and the “horizontal” horizon at the tangent point defined by *mjd* and *lookvector*.

HRESULT AngleToTangentHorizon(
double *mjd*,
const ECIVECTOR* *lookvector,*
const ECIVECTOR* *vector,*
double* *angle*
);

```
IDL> angle = L1->ANGLETOTANGENTHORIZON(mjd, lookvector, vector)
```

```
MATLAB> [angle] = AngleToTangentHorizon(L1, mjd, lookvector, vector )
```

Parameters

mjd
The instant, expressed as a Modified Julian Date, at which the tangent horizon should be determined.

lookvector
The look direction from the spacecraft position at the instant given by mjd. This is used to determine the tangent point. It must be expressed in ECI coordinates.

vector
The *vector* which is inclined to the tangent horizon. It must be expressed in ECI coordinates.

angle
Returns the inclination of vector to the tangent horizon in degrees.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

BlankOutRadiationHitPixels

Looks up all of the radiation hits for the given exposure and blanks out any data values and sets any flag

HRESULT BlankOutRadiationHitPixels (
OS_L1* *os*,
double* *data,*
nxBYTE* *flags*
);

```
IDL> N/A
```

```
MAT> [newflags, ok] = BlankOutRadiationHitPixels(L1, os, idx, flags)
```

Parameters

os

Pointer to the OS_L1 header of the desired exposure. The MATLAB version uses variable *idx* to index the specific exposure record.

data

Pointer to the array of data (only 1353x1 elements currently supported). The code will set the value of radiation hit pixels to zero. This parameter may be NULL.

flags

Pointer to the array of pixel exception flags (only 1353x1 elements currently supported). The code will set the OSPIX_FLAG_SEVERE and OSPIX_FLAG_RADIATIONHIT bits in each pixel with a radiation hit. This parameter may be NULL.

Returns

Return S_OK if success else E_FAIL or other COM error codes

Return to Odin Level 1 Services

CDBGen_Molecule_CrossSection

Retrieves the cross-section data for a given molecule as a function of wavelength and temperature. The function can retrieve the cross-section data either at full resolution or convolved with the spectrograph instrument profile.

```
HRESULT CDBGen_Molecule_CrossSection(
const char*         molecule,
nxBOOL              get_highres,
double**            xsection,
double**            wavelength,
double**            temperature,
int*                numwavelen,
int*                numtemperature
);
```

```
IDL> L1->CDBGEN_MOLECULE_CROSSSECTION,    molecule,
                                          get_highres,
                                          xsection,
                                          wavelength,
                                          temperature

MAT> [xsection,
      wavelength,
      temperature] = CDBGen_Molecule_CrossSection( L1,
                                                   molecule,
                                                   get_highres)
```

Parameters

molecule

The name of the molecule required. Valid names are “o3”, “no2”, “o4”, “bro”, and “oclo”. Other species may be added at a later date

get_highres

If nxTRUE then return a high resolution spectrum. If nxFALSE then return a spectrum that has been convolved with the instrument profile.

xsection

Returns a pointer to the molecules cross section data as a function of wavelength and temperature. The wavelength data extend from 270 nm to 820 nm in 0.1 nm intervals. The temperature range depends upon the species. The data are returned as a two dimensional array **double**(**numwavelen x numtemperature).** The data are organized as arrays of wavelength data. Each set of wavelength data is ***numwavelen** elements long. There are ***numtemperature** sets of wavelength data. The array is internally cached inside the Level1 services and is guaranteed to remain valid until the Level1 services are uninitialized. This parameter must not be NULL.

wavelength

Returns the wavelengths of the cross-section data. The data are returned as a pointer to an array of **double[*numwavelen].** The array contains the wavelength in nanometers of each point in the cross-section data. The array is internally cached inside the Level1 services and is guaranteed to remain valid until the Level1 services are uninitialized. This parameter must not be NULL.

temperature

Returns the temperature of each set of wavelength data in the cross-section array. The temperature data are returned as a pointer to an array of **double[*numtemperature].** The array contains the temperature in Kelvin of each set of wavelength data. The array is internally cached inside the Level1 services and is guaranteed to remain valid until the Level1 services are uninitialized. This parameter must not be NULL.

numwavelen

Returns the number of points in each set of wavelength data in the cross-section data. It is the "fastest" changing index for the two dimensional cross-section array. This parameter must not be NULL.

numtemperature

Returns the number of temperature points in the cross-section data. It is the "slowest" changing index for the two dimensional cross-section array. This parameter must not be NULL.

Returns

Returns E_FAIL

Return to Odin Level 1 Services

CDBGen_OS_DarkCurrent

Retrieves the dark current correction applicable for the given *spectrum. The dark current returned are the values subtracted from the Level 0 data product. I.E. the values include adjustments for the number of rows read out and for the exposure time. To obtain the dark current signal subtracted from the level 1 product (in level 1 units) you must divide these values by the number of readout rows (always 32 for normal scientific data) and by the exposure time. Then you must multiply by the flat-field correction.*

HRESULT CDBGen_OS_DarkCurrent(
OS_L1* *spectrum*,
double** *dcbuffer,*
double** *dcerror*
);

```
IDL> L1->CDBGEN_OS_DARKCURRENT, spectrum, dcbuffer, dcerror
```

```
MAT> [dcbuffer,dcerror] = CDBGen_OS_DarkCurrent(L1, spectrum, index)
```

Parameters

spectrum
Pointer to the OS data. Header information in *spectrum* is used to determine the appropriate dark current numbers associated with this spectrum. In Matlab spectrum is a 1x1 structure with fields. Each field is an array of numbers.

index (Matlab only)
Index into the desired element in each field array within the OS structure.

dcbuffer
Returns the dark-current data in DN as a pointer to a contiguous chunk of memory which is a 2D array of [`numcols` x `numrows`]. Where `numcols` and `numrows` are the fields in the **OS_L1** `spectrum` header. If there is an error it will return a `NULL` pointer. The pointer is guaranteed to remain valid until the next call to **CDBGen_OS_DarkCurrent** or until **UnInitializeLevel1Services** is called, whichever comes first. The returned dark current accounts for the exposure time and read-out binning.

dcerror
Same as *dcbuffer* except it points to the error in the estimate of the dark-current data in DN/second. The pointer may return NULL..

Returns

S_OK, All bits 0. Calibration properly executed
Bit 0 = 1. Dark current was extrapolated
Bit 1 = 2. Strap temperature was coarsely estimated
E_FAIL, internal error, calibration failed.

Return to Odin Level 1 Services

CDBGen_OS_FlatFieldResponsivity

Retrieves the non-polarized response applicable to the given OS *spectrum*. The non-polarized correction will convert the OS CCD A/D units expressed in DN/second/pixel to photons/(cm^2 nm s steradian)/DN.

The flat field data should be applied to data which has been corrected to DN/pixel/second, i.e. all on-chip and off-chip binning has been removed.

Flat field data is not available for 143 row or 286 row readout modes and the function will fail.

HRESULT CDBGen_OS_FlatFieldResponsivity (
OS_L1* *spectrum*,
double** *flatbuffer,*
double** *flaterror*
);

```
IDL> L1->CDBGEN_OS_FLATFIELDRESPONSIVITY, spectrum,
                                          flatbuffer,
                                          flaterror

MAT> [flatbuffer,
      flaterror] = CDBGen_OS_FlatFieldResponsivity( L1,spectrum,index)
```

Parameters

spectrum
Pointer to the OS spectrum. Header information in *spectrum* is used to determine the non-polarized response associated with this spectrum. In Matlab spectrum is a 1x1 structure with fields. Each field is an array of numbers.

index (Matlab only)
Index into the desired element in each field array within the OS structure.

flatbuffer
Returns the flat-field data as a pointer to a contiguous chunk of memory which is a 2D array of [`numcols` x `numrows`]. Where `numcols` and `numrows` are the fields in the **OS_L1** `spectrum` header. If there is an error it will return a `NULL` pointer. The pointer is guaranteed to remain valid until the next call to **CDBGen_OS_FlatFieldResponsivity** or until **UnInitializeLevel1Services** is called, whichever comes first.

flaterror
Same as *flatbuffer* except it points to the error in the estimate of the flat-field data. The pointer may return NULL..

Returns

Return S_OK if the flat field correction was generated by interpolating the absolute calibration table. Returns S_FALSE if a flat field correction was generated by extrapolating beyond the end of the absolute calibration table. Returns E_FAIL or other COM error codes if there are errors.

Returns

S_OK, All bits 0. Calibration properly executed
Bit 0 = 1. absolute calibration was extrapolated
E_FAIL, internal error, calibration failed.

Return to Odin Level 1 Services

CDBGen_OS_InternalScatter

Retrieves the value of gamma used for internal scattering.

HRESULT CDBGen_OS_InternalScatter (
double *mjd*,
double* *gamma,*
double* *gammaerr*
);

```
IDL> L1->CDBGEN_OS_InternalScatter, mjd, gamma, gammaerr
```

```
MAT> [gamma, gammaerr] = CDBGen_OS_InternalScatter(L1, mjd)
```

Parameters

mjd
The modified Julian date at which the value of gamma is required

gamma
Returns the value of gamma appropriate for the given.

gammaerr
Returns the error in gamma.

Returns

Return S_OK if success else E_FAIL or other COM error codes. The returned values of gamma and gammaerr are undefined if the routines fails.

Return to Odin Level 1 Services

CDBGen_OS_PointSpread

Retrieves the nominal full width at half-maximum (fwhm) value for the Gaussian point spread function of each pixel in the given *spectra*. The function is centred over the physical centre of each pixel and is expressed in nanometers. The level 1 data processing chain will guarantee that a change in the nominal point spread function will not occur during a satellite scan as this unduly complicates level 2 processing.

HRESULT CDBGen_OS_PointSpread(
OS_L1* *spectrum*,
double** *instpsf,*
double** *error,*
);

```
IDL> L1->CDBGEN_OS_POINTSPREAD, spectrum, instpsf, error
```

```
MAT> [instpsf,error] = CDBGen_OS_PointSpread(L1, spectrum, index)
```

Parameters

spectrum
Pointer to the OS spectrum. Header information in *spectrum* is used to determine the appropriate fwhm for each pixel in this spectrum. In Matlab spectrum is a 1x1 structure with fields. Each field is an array of numbers.

index (Matlab only)
Index into the desired element in each field array within the OS structure.

instpsf
Returns the *fwhm* in nanometers as a pointer to a contiguous chunk of memory which is a 2D array of [`numcols` x `numrows`]. Where `numcols` and `numrows` are the fields in the **OS_L1** `spectrum` header. If there is an error it will return a `NULL` pointer. The pointer is guaranteed to remain valid until the next call to **CDBGen_OS_PointSpread** or until **UnInitializeLevel1Services** is called, whichever comes first.

error
Same as *instpsf* except it points to the error in the estimate of the *fwhm*. The pointer may return NULL..

Returns

Returns S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

CDBGen_OS_PolarizedResponsivity

Retrieves the g_{12} and g_{13} polarization components applicable to the given OS *spectrum*. These coefficients are explained in the paper by McLinden et al. The parallel and perpendicular polarizations are defined as parallel and perpendicular to the spectrograph slit. The polarized response arrays are expressed as non dimensional units (between 0 and 1).

HRESULT CDBGen_OS_PolarizedResponsivity (
OS_L1* *spectrum*,
double** *g12,*
double** *g13,*
double** *g12error,*
double** *g13error,*
);

```
IDL> L1->CDBGEN_OS_POLARIZEDRESPONSIVITY, spectrum,
                                          g12,
                                          g13,
                                          g12error,
                                          g13error
```

```
MAT> [g12,
      g13,
      g12error,
      g13error] = CDBGen_OS_PolarizedResponsivity(L1, spectrum, index)
```

Parameters

spectrum
Pointer to the OS spectrum. Header information in *spectrum* is used to determine the polarized responsivities associated with this spectrum. In Matlab spectrum is a 1x1 structure with fields. Each field is an array of numbers.

index (Matlab only)
Index into the desired element in each field array within the OS structure.

g12
Returns the g12 polarized response as a pointer to a contiguous chunk of memory which is a 2D array of [`numcols` x `numrows`]. Where `numcols` and `numrows` are the fields in the **OS_L1** `spectrum` header. If there is an error it will return a `NULL` pointer. The pointer is guaranteed to remain valid until the next call to **CDBGen_OS_PolarizedResponsivity** or until **UnInitializeLevel1Services** is called, whichever comes first.

g13
Same as g12 except it returns the g13 responsivity.

g12error
Same as *g12* except it points to the error in the estimate of the g12 polarized responsivity data. The pointer may return NULL.

g13error

Same as *g12* except it points to the error in the estimate of the g13 polarized responsivity data. The pointer may return NULL..

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Reference

C.A. McLinden, J. C. McConnell, K. Strong, I. C. McDade, R. L. Gattinger, R. King, B. Solheim, E.J. Llewellyn, W.J.F. Evans: The impact of the OSIRIS grating efficiency on radiance and trace-gas retrievals, Can J. Phys. 78, 1-17, 2000

Return to Odin Level 1 Services

CDBGen_OS_ReferenceSpectrum

Retrieves the zero air mass, or top of the atmosphere. The user can choose to retrieve the solar spectrum convolved with the OSIRIS point spread function at the 1353 wavelengths of OSIRIS or a high resolution solar spectrum at 0.001 nm resolution from 250 nm to 830 nm (580001 data points).

NOTE THIS FUNCTION WAS CHANGED ON 2006-07-27 **

HRESULT CDBGen_OS_ReferenceSpectrum(
nxBOOL *get_highres,*
double** *wavelengths,*
double** *refspec,*
int* *numpts*
);

```
IDL> L1->CDBGEN_OS_REFERENCESPECTRUM,      get_highres,
                                           wavelengths,
                                           refspect
```

```
MAT>[refspect,wavelengths]= CDBGen_OS_ReferenceSpectrum(L1,get_highres)
```

Parameters

get_highres

If nxTRUE then return a high resolution spectrum. If nxFALSE then return a spectrum that has been convolved with the instrument profile.

wavelengths

Returns a pointer to an array of double[*numpts]. The array contains the wavelength in nanometers of each point in the reference spectrum. The array is internally cached inside the Level1 services and is guaranteed to remain valid until the next call to CDBGen_OS_ReferenceSpectrum. This parameter must not be NULL.

refspec

Returns a pointer to an array of double[*numpts]. The array contains the solar irradiance at the top of the atmosphere for each of the wavelength specified in array **wavelengths**. The spectrum is specified in units of photons/cm2/sec/steradian/nm. The array is internally cached inside the Level1 services and is guaranteed to remain valid until the next call to **CDBGen_OS_ReferenceSpectrum**. This parameter may not be NULL.

numpts

Returns the number of points the arrays pointed to **wavelengths** and **refspect**. The parameter will return 0 if there is an error in the function.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

References

Kurucz, R.L., The Solar Irradiance by Computation (see http://cfaku5.harvard.edu/
1997.

Return to Odin Level 1 Services

CDBGen_OS_Wavelength

Retrieves the nominal wavelength assignment of each pixel in the given *spectra*. The wavelength assignment will correspond to the physical centre of each pixel and is expressed in nanometers (nm). The algorithm may account for slit curvature on the CCD image due to optical aberration. It will not account for stretch and shift effects. The level 1 data processing chain will guarantee that a change in the nominal wavelength assignment will not occur during a satellite scan as this unduly complicates level 2 processing.

HRESULT CDBGen_OS_Wavelength(
OS_L1* *spectrum*,
double** *wavelengths,*
double** *widths,*
double** *waveerror*
);

```
IDL> L1->CDBGEN_OS_WAVELENGTH,      spectrum,
                                    wavelengths,
                                    widths,
                                    waveerror

MAT> [wavelengths,
      widths,
      waveerror] = CDBGen_OS_Wavelength(L1, spectrum, index)
```

Parameters

spectrum
Pointer to the OS spectrum. Header information in *spectrum* is used to determine the appropriate wavelength assignments for each pixel in this spectrum. The wavelength assignment corresponds to the centre of each pixel. In Matlab spectrum is a 1x1 structure with fields. Each field is an array of numbers.

index (Matlab only)
Index into the desired element in each field array within the OS structure.

wavelengths
Returns the wavelength of the center of each pixel in nanometers as a pointer to a contiguous array of doubles. Returns NULL if there is an error. The array is internally cached within the Level 1 Services and is valid until the next call to **CDBGen_OS_Wavelength**. The array is guaranteed to have the same size and dimensions as the data stored in *spectrum.* The parameter may be NULL in which case the data pointer is not returned.

widths
Returns the width of each pixel in nanometers as a pointer to a contiguous array of doubles. Returns NULL if there is an error. The array is internally cached within the Level 1 Services and is valid until the next call to **CDBGen_OS_Wavelength**. The array is guaranteed to have the same size and dimensions as the data stored in *spectrum.* The parameter may be NULL in which case the data pointer is not returned.

waveerror

Returns the error on the wavelength assignment of each pixel in nanometers as a pointer to a contiguous array of doubles. Returns NULL if there is an error. The array is internally cached within the Level 1 Services and is valid until the next call to **CDBGen_OS_Wavelength**. The array is guaranteed to have the same size and dimensions as the data stored in *spectrum.* The parameter may be NULL in which case the data pointer is not returned.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

CDBGen_OS_XsectionFlatField

Retrieves the non-polarized response applicable to a specific column of crosssection data in the OS_L1 structure. The non-polarized correction will convert the OS CCD A/D units expressed in DN/second/pixel to photons/(cm^2 nm s steradian)

Flat field cross-section data is not available for 143 row or 286 row readout modes and the function will fail.

HRESULT CDBGen_OS_XSectionFlatField (
OS_L1* *spectrum*,
int *pixelindex,*
double** *flatbuffer*
);

```
IDL> L1->CDBGEN_OS_XSECTIONFLATFIELD,     spectrum,
                                          pixelindex,
                                          flatbuffer
```

```
MAT> [flatbuffer] = CDBGen_OS_XSectionFlatField(L1, spectrum, index)
```

Parameters

spectrum

Pointer to the OS spectrum. Header information in *spectrum* is used to determine the non-polarized response associated with this spectrum. In Matlab spectrum is a 1x1 structure with fields. Each field is an array of numbers.

index (Matlab only)

Index into the desired element in each field array within the OS structure.

flatbuffer

Returns the flat-field data across the slit at the specified pixel index as a pointer to a contiguous chunk of memory. The memory is a 2D array of double [`roenumrows`]. The array size, `roenumrows`, is derived from the `roe` field in the OS_L1 header: `roe` 0, 1, 2 will have `roenumrows` set to 32, 16 or 8 respectively. The pointer is guaranteed to remain valid until the next call to **CDBGen_OS_FlatFieldResponsivity** or until **UnInitializeLevel1Services** is called, whichever comes first.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

CFToECI

Converts a unit vector specified in the spacecraft control frame to ECI reference frame. Algorithm uses the quaternions provided by SSC attitude solution to rotate from spacecraft control frame to ECI reference frame. Details on CFVECTOR and ECIVECTOR are located elsewhere in this document. The user must select the appropriate velocity reference frame. This choice applies an appropriate correction for light aberration between the spacecraft velocity and the velocity of the chosen reference frame. The light aberration correction is such that orthogonal vectors before the transform will not be orthogonal after the transform. The light aberration correction is of the order 0.1 arc minutes.

```
HRESULT CFToECI(
double                  mjd,
const CFVECTOR*         cf_vector,
ECIVECTOR*              eci_vector,
int                     refframe_id
);
```

```
IDL> eci_vector = L1->CFTOECI( mjd, cf_vector, reframe_id)

MAT> [eci_vector] = CFToECI(L1, mjd, cf_vector, reframe_id)
```

Parameters

mjd
The UTC time expressed as a Modified Julian Date for which the conversion is required.

cf_vector
Pointer to the vector in the spacecraft control frame. The data are read only.

eci_vector
Returns the vector specified in the ECI coordinate system.

refframe_id
Specifies the reference frame in which the attitude should be specified, chosen from **enum ECI_REFERENCE_FRAME**. This parameter is used to define the correction for light aberration. Most users will use the value of ECI_SPACECRAFT, which effectively disables the light aberration correction. Users requiring higher precision should use the following guidelines. Attitude vectors used in determining tangent altitudes, geodetic locations and ray paths should convert to reference frame ECI_TOPOCENTRIC. Vectors used for comparison against star catalogues and ephemerides should convert to reference frame ECI_GEOCENTRIC

Returns

Return S_OK if success. Returns S_FALSE if there was a problem in the light aberration correction. In this special case the attitude is that observed in the spacecraft frame which may be acceptable to the user is returned. All other errors return E_FAIL or other COM error codes and the pointing vector is undefined.

Return to Odin Level 1 Services

CreateOsirisEcmwfClimatologyInstance

Creates an skClimatology that uses the ECMWF values extracted and interpolated for the OSIRIS scans primary tangent point. The climatology supports pressures temperature and density.

HRESULT CreateOsirisEcmwfClimatologyInstance (
skClimatology** *osirisclimatology*
);

```
IDL> NotAvailable
```

```
MAT> Available as class @skClimatology_OsirisEcmwf
```

Parameters

osirisclimatology
Returns the desired instance of skClimatology that supports pressures density and temperature and is derived from interpolation of ECMWF to the OSIRIS scan. The user must call osirisclimatology->Release when finished with the object.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

ECIPlanetaryBody

Calculates the ECI position of the requested planetary body.

```
HRESULT ECIPlanetaryBody(
double                          mjd,
int                             planet_id,
ECIVECTOR*                      pos
);
```

```
IDL> pos = L1->ECIPLANETARYBODY( mjd, planet_id, /NORMALIZE)
```

```
MAT> [pos] = ECIPlanetaryBody(L1, mjd, planet_id)
```

Parameters

mjd
The UTC expressed as a Modified Julian Date at which the position of the planetary body is required.

planet_id
The id code of the required planetary body. The value is assumed to be chosen from **enum PLANETARY_BODY**.

pos
Returns the ECI position of the planetary body in meters. The vector is returned in the ECI_GEOCENTRIC velocity reference frame.

NORMALIZE
IDL Keyword. If true then return the position as a ECI unit vector.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

ECIToGEO

Converts from ECI coordinates to Geographic (X,YZ) coordinates, this simply involves a rotation about the Z axis based upon Sidereal time. Descriptions of ECIVECTOR and GEOVECTOR are given elsewhere in this document.

HRESULT ECIToGEO(
double *anmjd*,
const ECIVECTOR* *eci_position*
GEOVECTOR* *geo_position*,
);

```
IDL> geo_position = L1->ECIToGEO( anmjd, eci_position)
```

```
MAT> [geo_position] = ECIToGEO(L1, mjd, eci_position)
```

Parameters

anmjd
The UTC expressed as a Modified Julian Date associated with the point. This is required to calculate sidereal time.

eci_position
The location of a point expressed in ECI coordinates (meters).

geo_position
The location of the same point expressed in geographic coordinates

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

ECIToGeodetic

Converts the ECI coordinates of a point to geodetic coordinates. Descriptions of ECIVECTOR and GEODETIC_COORD are given elsewhere in this document.

HRESULT ECIToGeodetic(
double *anmjd*,
const ECIVECTOR* *eci_position*,
GEODETIC_COORD* *geodetic_pos*
);

```
IDL> geodetic_pos = L1->ECIToGeodetic( anmjd, eci_position )
```

```
MAT> [geodetic_pos] = ECIToGeodetic(L1, mjd, eci_position)
```

Parameters

anmjd
The UTC expressed as a Modified Julian Date associated with the point. This is required to calculate sidereal time.

eci_position
The location of a point expressed in ECI coordinates (meters).

geodetic_pos
The location of the same point expressed in geodetic coordinates (latitude, longitude and height in km).

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

ECItoOrbitalPlane

Converts the ECI coordinates of a point to orbital plane coordinates. The orbital plane coordinates are returned as a GEODETIC_COORD but the caller must be aware that they are not actual geodetic coordinates.

HRESULT ECItoOrbitalPlane(
double *anmjd*,
nxBOOL *updateplane*,
const ECIVECTOR* *eci_position*,
GEODETIC_COORD* *geodetic_pos*
);

```
IDL> geodetic_pos = L1->ECItoOrbitalPlane( anmjd, eci_position,
                                           updateplane=updateplane)

MAT> [geodetic_pos]= ECItoOrbitalPlane(L1, anmjd,
                                           updateplane,
                                           eci_position)
```

Parameters

anmjd
The UTC expressed as a Modified Julian Date used to calculate the orbital plane. This value is only used if *updatereferenceplane* is *true* or if the plane is internally undefined. The value is also used to determine the number of revolutions that have occurred since the epoch that defines the orbital plane.

updateplane
If nxTRUE then update the reference orbital plane using the ascending node prior to time *anmjd.* If nxFALSE then don't update the orbital plane. This is a keyword in the IDL implementation.

eci_position
The location of a point expressed in ECI coordinates (meters).

geodetic_pos
The location of the same point expressed in orbital plane coordinates: geocentric latitude, longitude and geocentric radius in km. In orbital plane coordinates, X points to the ascending node in the equatorial plane. Z is the spin axis of the orbit perpendicular to the orbital plane (towards sun for ODIN) and Y is perpendicular to X and Z. Longitude is the angular distance from the ascending node in degrees. Latitude is the angular distance in degrees from the orbital plane and height is the geocentric radius in kilometers. The longitude is adjusted to account for multiple revolutions as determined by *anmjd*

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GeodeticToECI

Converts the geodetic coordinates of a point to ECI coordinates. Descriptions of ECIVECTOR and GEODETIC_COORD are given elsewhere in this document.

HRESULT GeodeticToECI(
double *anmjd*,
const GEODETIC_COORD* *geodetic_pos*,
ECIVECTOR* *eci_position*
);

```
IDL> eci_position = L1->GeodeticToECI( anmjd, geodetic_pos )

MAT> [eci_position] = GeodeticToECI(L1, anmjd, geodetic_pos)
```

Parameters

anmjd
The UTC expressed as a Modified Julian Date associated with the point. This is required to calculate sidereal time.

geodetic_pos
The location of a point expressed in geodetic coordinates (latitude, longitude and height in km).

eci_position
The location of the same point expressed in ECI coordinates (meters).

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GeodeticToGEO

Converts the geodetic coordinates of a point to Geographic X,Y,Z coordinates. Descriptions of GEOVECTOR and GEODETIC_COORD are given elsewhere in this document.

HRESULT GeodeticToGEO(
const GEODETIC_COORD* *geodetic_pos*,
GEOVECTOR* *geo_position*
);

```
IDL> geo_position = L1->GeodeticToGEO( geodetic_pos )

MAT> [geo_position] = GeodeticToGEO(L1, geodetic_pos)
```

Parameters

geodetic_pos
The location of a point expressed in geodetic coordinates (latitude, longitude and height in km).

geo_position
The location of the same point expressed in Geographic coordinates (meters).

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GEOToECI

Converts from Geographic (X,Y,Z) coordinates to ECI coordinates, this simply involves a rotation about the Z axis based upon Sidereal time. Descriptions of ECIVECTOR and GEOVECTOR are given elsewhere in this document.

HRESULT GEOToECI(
double *anmjd*,
const GEOVECTOR* *geo_position*,
ECIVECTOR* *eci_position*
);

```
IDL> eci_position = L1->GEOToECI( anmjd, geo_position )
```

```
MAT> [eci_position] = GEOToECI(L1, anmjd, geo_position)
```

Parameters

anmjd
The UTC expressed as a Modified Julian Date associated with the point. This is required to calculate sidereal time.

geo_position
The location of the a point expressed in geographic coordinates

eci_position
The location of the same point expressed in ECI coordinates (meters).

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GEOToGeodetic

Converts Geographic (X,Y,Z) coordinates of a point to equivalent geodetic. Descriptions of GEOVECTOR and GEODETIC_COORD are given elsewhere in this document.

HRESULT GEOToGeodetic(
const GEOVECTOR* *geo_position*
GEODETIC_COORD* *geodetic_pos*,
);

```
IDL> geodetic_pos = L1->GEOToGeodetic( geo_position )

MAT> [geodetic_pos] = GEOToGeodetic(L1, geo_position)
```

Parameters

geo_position
The location of a point expressed in Geographic coordinates (meters).

geodetic_pos
The location of the same point expressed in geodetic coordinates (latitude, longitude and height in km).

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetAltitudeError

Retrieves the error in the altitude pointing of the attitude system in arc-minutes. The value is taken from the SSC attitude solution. This function is only available in Version 1.14 and later (i.e. installations after 2006-11-15).

HRESULT GetAltitudeError (
double *mjd*,
double* arcminute_error
);

```
IDL> arcminute_error = L1->GETALTITUDEERROR( mjd)

MAT> [arcminute_error] = GetAltitudeError(L1, mjd )
```

Parameters

mjd
The UTC time expressed as a Modified Julian Date at which the altitude error is required.

arcminute_error
Returns the error in the altitude direction of the attitude solution in arc minutes. One arc minute of angle corresponds to approximately 1 km of altitude at the tangent point for the OSIRIS geometry.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetCFUnitVector

Retrieves the unit vector of the specified instrument axis in the spacecraft Control Frame. The estimate is for the primary axis/boresight of the instrument and does not account for pointing vector adjustments for the individual pixels on instrument detectors.

HRESULT GetCFUnitVector(

double	*mjd*,
int	*odin_pointing_frame_id*,
CFVECTOR*	*att,*

);

```
IDL> att = L1->GETCFUNITVECTOR( mjd, odin_pointing_frame_id)

MAT> [att] = GetCFUnitVector(L1, mjd, odin_pointing_frame_id frame )
```

Parameters

mjd

The UTC time expressed as a Modified Julian Date at which the control frame unit vector is required.

odin_pointing_frame_id

An integer assumed to be selected from **enum ODIN_POINTING_FRAME** which identifies the required axis.

att

Returns the unit vector of the desired axis in the spacecraft control frame coordinate system. If there was an error then it sets all components to zero.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetDataBetweenTimes

Retrieve all of the scientific data for a specific instrument between the specified times. This function will retrieve all data between the specified times. The user may release the collection of data through a call to **ReleaseScanData().** The total size of virtual memory is the only limitation on the amount of data that may be held in memory at any instant.

HRESULT GetDataBetweenTimes(
double *start_mjd,*
double *end_mjd,*
int *instrument*,
const char* *levelstr*,
IOnyxCollection** *collection*
);

```
IDL> data =L1->GETDATABETWEENTIMES( start_mjd, end_mjd, instrument,
                                    levelstr,
                                    collection)

MAT> [data, collection] = GetDataBetweenTimes( L1,
                                               start_mjd,
                                               end_mjd,
                                               instrument,
                                               levelstr )
```

Parameters

start_mjd
The initial UTC expressed as a Modified Julian Date at which to collect data.

end_mjd
The final UTC expressed as a Modified Julian Date at which to collect data.

instrument
The number of the instrument for which data are requested. This number is assumed to be chosen from **enum ODIN_INSTRUMENT.**

levelstr
A string used to identify the level required. Typical examples are “1A” for level 1A and “1B” for level 1B. This string is used when the level 1 filename is calculated.

collection
Returns a pointer to a new IOnyxCollection object which contains all of the records between *start_mjd* and *end_mjd*. The records will be guaranteed to be in ascending order in time.

C/C++: The user must call **collection->Release()** or **ReleaseScanData()** to release memory and resources associated with the collection.

IDL and Matlab: this variable is an object that must be kept alive if the user plans to access any of the **IOnyxArray** data fields of the structure data using

IOnyxArrayGetData. The resources associated with this object should be released by calling
IDL> status = collection->Release()
MAT> status = Release(collection);

data

IDL: returns an array of structures extracted from the underlying collection.
Matlab: returns a 1x1 structure with a set of field arrays extracted from the underlying collection

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetFOVSize

Returns the nominal field of view for a given instrument. This does not attempt to account for details such as vignetting. Its primary goal is to provide a field of view for functions that need to locate nearby astronomical bodies such as the moon, planets and stars.

HRESULT GetFOVSize(
int *odin_instrument_id*,
double* *fovy,*
double* *fovz*
);

```
IDL> fov =L1->GETFOVSIZE( odin_instrument_id)

MAT> [fov] = GetFOVSize(L1, odin_instrument_id )
```

Parameters

odin_instrument_id
The number of the instrument for which data are requested. This number is assumed to be chosen from **enum ODIN_INSTRUMENT.**

fovy
Returns the nominal field of view in the instrument Y direction in degrees. The instrument Y direction is very close to the Odin Control Frame Y direction. The Y direction is normally directed from the spacecraft towards the ground. Y would normally be considered the height direction (going from high altitude to low altitude). Y is anti-parallel to the IR linear arrays. (IDL and Matlab): This is the second element of the returned array.

fovz
Returns the nominal field of view in the instrument Z direction in degrees. The instrument Z direction is very close to the Odin Control Frame Z direction. The CF Z direction is perpendicular to the solar panels and is directed away from the Sun. It is nominally anti-parallel to the OSIRIS slit commonly called the "spatial" direction. IDL and Matlab: This is the first element of the returned array

Returns

Return S_OK if success else E_FAIL or other COM error codes. IDL and Matlab return a double[2] array specifying the field of view.

Return to Odin Level 1 Services

GetInstrumentXandYECI

Returns the X and Y unit vectors of the specified instrument in the ECI coordinate frame. The instrument X axis is defined as the instrument boresight and is very close to the satellite Control Frame X axis. The Y axis is the instrument's natural Y axis: perpendicular to the instrument X axis and close to the satellite Control Frame Y axis. The routine may apply a correction for the aberration of light and can transform to any of the coordinate systems specified in refframe_id.

HRESULT GetInstrumentXandYECI(
double *mjd,*
int *odin_instrument_id*,
ECIVECTOR* *instrumentx*,
ECIVECTOR* *instrumenty*,
int *refframe_id*,
);

```
IDL> L1->GETINSTRUMENTXANDYECI,     mjd,
                                    odin_instrument_id,
                                    instrumentx,
                                    instrumenty,
                                    reframe_id

IDL> [instrumentx,
      instrumenty] = GetInstrumentXandYECI( L1,
                                            mjd,
                                            odin_instrument_id,
                                            reframe_id )
```

Parameters

mjd
The UTC expressed as a Modified Julian Date.

odin_instrument_id
The number that identifies the requested instrument. This value should be chosen from **enum ODIN_INSTRUMENT.**

instrumentx
Returns the instrument's X axis expressed as an ECI unit vector. If there is an error then the vector is set to [0,0,0]

instrumenty
Returns the instrument's Y axis expressed as an ECI unit vector. If there is an error then the vector is set to [0,0,0]. Please note that the X and Y vectors will generally not be perpendicular after the light aberration correction.

refframe_id
Specifies the reference frame of the ECI unit vectors. Most users will use ECI_SPACECRAFT. Precision calculations can choose from **enum ECI_REFERENCE_FRAME**. The value determines the correction for light aberration due to the spacecraft motion.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Comments

The instrument pointing vectors can also be retrieved using **GetCFUnitVector** followed by a call to CFToECI.

Return to Odin Level 1 Services

GetLevel1Version

HRESULT GetLevel1Version (
const char** version
);

```
IDL> version = L1->GETLEVEL1VERSION()

MAT> [version] = GetLevel1Version(L1)
```

Parameters

version

Returns a pointer to a NULL terminated string that describes the current version of the OSIRIS Level 1 services. The pointer will remain valid until a call to **UninitializeLevel1Services.**

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetOdinPosition

Fetch the best estimate ECI X,Y,Z position of Odin using any adjustments and/or corrections to initial SSC attitude solution. Our biggest concern at the present is whether SSC will remove outlier points from the GPS data set. We assume they will clean the position data before delivery to Level 1 processing. SSC guarantee an accuracy of xxx in the Odin ECI position.

HRESULT GetOdinPosition(
double *mjd,*
ECIVECTOR* *pos*
);

```
IDL> pos = L1->GETODINPOSITION(mjd)
```

```
MAT> [pos] = GetOdinPosition(L1, mjd)
```

Parameters

mjd

The time at which the position is required.

pos

Returns the ECI position of Odin in meters. If there is an error then it will set all coordinates to zero.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetOdinVelocity

Fetch the ECI X,Y,Z velocity of Odin using any adjustments and/or corrections to initial SSC attitude solution. SSC guarantee an accuracy of xxx in the Odin ECI position.

HRESULT GetOdinVelocity(
double mjd,
ECIVECTOR* v
);

```
IDL> v = L1->GETODINVELOCITY( mjd)
```

```
MAT> [v] = GetOdinVelocity(L1, mjd)
```

Parameters

mjd
  The UTC at which the position is required. Expressed as a Modified Julian date.

v
  Returns the ECI velocity of Odin in meters per second. If there is an error then it will set all coordinates to zero.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetOperatingMode [Not Properly Implemented]

Fetches the science operating mode of a specific instrument at a specific instant. For OSIRIS this value is identical to the *scienceprog* field in the OS_L1 and IR_L1 structures.

```
HRESULT GetOperatingMode(
double          mjd,
int             instrument,
int*            mode_id,
double*         start_time,
double*         end_time
);
```

Parameters

mjd
: The UTC at which the mode is required. Expressed as a Modified Julian date.

instrument
: The id code of the required instrument. The code only guarantees support for OSIRIS detectors. The value is assumed to be selected from **enum ODIN_INSTRUMENT.**

mode_id
: Receives the id code of the operating mode active on the specified instrument at the specified time. If there is an error then this value will be set to -1. May be NULL.

start_time
: Returns the UTC start time of the specified operating mode. Expressed as a Modified Julian Date.

end_time
: Returns the UTC end time of the specified operating mode. Expressed as a Modified Julian Date.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetOrbitNumber

Fetch the Odin orbit number for the specified time.

HRESULT GetOrbitNumber(
double *mjd*,
int* *n*
);

```
IDL> n = L1->GetOrbitNumber( mjd )
```

```
MAT> n = GetOrbitNumber(L1,mjd)
```

Parameters

mjd

The UTC time at which the orbit number is required. Expressed as a Modified Julian Date.

n

Returns the orbit number. If no orbit number is valid at the specified time then returns -1.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetOrbitStartAndEndTimes

Fetch the start and end time of the specified orbit.

```
HRESULT GetOrbitStartAndEndTimes(
int                 orbitnumber,
double*             starttime,
double*             endtime
);
```

```
IDL> L1->GETORBITSTARTANDENDTIMES,  orbitnumber,
                                    starttime,
                                    endtime

MAT> [starttime,
      endtime] = GetOrbitStartAndEndTimes( L1, orbitnumber)
```

Parameters

orbitnumber
The specified orbit number.

starttime
returns the UTC start time of this orbit. Expressed as a Modified Julian Date. This is defined as the ascending node of the orbit; i.e. the northward crossing of the equator, identified by the ECI Z component of the spacecraft position changing from negative to positive. If there is an error then this will return 0.0

endtime
Returns the UTC end time of the orbit. Expressed as a Modified Julian Date. If there is an error then this will return 0.0

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetOsirisEcmwf

Retrieve the OSIRIS_ECMWF structure corresponding to the specified mjd. This structure contains height profiles of temperature and density from the ground to 70 km at the tangent point when the spacecraft was looking at 30 km altitude. The OSIRIS_ECMWF structures are nominally available only for UP/DOWN aeronomy scans. The user must check that the returned structure is close enough in time to the requested mjd for their purposes. It is possible under dysfunctional conditions that the call may succeed but the returned time is several hours from that requested. The user must also check for missing data as the ECMWF model frequently does not cover the entire region from ground to 70 km. Missing values of temperature or density are indicated by a negative value (-9999999.0). The density is in molecules/cm^3 and the temperature is in Kelvin.

HRESULT GetOsirisEcmwf(
double *mjd,*
OSIRIS_ECMWF** ecmwf*,*
);

```
IDL> ecmwf =L1->GETOSIRISECMWF( mjd, altitude, density,temperature )

MAT> [ecmwf, altitude, density, temperature] = GetOsirisEcmwf(L1, mjd)
```

Parameters

mjd

The UTC expressed as a Modified Julian Date at which ECMWF data are requested.

ecmwf

Returns a pointer to the OSIRIS_ECMWF structure for the requested mjd. May return NULL. User must check that the structure is appropriate for their requested time.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Note:
The IDL version returns the altitude, density and temperature as arrays rather than as part of the structure
Return to Odin Level 1 Services

GetOSSlitCurvature

Retrieves the offset of the spectrograph slit in spatial pixels at a specific wavelength index. The offset of the slit is set so it is close to 0 near to wavelength index 730. This function is provided for users who must account for the off-axis nature of pixels on a detector.

HRESULT GetOSSlitCurvature(
double *wavelength_Index*,
double* *offset*
);

```
IDL> offset =L1->GETOSSLITCURVATURE( wavelength_index)
```

```
MAT> [offset] = GetOSSlitCurvature(L1, wavelength_index)
```

Parameters

wavelength_index
A value between 0 and 1353 used to index the wavelength. A value of -9999 can be used to force the code to return an offset value of 0.0.

offset
The offset of the slit in the spatial (along the slit) direction at this wavelength. It is specified in spatial pixels. This value should be added to the spatial position at wavelength index 730 to get the true pointing at the specified wavelength.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetPixelCFUnitVector (Deprecated)

This function has been replaced by GetPixelCFUnitVectorExt. This function is provided for users who must account for the off-axis nature of pixels on a detector.

HRESULT GetPixelCFUnitVector(

double	*mjd*,
int	*odin_instrument_id*,
int	*column*,
int	*row*,
CFVECTOR*	*v*

);

```
IDL> NOT AVAILABLE
MAT> NOT AVAILABLE
```

Parameters

mjd

The UTC time expressed as a Modified Julian date at which the control frame unit vector is required.

odin_instrument_id

An integer identifying the required detector. This number is assumed to be chosen from **enum ODIN_INSTRUMENT**.

column

The zero-based, column (x coord) of the pixel on the desired instrument. This value is ignored for IR channels. The value is between 0 and 31 for the spectrograph and corresponds to the centre of the specified pixel. This direction is approximately parallel to the "spatial" direction of the CCD.

row

The zero-based, row (y coord) of the pixel on the desired instrument. This value is ignored for the spectrograph. The value is between 0 and 127 for the IR channels. This direction is approximately parallel to the "height" direction. Note that the function returns the pointing at the center of the specified pixel.

v

Returns the unit vector of the desired axis in the spacecraft control frame coordinate system. If there was an error then it sets all components to zero.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetPixelCFUnitVectorExt

Retrieves the unit vector of the pointing/viewing of a specified pixel on the specified instrument. Returns the coordinates in the Odin Control Frame.

HRESULT GetPixelCFUnitVectorExt(
double *mjd*,
int *odin_instrument_id*,
double *column*,
double *row*,
CFVECTOR* *v*
);

```
IDL> v =L1->GETPIXELCFUNITVECTOREXT( mjd,
                                     odin_instrument_id,
                                     column,
                                     row)
```

```
MAT> [v] = GetPixelCFUnitVectorExt( L1,
                                    mjd,
                                    odin_instrument_id,
                                    column,
                                    row)
```

Parameters

mjd
The UTC time expressed as a Modified Julian date at which the control frame unit vector is required.

odin_instrument_id
An integer identifying the required detector. This number is assumed to be chosen from **enum ODIN_INSTRUMENT**.

column
This value is between 0.00 and 32.00 for the spectrograph spatial direction parallel to the spectrograph slit. Note that the user must enter the "0.5" to get the centre of a spatial pixel e.g. enter 0.5 for the centre of pixel 0 and 31.5 for the centre of pixel 31. This value is ignored for IR channels.

row
This value is interpreted as the wavelength pixel (0-1353) for the spectrograph and is used to correct for the curvature of the slit imaged onto the CCD as a function of wavelength, enter a value of -9999 if you do not wish to correct for slit curvature. The value is between 0.00 and 128.00 for the IR channels. Note that the user must enter a value of 0.5 to get the centre of IR pixel 0 and 127.5 to get the center of IR pixel 127.

v
Returns the unit vector of the desired axis in the spacecraft control frame coordinate system. If there was an error then it sets all components to zero.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetPlanetInFOV

Determines whether a planetary body is within a specified field of view. The code determines whether any portion of the planetary body's disk is within the field of view.

HRESULT GetPlanetInFOV(
double *mjd*,
const ECIVECTOR* , *ecibore,*
const ECIVECTOR* *eciinsy*,
double *fovy*,
double *fovz*,
int *bodyid*,
nxBOOL* *infieldofview*,
double* *angulardistance*
);

```
IDL> infieldofview = L1->GETPLANETINFOV(  mjd,
                                          ecibore,
                                          eciinsy,
                                          fovy,
                                          fovz,
                                          bodyid,
                                          angulardistance)

MAT> [infieldofview,
      angulardistance] = GetPlanetInFOV(  L1,
                                          mjd,
                                          ecibore,
                                          eciinsy,
                                          fovy,
                                          fovz,
                                          bodyid)
```

Parameters

mjd
The UTC expressed as a Modified Julian Date.

ecibore
The look-direction specified in ECI coordinates. The direction corresponds to the center of the instrument's field of view.

eciinsy
The instrument's Y axis, unit vector specified in the ECI coordinate system.

fovy
The instrument's Y-axis, full, angular width (edge to edge) specified in degrees. If **fovz** is less than or equal to 0.0 then **fovy** specifies the angular diameter of a circular field of view.

fovz
The instrument's Z axis, full, angular width (edge to edge) specified in degrees. If **fovz** is less than or equal to 0.0 then the field of view is assumed to be circular.

bodyid

The id code of the requested planetary body. It is assumed the value is chosen from enum PLANETARY_BODY.

infieldofview

Returns nxTRUE if the centre of the object is within the field of view. Users should be aware that large objects like the moon may actually have parts within the field of view even if the centre of the moon isn't.

angulardistance

Returns the angular distance, in degrees, of the centre of the planetary object from the specified **boresight** look vector.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Comments

The user should note that the function indicates that the planetary body is inside the field of view even if the portion of the body's disk is not illuminated by sunlight. This is of particular importance for the moon which will normally have only 50% of the disk illuminated for a dawn dusk orbit. The code corrects for light aberration due to the motion of the satellite and also accounts for the topocentric location of the satellite. The overall accuracy of the code is better than 1 arc second.

Return to Odin Level 1 Services

GetPlanetInInstrumentFOV

Determines whether a planetary body is within a specified field of view. The code determines whether any portion of the planetary body's disk is within the field of view.

HRESULT GetPlanetInInstrumentFOV(
double *mjd*,
int *odin_instrument_id*,
int *pointingaccuracy*,
int *bodyid*,
nxBOOL* *infieldofview*,
double* *angulardistance*
);

```
IDL> infieldofview = L1->GETPLANETININSTRUMENTFOV(mjd,
                                                odin_instrument_id,
                                                pointingaccuracy,
                                                bodyid,
                                                angulardistance)

MAT> [infieldofview,
      angulardistance] = GetPlanetInInstrumentFOV( L1,
                                                   mjd,
                                                   odin_instrument_id,
                                                   pointingaccuracy,
                                                   bodyid)
```

Parameters

mjd
The UTC expressed as a Modified Julian Date.

odin_instrument_id
The id of the required instrument. This value should be selected from enum ODIN_INSTRUMENT.

pointingaccuracy
The overall pointing accuracy of the Odin satellite expressed in degrees. This value is added to the instrument's intrinsic field of view. Hence the value returned indicates whether the specified planet is anywhere in the potential field of view rather than the exact field of view.

bodyid
The id code of the requested planetary body. It is assumed the value is chosen from enum PLANETARY_BODY.

infieldofview
Returns nxTRUE if the centre of the object is within the field of view. Users should be aware that large objects like the moon may actually have parts within the field of view even if the centre of the moon isn't.

angulardistance

Returns the angular distance, in degrees, of the centre of the planetary object from the instrument's **boresight** look vector.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Comments

The user should note that the function will indicate the planetary body is inside the field of view even if the portion of the body's disk is not illuminated by sunlight. This is of particular importance for the moon which will normally have only 50% of the disk illuminated for a dawn dusk orbit. The code corrects for light aberration due to the motion of the satellite and also accounts for the topocentric location of the satellite. The overall accuracy of the code is better than 1 arc second.

Return to Odin Level 1 Services

GetScanData

Retrieve all of the data for a specific scan for a specific instrument. The user is responsible for selecting records of interest within the scan. The user may release the scan of data through a call to **ReleaseScanData().** The total size of virtual memory is the only limitation on the number of scans that may be held in memory at any instant. GetScanData will properly handle scans that straddle file boundaries.

HRESULT GetScanData(

int	*scannumber,*
int	*instrument*,
const char*	*levelstr*,
IOnyxCollection**	*collection*

);

```
IDL> data = L1->GETSCANDATA(  scannumber,
                              instrument,
                              levelstr,
                              collection)

IDL> [data,
      collection] = GetScanData( L1,
                                 scannumber,
                                 instrumentid,
                                 levelstr )
```

Parameters

scannumber
The number of the desired scan.

instrument
The instrument for which data are requested. It is assumed the value is chosen from **enum ODIN_INSTRUMENT**. Only OSIRIS detectors are guaranteed to be supported by this algorithm.

levelstr
A string used to identify the level required. Typical examples are "1A" for level 1A and "1B" for level 1B. This string is used when level 1 filenames are generated.

collection
Returns a pointer to a new IOnyxCollection object which contains all of the records associated with the scan. The records will be guaranteed to be in ascending order in time.

C/C++: The user must call **collection->Release()** or **ReleaseScanData()** to release memory and resources associated with the collection.

IDL and Matlab: this variable is an object that must be kept alive if the user plans to access any of the **IOnyxArray** data fields of the structure data using IOnyArrayGetData. The resources associated with this object should be released by calling,
IDL> status = collection->Release()

MAT> status = Release(collection);

data

IDL: returns an array of structures extracted from the underlying collection.
Matlab: Returns a 1x1 structure with fields that are arrays extracted from the underlying collection

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetScanDiagnostics

Fetch the diagnostic information for the given scan.

HRESULT GetScanDiagnostics(
int *scannumber,*
const ODIN_SCAN_DIAGNOSTICS** *scandiagnostics,*
);

```
MAT> [scandiagnostics] = GetScanDiagnostics(L1, scannumber)
```

Parameters

scannumber
The number of the scan.

scandiagnostics
Returns a pointer to the current scan diagnostics. The pointer is valid until the next call to any OSIRIS Level1 API functions. The function will return a NULL if there is no information available for the specified scan.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

History

First implemented in version 1.17, October 2, 2008.
Return to Odin Level 1 Services

GetScanInfo

Fetch the auxiliary information for the given scan.

HRESULT GetScanInfo(
int *scannumber,*
const ODIN_SCAN_ENTRY** *scaninfo,*
);

```
IDL> scaninfo = L1->GETSCANINFO( scannumber )

MAT> [scaninfo] = GetScanInfo(L1, scannumber)
```

Parameters

scannumber
The number of the scan.

scaninfo
Returns a pointer to the current scan info. The pointer is valid until the next call to any OSIRIS Level1 API functions. The function will return a NULL if there is no information available for the specified scan.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

History

A bug was found in this function that caused an access violation when accessing orbits with no scan data. This bug was fixed in software version 1.06 released on 2003-06-27.

Return to Odin Level 1 Services

GetScanNumber

Fetch the Odin scan number at any given instant during the mission.

HRESULT GetScanNumber(
double *mjd*,
int* *n*
);

```
IDL> n =L1->GETSCANNUMBER( mjd)
```

```
MAT> [n] = GetScanNumber(L1, mjd)
```

Parameters

mjd
: The UTC for which the scan number is required. Expressed as a Modified Julian Date

n
: Returns the associated scan number. If there is an error then it returns -1;

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetScanStartAndEndTimes

Fetch the start and end time of the specified scan.

HRESULT GetScanStartAndEndTimes(
int *scannumber*,
double* *starttime*,
double* *endtime*
);

```
IDL> L1->GETSCANSTARTANDENDTIMES,   scannumber,
                                    starttime,
                                    endtime
```

```
MAT> [starttime,
      endtime]  = GetScanStartAndEndTimes(L1, scannumber)
```

Parameters

orbitnumber
The specified scan number.

starttime
Returns the UTC start time of the scan expressed as a Modified Julian Date. If there is an error then this will return as 0.0

endtime
Returns the UTC end time of the scan expressed as a Modified Julian Date. If there is an error this will return as 0.0

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetScienceProgram [Not properly Implemented]

Fetches a string describing the overall Odin science program in effect at any given instant.

HRESULT GetScienceProgram(
double *mjd,*
const char** *science_program*
);

```
IDL> science_program =L1->GETSCIENCEPROGRAM(mjd)
```

```
MAT> [science_program] = GetScienceProgram(L1, mjd)
```

Parameters

mjd

The UTC at which the mode is required. Expressed as a Modified Julian date.

science_program

Returns a pointer to a zero terminated string describing the Odin science program in effect at the specified time. It is strongly recommended that the user copy this string to a local buffer as soon as possible as the pointer will become invalid once the Odin level 1 services database is closed down.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetScanTropopauseEntry

Fetch a tropopause definition structure for the given scan.

HRESULT GetScanTropopauseEntry (
int *scannumber*,
ODIN_SCAN_TROPOPAUSE* *entry*
);

```
IDL> Not available

MAT> [thermaltrop,
      dynamictropo,
      thetatropo,
      latitude,
      longitude,
      mjd]  = GetScanTropopauseEntry(L1, scannumber)
```

Parameters

scannumber
: The specified scan number.

entry
: Returns the ODIN_SCAN_TROPOPAUSE for the requested scan. The structure contains the thermal , dynamic and potential temperature (theta) tropopause heights as derived from NCEP. If there is an error then these fields will return as negative values (e.g. -9999.0)

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetScanV507AlbedoEntry

Fetch the V507 processing albedo structure for the given scan.

HRESULT GetScanV507AlbedoEntry(
int *scannumber*,
ODIN_SCAN_V507ALBEDO* *entry*
);

```
IDL> Not available
```

```
MAT> Not Available
```

Parameters

scannumber
The specified scan number.

entry
Returns the ODIN_SCAN_V507Albedo for the requested scan. The structure contains the version and albedo calculated in the Version 507 Level 2 processing. This is often used as an input the future versions of processing. If there is an error then these fields will return as negative values (e.g. -9999.0)

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetSolarAngles

Returns solar angles appropriate to the given tangent point location and look-direction unit vector.

HRESULT GetSolarAngles(	
double	*mjd*,
const ECIVECTOR*	*location*,
const ECIVECTOR*	*lookvector*,
double*	*sza*,
double*	*saa*,
double*	*ssa*
);	

```
IDL> L1->GETSOLARANGLES, mjd, location, lookvector, sza, saa, ssa
```

```
MAT> [sza, saa, ssa] = GetSolarAngles(L1, mjd, location, lookvector)
```

Parameters

mjd
The UTC expressed as a Modified Julian Date.

location
The tangent point location specified in the ECI coordinate system.

lookvector
The boresight, look-direction unit vector specified in the ECI coordinate system. It is assumed that this vector corresponds to the centre of the instrument's field of view. Note that the optic axis for the IR channels is not centred on the detector.

sza
The solar zenith angle in degrees at *location*, defined as: 0 is overhead, 90 is on the horizon and 180 is directly below.

saa
The solar azimuth angle in degrees at *location*, defined as: 0 is due North, 90 is due East, 180 is due South and 270 is due West.

ssa
The scattering angle in degrees defined by the (dot product) angle between *lookvector* and a vector from *location* to the Sun. The scattering angle is defined as: 0 is straight through (forward-scatter), 90 is right angle scatter and 180 is back-scatter.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetStarsInFOV

Returns a list of stars within the specified field of view. The algorithm transforms the instrument field of view from the ECI coordinate system to the J2000/FK5 standard epoch. The accuracy of the transformation is better than 1 arc second and includes a relativistic correction for the aberration of light due to the spacecraft velocity.

HRESULT GetStarsInFOV(
double *mjd,*
const ECIVECTOR* *ecibore*,
const ECIVECTOR* *eciinsy*,
double *fovy*,
double *fovz*,
double *thresholdmag*,
STAR_CATALOGUE_ENTRY* *stars*,
int* *maxstars,*
);

```
IDL> stars =L1->GETSTARSINFOV(      mjd,
                                    ecibore,
                                    eciinsy,
                                    fovy,
                                    fovz,
                                    thresholdmag)
```

```
MAT> [stars] = GetStarsInFOV( L1,
                              mjd,
                              ecibore,
                              eciinsy,
                              fovy,
                              fovz,
                              thresholdmag)
```

Parameters

mjd
The UTC expressed as a Modified Julian Date.

ecibore
The boresight, look-direction unit vector specified in the ECI coordinate system. It is assumed that this vector corresponds to the centre of the instruments field of view. Note that the optic axis for the IR channels is not centred on the detector.

eciinsy
The instrument's Y axis, unit vector specified in the ECI coordinate system.

fovy
The angular, full width, field of view along the instrument's Y axis specified in degrees. If *fovz* is negative or zero then the field of view is assumed to be circular with angular diameter given by *fovy*. Note that the user may want to over-estimate the field of view to account for errors in the satellite attitude determination.

fovz

The angular, full width, field of view along the instrument's Y axis specified in degrees. If *fovz* is negative or zero then the field of view is assumed to be circular with angular diameter given by *fovy*. Note that the user may want to over-estimate the field of view to account for errors in the satellite attitude determination.

thresholdmag

Ignore stars fainter than this magnitude

stars

A user supplied buffer used to store descriptions of stars within the field of view. Upon input *maxstars* specifies the maximum size of this list. In IDL and Matlab it returns a structure whose fields are derived from the star catalogue entries stored in C++.

maxstars

Upon input specifies the maximum number of entries that can be placed in the used supplied buffer, *stars.* Upon output contains the total number of stars within the field of view above the specified magnitude regardless of the size of the user supplied buffer. Matlab and IDL will never return more than 100 stars.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Comments

The instrument pointing vectors can be retrieved using **GetCFUnitVector** followed by a call to CFToECI.

Return to Odin Level 1 Services

GetStarsInInstrumentFOV

A convenient wrapper function for the API function **GetStarsInFOV.** Returns a list of stars within the specified field of view. The algorithm ensures proper transformation of the light aberration correction and selects the proper axes and type of field of view for the specified instrument.

HRESULT GetStarsInInstrumentFOV(

double	*mjd*,
int	*odin_instrument_id*,
double	*pointingaccuracy*,
double	*thresholdmag*,
STAR_CATALOGUE_ENTRY*	*stars*,
int*	*maxstars*

);

```
IDL> stars = L1->GETSTARSININSTRUMENTFOV( mjd,
                                          odin_instrument_id,
                                          pointingaccuracy,
                                          thresholdmag)
```

```
MAT> [stars] = GetStarsInInstrumentFOV( L1,
                                        mjd,
                                        odin_instrument_id,
                                        pointingaccuracy,
                                        thresholdmag)
```

Parameters

mjd
The UTC expressed as a Modified Julian Date.

odin_instrument_id
The id of the requested instrument. This must be selected from enum ODIN_INSTRUMENT.

pointingaccuracy
The overall pointing accuracy of the spacecraft expressed in degrees. This function serves to widen the effective field of view of the instrument. Hence the function returns the list of stars that may be in the field of view rather than an absolute list of stars that are definitely in the field of view.

thresholdmag
Ignore stars fainter than this magnitude

stars
A user supplied buffer used to store descriptions of stars within the field of view. Upon input *maxstars* specifies the maximum size of this list. In IDL and Matlab it returns a structure whose fields are derived from the star catalogue entries stored in C++.

maxstars
Upon input specifies the maximum number of entries that can be placed in the user-supplied buffer, *stars.* Upon output contains the total number of stars

within the field of view. This value may be greater than the size of the user supplied buffer but the algorithm will not write beyond the end of the buffer. Matlab and IDL will never return more than 100 stars.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

GetUVPSF

Get the spectrograph FWHM point spread function using the PSF measured from the 3 UV Fraunhofer lines (313nm, 320 nm and 351 nm).

HRESULT GetUVPSF (
double *mjd*,
double *wavelen_nm,*
double* *fwhm_psf_nm*
);

```
IDL> N/A

MAT> [fwhm_psf_nm, ok] = GetUVPSF(L1, mjd, wavelen_nm)
```

Parameters

mjd
The modified Julian date at which the value of gamma is required

Wavelen_nm
The wavelength in nanometers at which the point spread function is required. Note that the current system truncates the wavelength interpolation at 313 nm and 351 nm

Fwhm_psf_nm
Returns the full width half maximum point spread function expressed in nanometers. May return negative answer if the function does not succeed.

Returns

Return S_OK if success else E_FAIL or other COM error codes

Return to Odin Level 1 Services

InitializeLevel1Services

This function must be called before using the level 1 services. Each call to **InitializeLevel1Services** must be matched by a call to **UninitializeLevel1Services**.

HRESULT InitializeLevel1Services(
InxLog* *logger*
);

```
IDL> Not required, automatically performed at object creation.

MAT> Not required, automatically performed at object creation.
```

Parameters

logger

The InxLog* interface of the object that will be used for reporting error messages within the Level 1 services and lower level Onyx database software. This parameter may be NULL in which case all error messages are disabled.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

LoadMjdOS

Loads all the requested range of OSIRIS spectrograph records, tangent point and solar angles in to local variables for a specified time range. Only available in Matlab and IDL.

```
IDL> os = L1->LoadMjdOS( startmjd, endmjd, data, tp, sun, error, keywords)

1. MAT> [os, data, tp, sun, error] = LoadOrbitOS( L1, startmjd, endmjd);
2. MAT> [os, data, tp, sun, error] = LoadOrbitOS( L1, startmjd, endmjd, options);
```

Parameters

startmjd

The time at which to start retrieving records expressed as a modified Julian Date.

endmjd

The time at which to finish retrieving records expressed as a modified Julian Date.

See function LoadOrbitOS for a description of all other variables and keyword options.

LoadOrbitOS

Loads all the requested range of OSIRIS spectrograph records, tangent point and solar angles in to local variables for a specified orbit. Only available in Matlab and IDL.

```
IDL> os = L1->LoadOrbitOS( orbit, data, tp, sun, error, keywordoptions)

1. MAT> [os, data, tp, sun, error] = LoadOrbitOS( L1, orbit );
2. MAT> [os, data, tp, sun, error] = LoadOrbitOS( L1, orbit, options);
```

Parameters

orbit
The orbit number to load.

data
Returns a double array (1353, N) corresponding to the spectra for each of the N records of variable *os*..

tp
Returns a double array (3, N) corresponding to the tangent point for each of the N records of *os*. Index 1 = latitude, Index 2 = longitude, Index 3 = Height in kms.

sun
Returns a double array (3, N) corresponding to the solar angles for each of the N records of *os*. Index 1 = solar zenith angle at tangent point, Index 2 = solar azimuth angle at tangent point, Index 3 = solar scattering angle.

error
If requested through option *geterror*, returns a double array (1353, N) corresponding to the error of each value of variable *data*. By default it is not requested and returns a scalar 0.

options
A string that specifies keyword options eg. 'szarange = [minsza, maxsza];'

Keyword options

szarange = [minsza, maxsza];
Sets the acceptable solar zenith range, default is ‘szarange = [0,180];’

heightrange = [minh_kms, maxh_kms];
Sets the acceptable height range in kms, default is ‘heightrange=[-100,200];’

level = ‘’0_’’;
Loads level 0 data instead of level 1. Default is level 1

geterror = true;

Loads the error bars and return the value in variable *error*. The default is to not load errors and set variable *error* to a scalar value of 0.

checkflags = false;

Disable checking the pixel flags and masking of bad pixels. The default is to check the pixel flags and set any bad pixels in variable *data* to 0.

Return to Odin Level 1 Services

LoadScanOS

Loads all the requested range of OSIRIS spectrograph records, tangent point and solar angles in to local variables for a specified scan number. Only available in Matlab and IDL.

```
IDL> os = L1->LoadMjdOS( scan, data, tp, sun, error, keywords)

1. MAT> [os, data, tp, sun, error] = LoadOrbitOS( L1, scan);
2. MAT> [os, data, tp, sun, error] = LoadOrbitOS( L1, scan, options);
```

Parameters

scan

Either the requested scan number (1000*orbit plus scan in orbit) or a SCAN_INFO structure returned by GetScanInfo.

See function LoadOrbitOS for a description of all other variables and keyword options.

LocateOrbitFile

Function searches all of the directories specified by environment variable ODINORBITDIR for the specified orbital data file. Saves the user from having to develop their own code. Function returns the full path name of the file if found.

HRESULT LocateOrbitFile(
char *sitecode*,
char *instrumentid*,
const char* *level*,
int *orbitnumber*,
const char* *extension,*
char* *buffer,*
int *maxchar*
);

```
IDL> buffer = L1->LOCATEORBITFILE(  sitecode,
                                    instrumentid,
                                    level,
                                    orbitnumber,
                                    extension)

MAT> [buffer] = LocateOrbitFile( L1,sitecode,
                                    instrumentid,
                                    level,
                                    orbitnumber,
                                    extension)
```

Parameters

sitecode
The character identifying the sitecode. This is the first character of the orbital filename. All OSIRIS orbital data files start with 's'.

instrumentid
The character identifying the type of data. This is the second character of the orbital filename. Spectrograph data is 's', IR data is 'i', attitude data is 'o' and housekeeping data is 'h'.

level
A two character string identifying the product level. For OSIRIS, '0_' is level 0 and '1_' is level 1.

orbitnumber
The requested orbit number.

extension
The file extension of the requested file. Spectrograph Level 1 files are '.os1'. IR Level 1 files are '.ir1', Attitude level 1 files are '.oat' and housekeeping level 1 files are '.hk1'. . Spectrograph Level 0 files are '.os0'. IR Level 1 files are '.ir0', Attitude level 0 files are '.oat' and housekeeping level 0 files are '.hk'

buffer

A user supplied buffer that will return the full pathname of the requested file if found otherwise will contain an empty, zero terminated string. The number of characters in the buffer is specified by **maxchar**. The function will fail and return an empty string if the buffer is not large enough to hold the files full path.

maxchar

The size of the user supplied buffer in characters. The function will guarantee that it does not write more than maxchar characters to the buffer

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

LOSEntrancePoints

Calculates the points at which a given line of sight from a given position intersects, (enters and exits), a shell at a specific height above the geoid Earth. The algorithm may use straight line geometry for a rapid solution or ray-tracing through a standard model for more accurate, but slower, refractive index corrections. Note that the line of sight will generally not intersect height shells the tangent point. Raytracing option is not yet implemented.

HRESULT LOSEntrancePoints(
const ECIVECTOR* *satelliteposition*,
const ECIVECTOR* *lookvector*,
double *geodetic_height*,
ECIVECTOR* *entrance_point*,
ECIVECTOR* *exit_point,*
nxBOOL *doraytracing*
);

```
IDL> L1->LOSENTRANCEPOINTS,   satelliteposition,
                              lookvector,
                              geodetic_height,
                              entrance_point,
                              exit_point,
                              doraytrace

MAT> [entrance_point,
      Exit_point] = LOSEntrancePoints( L1,
                                       satelliteposition,
                                       lookvector,
                                       geodetic_height,
                                       doraytrace)
```

Parameters

satelliteposition
The position of the satellite/observer in the ECI reference frame. This array is not modified by this procedure.

lookvector
The look direction of the observer in the ECI reference frame. This array is not modified by this procedure. See CFToECI.

*geodetic*_height
The height of the shell in km above the surface of the reference geoid.

entrance_point
Returns the ECI point where the ray emerging from the *satelliteposition* in direction *lookdirection* enters the shell at height *geodetic_height* above the reference geoid. If there is no intersection because the shell is either above the satellite or below the tangent point then *entrance_point* is set to 0.

exit_point

Returns the ECI point where the ray emerging from the *satelliteposition* in direction *lookdirection* exits the shell at height *geodetic_height* above the reference geoid. If there is no intersection because the shell is below the tangent point then *exit_point* is set to 0.

doraytracing

If nxTRUE then determine the entrance and exit points using a ray-tracing algorithm otherwise use straight line geometry.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

LOSTangentPoint

Calculates the ECI coordinates of the tangent point of a given look direction at a given position. The calculation assumes the standard geoid Earth used in Odin. It does not require a correction for sidereal time as there is circular symmetry in longitude.

HRESULT LOSTangentPoint(
const ECIVECTOR* *satelliteposition*,
const ECIVECTOR* *lookvector*,
ECIVECTOR* *tangent_point,*
nxBOOL *doraytracing*
);

```
IDL> tangent_point =LOSTANGENTPOINT(      satelliteposition,
                                          lookvector,
                                          doraytrace)

MAT> [tangent_point] = LOSTangentPoint(L1,satelliteposition,
                                          lookvector,
                                          doraytrace)
```

Parameters

satelliteposition
The position of the satellite/observer in the ECI reference frame. This array is not modified by this procedure.

lookvector
The look direction of the observer in the ECI reference frame. This array is not modified by this procedure.

tangent_point
Returns the tangent point in the ECI reference frame. If no tangent point exists then it returns 0 in all fields.

doraytracing
If nxTRUE then determine the tangent point using a ray tracing algorithm otherwise use straight line geometry. Not yet supported.

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

OrbitalPlanetoECI

Converts the ECI coordinates of a point to orbital plane coordinates. The orbital plane coordinates are returned as a GEODETIC_COORD but the caller must be aware that they are actually geocentric coordinates and not geodetic coordinates.

HRESULT OrbitalPlanetoECI (
double *anmjd*,
nxBOOL *updateplane*,
const GEODETIC_COORD**geodetic_pos*
ECIVECTOR* *eci_position*,
);

```
IDL> eci_position = L1->OrbitalPlanetoECI( anmjd, geodetic_pos,
                                           updateplane=updateplane)

MAT> [eci_position] = OrbitalPlanetoECI( L1,
                                         anmjd,
                                         updateplane,
                                         geodetic_pos )
```

Parameters

anmjd
The UTC expressed as a Modified Julian Date used to calculate the orbital plane. This value is only used if *updatereferenceplane* is *true* or if the plane is internally undefined. The value is also used to determine the number of revolutions that have occurred since the epoch that defines the orbital plane.

updateplane
If nxTRUE then update the reference orbital plane using the ascending node prior to time *anmjd.* If nxFALSE then don't update the orbital plane. This is a keyword in the IDL implementation.

geodetic_pos
The location of a point expressed in orbital plane coordinates: geocentric latitude, longitude and geocentric radius in kilometers. In orbital plane coordinates, X points to the ascending node in the equatorial plane. Z is the spin axis of the orbit perpendicular to the orbital plane (towards sun for ODIN) and Y is perpendicular to X and Z. Longitude is the angular distance from the ascending node in degrees. Latitude is the angular distance from the orbital plane and height is the geocentric radius in kilometers.

eci_position
The location of the same point expressed in ECI coordinates (meters).

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

OSIRISAvgTemperature

Returns one of several temperatures from the OSIRIS HouseKeeping. The code looks up the temperature from the onboard statistics stored and transmitted by OSIRIS, which usually averages several hundred measurements to get a smoother value.

HRESULT OSIRISAvgTemperature (
double *mjd*,
double* *temperature,*
int *avgtype*
);

```
IDL> N/A

MAT> [temperature] = OSIRISAvgTemperature(L1, mjd, avgtype)
```

Parameters

mjd
The modified Julian date at which the temperature is required.

temperature
returns the temperature value. May return very large or negative value if there is an error, eg -9999.0.

avgtype
Identifies the type of temperature required.
0 retrieves optics box temperature.
1 retrieves OS strap temperature.
2 retrieves OS CCD temperature.

Returns

Return S_OK if success else E_FAIL or other COM error codes

Return to Odin Level 1 Services

OSSolarAngles

Calculates the solar angles associated with the OSIRIS-Spectrograph bore sight look vector for the array of instants specified in mjdarray. Returns the solar angles as an array of (3,N) numbers. This function is only available in Matlab and IDL.

```
IDL>  geodata  = L1->OSSolarAngles( mjdarray)

MAT> [geodata] = OSSolarAngles( L1, mjdarray );
```

Parameters

mjdarray

An array of UTC instants expressed as a Modified Julian Date.

Returns

The solar angles an array (3,N). First index is returned as 1= solar zenith angle at tangent point, 2 = solar azimuth angle at tangent point, 3 = solar scattering angle.

Return to Odin Level 1 Services

OSTangentPoint

Calculates the tangent point of the OSIRIS-Spectrograph, bore-sight look vectors for the array of times specified in mjdarray. Returns the tangent point as an array of geodetic coords (3,N) . This function is only available in Matlab and IDL.

```
IDL>  geodata  = L1->OSTangentPoint( mjdarray)

MAT> [geodata] = OSTangentPoint( L1, mjdarray );
```

Parameters

mjdarray

An array of UTC instants expressed as a Modified Julian Date.

Returns

The location of the tangent point as an array (3,N). First index is returned as 1= latitude, 2 = longitude, 3 = height in km.

Return to Odin Level 1 Services

ReleaseScanData

Release the scan of data acquired by either **GetScanData** or **GetDataBetweenTimes.** The status of the collection object is undefined after this call. The user should assume that the object no longer exists and must not make any calls to its interface functions.

HRESULT ReleaseScanData(
IOnyxCollection* *collection*
);

```
IDL> Not used

MAT> Not Used
```

Parameters

collection
Release the collection returned from either **GetScanData** or **GetDataBetweenTimes.**

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

StwLocateResetEpoch

Informs the code that it should convert the times in all subsequent calls to StwToUtc using a reset epoch that is derived from the nominal UTC passed to the routine. This algorithm will be used by users who need STW conversion for a time period that they know is free of Odin resets. The reset epoch will stay in force until the user either calls **StwUsesFixedResetEpoch** or **StwLocateResetEpoch**.

HRESULT StwLocateResetEpoch(
double *mjd*
);

```
IDL> L1->STWLOCATERESETEPOCH, mjd
```

```
MAT> [status] = StwLocateResetEpoch(L1, mjd)
```

Parameters

mjd

The nominal UTC of the desired time conversion expressed as a modified Julian Date. The user should be careful when using this function near to periods when the Odin STW is known to reset.

Returns

Return S_OK otherwise it will return E_FAIL and other COM error codes. This is the value of *status* in the Matlab version.

Return to Odin Level 1 Services

StwToUtc

Converts the Odin Satellite Time Word (STW also known as the format counter) to Coordinated Universal Time. The algorithm implements the piecewise linear segments distributed by SSC and stored as part of the level 0 data product in file STWTOUTC.INI. Proper behaviour of the function requires that the user calls either **StwUsesFixedResetEpoch** or **StwLocateResetEpoch** at least once prior to calling **StwToUtc**. This is so the code knows how to locate the appropriate epoch for the given STW. The STW epochs arise from the fact that the Odin platform does not guarantee that the STW may not reset during the mission. It is anticipated that only level 0 users will need to call this function.

HRESULT StwToUtc(
nxDWORD *stw*
double* *mjd*
);

```
IDL> mjd = L1->STWTOUTC( stw)

MAT> [mjd] = StwToUtc(L1, stw)
```

Parameters

stw
The satellite time word.

mjd
Returns the best estimate of the UTC expressed as a modified Julian Date. Note that there are no guarantees as to the accuracy of the conversion if the routine does not return S_OK.

Returns

Return S_OK if success and the accuracy of the conversion is reasonable. Returns S_FALSE if the conversion is of dubious quality. This will normally occur when the algorithm is extrapolating out of range or there are no points in the requested reset epoch. Future version may return E_FAIL and other COM error codes.

Note that stw and mjd field in the spectrograh and IR imaging structure may not agree with this conversion as there other internal timing offsets applied to these fields.

Return to Odin Level 1 Services

StwUsesFixedResetEpoch

Informs the code that it should convert the times in all subsequent calls to StwToUtc using a specific STW reset epoch. The Reset epoch is defined in the <RESET EPOCH> section of the level 0 data poduct file STWTOUTC.INI. It is anticipated that only level 0 users will need to call this function. It is intended that the OSIRIS Level 0 SSC decode algorithms will read the STW reset epoch index directly from the SSC level 0 filenames.

HRESULT StwUsesFixedResetEpoch(
int *resetepochindex*
);

```
IDL> L1->STWUSESFIXEDRESETEPOCH, resetepochindex
```

```
MAT> [status] = StwUsesFixedResetEpoch(L1, resetepochindex)
```

Parameters

resetepochindex
Specifies the reset epoch index which the user knows a-priori to exist in file STWTOUTC.INI**.**

Returns

Return S_OK otherwise it will return E_FAIL and other COM error codes. This is the value return in *status* in the Matlab version.

Return to Odin Level 1 Services

UninitializeLevel1Services

This function must be called after the user has finished using the level 1 services. Each call to **InitializeLevel1Services** must be matched by a call to **UninitializeLevel1Services**. Failure to observe this constraint will result in memory leaks at program termination. The user must not access OSIRIS Level 1 services after calling this function.

Note that IDL and Matlab users may simply exit their session without calling this function. This does no real harm as the Level 1 database is read-only.

HRESULT UninitializeLevel1Services();

```
IDL> Not used. Automatically called when destroying objects
```

```
MAT> [status] = UninitializeLevel1Services(L1).
```

Returns

Return S_OK if success else E_FAIL or other COM error codes.

Return to Odin Level 1 Services

OSIRIS LEVEL 1 API Structures

Spectrograph and IR Data Product History: Audit Field Values

The spectrograph and IR, Level 0 and Level 1, data products contain an *audit* field in the record headers. The audit field tracks changes in the Level 0 and Level 1 processing software. The field *audit[0]* is used to track the version of Level 0 software used in the processing and *audit[1]* is used to track the version of Level 1. Note that these dates are different to the date used by the Level 1 services.

The audit fields were originally generated from the time of compilation of specific modules in the Level 0 and Level 1 software. This technique ultimately proved difficult to track and maintain and was replaced in November 2005 by an explicit dating system. Fortunately there were remarkably few changes in the Level 0 and Level 1 processing system until November 2005.

Level 0 Audit[0]

1. Less than 53675.0. (pre 2005-11-01).
   Original software decoding. No detailed tracking available.

2. 53675.0 (2005-11-01)
   Fixed STW to UTC conversion and added internal timing offsets for spectrograph and IR. This version is only applicable for merged, level 0, orbital files. All STW to UTC conversions are derived solely from STW_PLN.DAT. In addition an extra 129.5 milliseconds is added to the *mjd* field of the IR records and an extra 97.8 milliseconds is added to the *mjd* field of the spectrograph records. These times account for apparent timing offsets between OSIRIS and the Odin attitude control system.

3. 53689.0 (2005-11-15)
   Fixed bug in spectrograph level 0, OS_L0 structure: The crosssectionindex was off by an offset of 8, i.e. 8 had to be subtracted from the old value to make it properly reference the pixels. This was due to the fact that the OSIRIS instrument has an extra 8 "readout" pixels before it reads out the spectrum. This bug has been addressed in Level 0 versions after this time.

Level 1, Audit[1]

1. Less than 53675.0 (pre 2005-11-01)
   Original software decoding.

2. 53675.0 (2005-11-01)
   Fixed *mjd* field to reflect the changes at Level 0.

3. 53745.0 (2006-01-10)
   Applied Vega Absolute Calibration to spectrograph data.

Base Data Types

The Odin/OSIRIS software uses the following base data types

nxBYTE	unsigned 8 bit integer
nxWORD	unsigned 16 bit integer
nxDWORD	unsigned 32 bit integer
nxBOOL	*bool* on "C" compilers that support this base type else *int*
nxTRUE	*true* on "C" compilers that support *bool* else 1.
nxFALSE	*false* on "C" compilers that support *bool* else 0.
nxCHAR	signed 8 bit integer (rarely used)
nxSHORT	signed 16 bit integer (rarely used)
nxLONG	signed 32 bit integer (rarely used)
float	A floating point which is at least 4 bytes but can be greater.
double	A floating point which is at least 8 bytes but can be greater.
int	An integer that is at least 4 bytes but can be greater

Return to Structures and Enumerations

CFVECTOR, Spacecraft Control Frame

A reference frame based upon the physical layout of the spacecraft. It is assumed that the orientation of all instrumentation is known in the spacecraft control frame. The SSC attitude solution provides quaternions every 17/16 seconds to rotate from the spacecraft control frame to ECI coordinates, SSC Document SSAK31-7, *Odin Operations- Attitude and Orbit Related Definitions*.

The **CFVECTOR** is identical in structure to the **ECIVECTOR** and is distinguished for clarity in the software interfaces.

typedef ECIVECTOR CFVECTOR;

IDL: ECIVECTOR is impleneted as dblarr(3).

Return to Structures and Enumerations

ECIVECTOR, Earth Centered Inertial Reference Frame

Primary reference frame for attitude calculations. Following the definition used by SSC, Document SSAK31-7, *Odin Operations- Attitude and Orbit Related Definitions*, the ECI frame is based upon the celestial sphere using a true equator and equinox for the current date. The ECI frame is centred at the satellites centre of mass and moves with the spacecraft velocity (private communication with SSC.

The biggest correction to the attitude is for light aberration due to the relative motion o fthe spacecraft. This term is of the the order 0.1 to 0.2 arc minutes.

Most ECIVECTOR objects are either dimensionless unit-vectors, position vectors expressed in meters or velocity expressed in meters per second. We try to stick with S.I units

typedef double[3] ECIVECTOR;

Refers to a three dimensional vector expressed in the Earth Centred Inertial (ECI). The components are stored in the array as [X,Y,Z]; i.e. element 0 = X, element 1 = Y, element 2 = Z.

IDL: implemented as dblarr(3)

Return to Structures and Enumerations

GEOVECTOR, Geographic Earth Centered Reference Frame

A geocentric, X, Y, Z coordinate system that rotates with the Earth. The X axis is in the plane of the equator and the Greenwich meridian. The Z axis is parallel to the pin axis of the earth and Y makes a right handed orthogonal system.

Most GEOVECTOR objects are either dimensionless unit-vectors or position vectors expressed in meters.

typedef ECIVECTOR GEOVECTOR;

Refers to a three dimensional vector expressed in the Geographic Earth Centred frame. The components are stored in the array as [X,Y,Z]; i.e. element 0 = X, element 1 = Y, element 2 = Z.

IDL: implemented as dblarr(3)

Return to Structures and Enumerations

enum ECI_REFERENCE_FRAME

Identifies the primary velocity reference frames used by the analysis. The purpose is to provide proper corrections for the aberration of light to different reference frames.

```
enum ECI_REFERENCE_FRAME{
ECI_SPACECRAFT,
ECI_TOPOCENTRIC,
ECI_GEOCENTRIC
};
```

IDL:

```
ECI_SPACECRAFT()
ECI_TOPOCENTRIC()
ECI_GEOCENTRIC()
```

ECI_SPACECRAFT

Defines the velocity reference frame which is moving with the satellite. This is the velocity reference frame in which SSC deliver attitude solutions. Most users will always use this value and accept the error of a few arc seconds.

ECI_TOPOCENTRIC

The velocity of a point at the location of the satellite which is rotating with the Earth. From an observer on the surface of the Earth this point would appear stationary.

ECI_GEOCENTRIC

The velocity of the center of mass of the Earth. This is the velocity reference frame used by the Novas software when calculating apparent positions of stars and planets.

Return to Structures and Enumerations

enum ODIN_INSTRUMENT

This enumeration indentifies all of the possible instrumentation on board the ODIN spacecraft.

enum ODIN_INSTRUMENT{
INST_PLATFORM = 1,
INST_ACS = 2,
INST_SMR = 3,
INST_OS = 4,
INST_IR1 = 5,
INST_IR2 = 6,
INST_IR3 = 7,
};

IDL and Matlab:

```
INST_PLATFORM()
INST_ACS()
INST_SMR()
INST_OS()
INST_IR1()
INST_IR2()
INST_IR3()
```

Note:
INST_ACDC() has been replaced with INST_ACS(), Version 1.17, 2008-10-02.

Return to Structures and Enumerations

enum ODIN_POINTING_FRAME

This enumeration identifies all of the possible instrumentation unit vectors on board the Odin spacecraft.

enum ODIN_POINTING_FRAME {

ODIN_X_CF	Odin Control Frame X axis.
ODIN_Y_CF	Odin Control Frame Y axis.
ODIN_Z_CF	Odin Control Frame Z axis.
SMR_X_BORE	The sub millimeter bore-sight.
SMR_Y	The sub millimeter Y axis close to the Odin Control Frame Y axis.
SMR_Z	The sub millimeter Z axis close to the Odin Control Frame Z axis.
OS_X_BORE	The OSIRIS Optical spectrograph bore-sight, close to the Odin Control Frame X axis.
OS_Y_HEIGHT	The OSIRIS Optical spectrograph height height direction, close to Odin Control Frame Y direction.
OS_Z_SLIT	The OSIRIS Optical spectrograph slit/spatial direction, close to the Odin Control Frame Z axis.
IR1_X_BORE	The OSIRIS IR1 bore-sight, close to the Odin Control Frame X axis.
IR1_Y_ARRAY	The OSIRIS IR1 linear array direction, close to the Odin Control Frame Y axis.
IR1_Z	The OSIRIS IR1 spatial direction, close to the Odin control Frame Z axis.
IR2_X_BORE	The OSIRIS IR2 bore-sight, close to the Odin Control Frame X axis.
IR2_Y_ARRAY	The OSIRIS IR2 linear array direction, close to the Odin Control Frame Y axis.
IR2_Z	The OSIRIS IR2 spatial direction, close to the Odin control Frame Z axis.
IR3_X_BORE	The OSIRIS IR3 bore-sight, close to the Odin Control Frame X axis.
IR3_Y_ARRAY	The OSIRIS IR3 linear array direction, close to the Odin Control Frame Y axis.
IR3_Z	The OSIRIS IR3 spatial direction, close to the Odin control Frame Z axis.
ST2_X_BORE	The bore sight of Star Tracker 2.
ST2_Y_AXIS	The Y axis of Star tracker 2. This direction is about 7.1 degrees wide and is nominally aligned with the tangent height direction.
ST2_Z_AXIS	The Z axis of Star tracker 2. This direction is about 9.55 degrees wide and is nominally parallel to the Earth's horizon.

};

IDL and Matlab:

```
ODIN_X_CF()
ODIN_Y_CF()
ODIN_Z_CF()
SMR_X_BORE()
SMR_Y()
SMR_Z()
OS_X_BORE()
```

```
OS_Y_HEIGHT()
OS_Z_SLIT()
IR1_X_BORE()
IR1_Y_ARRAY()
IR1_Z()
IR2_X_BORE()
IR2_Y_ARRAY()
IR2_Z()
IR3_X_BORE()
IR3_Y_ARRAY()
IR3_Z()
ST2_X_BORE()
ST2_Y_AXIS()
ST2_Z_AXIS()
```

Return to Structures and Enumerations

enum PLANETARY_BODY

Used to identify different heavenly bodies located within the solar system. The numbers used here match the body identification numbers used by the NOVAS Ephemeris package (see NOVAS function **solarsystem**).

enum PLANETARY_BODY{
PB_MERCURY = 1,
PB_VENUS = 2,
PB_EARTH = 3,
PB_MARS = 4,
PB_JUPITER = 5
PB_SATURN = 6,
PB_URANUS = 7,
PB_NEPTUNE = 8,
PB_PLUTO = 9
PB_SUN = 10,
PB_MOON = 11
};

IDL and Matlab:

```
PB_MERCURY()
PB_VENUS()
PB_EARTH()
PB_MARS()
PB_JUPITER()
PB_SATURN()
PB_URANUS()
PB_NEPTUNE()
PB_PLUTO()
PB_SUN()
PB_MOON()
```

Return to Structures and Enumerations

GEODETIC_COORD

Used to define a location using a geodetic (ellipsoidal) earth. The geoid used by OSIRIS matches the geoid used by SSC, SSC Document SSAK31-7, *Odin Operations-Attitude and Orbit Related Definitions*. Note that the **GEODETIC_COORD** expresses height in kilometers while **ECIVECTOR** uses meters.

```
struct GEODETIC_COORD{
double      latitude;
double      longitude;
double      height;
};
```

Entries

latitude
The geodetic latitude in degrees, range -90 to +90

longitude
The geodetic longitude in degrees, range 0 to 360.0

height
The height in kilometres above the reference geoid (a.k.a. surface of the Earth).

The reference geoid is,

Equatorial radius of Earth, a = 6378140.0 metres
Reciprocal flattening 1/f = 298.257

This model is from the IAU 1976 geodetic reference spheroid.

IDL:
Implemented as a dblarr(3) = [latitude, longitide, height]

Matlab:
Implemented as a (3x1) array = [latitude; longitude; height]

Return to Structures and Enumerations

IR_L1

Structure that represents the IR Level 1 data .

```
struct IR_L1{
double                  mjd;
nxDWORD                 stw;
double                  exposureTime;
float                   temperature;
float                   tempavg;
nxBYTE                  detectorid;
nxWORD                  mode;
nxWORD                  scienceprog;
nxBOOL                  shutter;
nxBOOL                  lamp1;
nxBOOL                  lamp2;
nxWORD                  targetIndex;
nxDWORD                 exceptions;
nxDWORD                 processingflags;
ONYX_VERSION_STRUCT     audit;
IOnyxArray*             data;
IOnyxArray*             error;
IOnyxArray*             flags;
 };
```

Entries

mjd

The UTC expressed as a Modified Julian Date at the start of the IR exposure. The time is derived from the satellite time word, which has an intrinsic resolution of 1/16th of a second

stw

The Satellite time word associated with the *mjd*. Note that the Satellite Time Word is not guaranteed to be monotonic for the duration of the mission. I.e. it may reset back to zero depending upon platform requirements.

exposureTime

The IR exposure time in seconds. The exposure time is corrected for any constant offsets inherent to the read-out-electronics.

temperature

The temperature of the detector in Celsius. This value is determined from a single read of the A/D converter at the beginning of the exposure.

tempavg

The running average temperature of the detector. The running average is derived from the average of temperature measurements over the previous "??" seconds.

detectorid

The Id number of this detector, INST_IR1, INST_IR2 or INST_IR3. All other values are undefined.

mode
The unique id code of the OSIRIS imaging mode used to collect this data.

scienceprog
The unique id code of the OSIRIS science mode used to collect this data. The value is related to the overall scientific goal of the current set of measurements. However the value may change during any given satellite scan as several OSIRIS *scienceprog* values are associated with one scientific goal. All calibration programs which would normally be excluded from standard level 2 processing are required to set the most significant bit (Bit 15) to 1. All atmospheric science programs which would be processed by level 2 algorithms are required to have bit 15 set to 0.

shutter
Indicates the status of the OSIRIS IR shutter. nxFALSE if open, nxTRUE if closed or moving

lamp1
Indicates the status of the OSIRIS primary calibration lamp. nxFALSE if off, nxTRUE if on.

lamp2
Indicates the status of the OSIRIS secondary calibration lamp. nxFALSE if off, nxTRUE if on.

targetIndex
The real-time ACDC target index at the start of the exposure.

exceptions
32 bits of flags used to determine various exceptions that have occurred in processing. The most significant bit, bit 31, indicates severity. If bit 31 is set then the record has a serious problem and is probably unusable. If bit 31 is clear then the record has exceptions but may be usable depending upon context.

processingflags
32 bits of flags used to indicate which processing steps have been applied to the data. For example data collected with the shutter closed and lamps off do not have the dark current removed while all other data do have the dark current removed. All bit fields are currently t.b.d. All definitions require consultation with level 1 and level 2 IR processing groups.

audit
Records the software/data versioning of this data record. The standard level 1 data processing will guarantee that this value only changes when OSIRIS is powered off.

data
OnyxDataObject object that provides access to the data array of **double[128]** using the ONYX interfaces

error

OnyxDataObject object that provides access to the error array of **double[128]** using the ONYX interfaces.

flags

OnyxDataObject object that provides access to the pixel flags array of **nxBYTE[128]** using the ONYX interfaces. Each nxBYTE provides 8 bits for t.b.d. exception flags specific to this pixel. A value of zero means the pixel is good. Any bit set in the array indicates earlier processing has identified an anomaly with the associated pixel.

Exception Flags

OSIEX_SERIOUS (0x80000000)

Flags a serious error in the data and indicates that the data should be discarded.

OSIEX_IR_BADDMA (0x00000004)

flags that the IR DMA channel failed in the OSIRIS firmware. Indicates data corruption

OSIEX_IR_BADUTC (0x00000008)

flags that the UT field (MJD) may be inaccurate. Normally set when processing raw level 0 before attitude is available

Return to Structures and Enumerations

Modified Julian Date

Modified Julian Date provides a convenient method to store time values. It is used as the primary storage mechanism for all times in the OSIRIS level 1 database. The integer part of the Modified Julian Date represents the day number while the fractional part represents the time of day since midnight.

January 1st 1970 at 00:00:00 UTC is 40587.0 when expressed as a Modified Julian Date.

Modified Julian Date = Julian Date - 2400000.5

In Matlab:
Matlab Serial Date = Modified Julian Date + 678942.0

Return to Structures and Enumerations

OS_L0

Structure used to represent level 0 data from the OSIRIS optical spectrograph.

```
struct OS_L0{
double                  mjd;
nxDWORD                 stw;
double                  exposureTime;
float                   temperature;
nxWORD                  mode;
nxWORD                  scienceprog;
nxBYTE                  roe;
nxBYTE                  shuttermode;
nxWORD                  spm_baserow;
nxWORD                  spm_numrows;
nxBYTE                  spm_processingMode;
nxWORD                  targetIndex;
nxDWORD                 exceptions;
double                  compressionrate;
nxWORD                  numcolumns;
nxWORD                  numrows;
ONYX_VERSION_STRUCT     audit;
double                  neargatedcbias;
double                  fargatedcbias;
double                  dark_average[3];
IOnyxArray*             data;
IOnyxArray*             darkrow;
IOnyxArray*             dcbias;
IOnyxArray*             crosssection;
IOnyxArray*             crosssectionindex;
IOnyxArray*             overflowcounter;
};
```

Entries

mjd

The UTC expressed as Modified Julian Date at time at the start of the OS exposure. The time is derived from the satellite time word which has an intrinsic resolution of 1/16th of a second.

stw

The Satellite time word associated with the *mjd*. Note that the Satellite Time Word is not guaranteed to be monotonic for the duration of the mission. i.e. it may reset back to zero depending upon platform requirements.

exposureTime

The OS exposure time in seconds. The exposure time is derived from the number of ROE ticks multiplied by the ROE exponent. It does not include any corrections inherent to delays in the read-out-electronics.

temperature

The temperature of the Optical Spectrograph CCD in Celsius. This value is determined from a single read of the A/D converter at the beginning of the exposure.

mode
The unique id code of the OSIRIS imaging mode used to collect this data.

scienceprog
The unique id code of the OSIRIS science mode used to collect this data. The value is related to the overall scientific goal of the current set of measurements. However the value may change during any given satellite scan as several OSIRIS *scienceprog* values are associated with one scientific goal.

roe
The configuration code of the OSIRIS OS CCD read-out-electronics. The pre-launch values are defined as follows:
0 = 32x1353. No on-chip binning.
1 = 16x1353. On-chip binning of 2 reduces 32 rows to 16.
2 = 8x1353. On-chip binning of 4 reduces 32 rows to 8.
3 = 286x1353. Full CCD including storage area. Used for engineering.
4 = 143x1353. CCD imaging area. Used for engineering.
7 = 2x1353. Obsolete and unsupported mode.

shuttermode
Indicates the mode of the OSIRIS OS shutter.
0 = Close,Open,Close
1 = Closed
2 = Open

spm_baserow
The base row used by OSIRIS in the Science Processing Module to bin the data off-chip but before transmission to ground.

spm_numrows
The number of rows, spatially binned in the OSIRIS Science Processing Module.

spm_processingMode
The OSIRIS Science Processing Mode. The valid values are; 0 not binned (1353xn), 1 = spatial binning (1353x1), 2 = McDade & Stegman binning (849x1), 3 = Llewellyn & Evans binning (11x32).

targetIndex
The real-time ACDC target index at the start of the exposure.

exceptions
32 bits of flags used to determine various exceptions that have occurred in processing. The most significant bit, bit 31, indicates severity. If bit 31 is set then the record has a serious problem and is probably unusable. If bit 31 is clear then the record has exceptions but may be usable depending upon context. All bit fields (except bit 31) are currently t.b.d. All definitions require consultation with level1 and level 2 OS processing groups.

compressionrate

The compression rate of the onboad Rice algorithm. Expressed as a number between 0 and 1. O is 100% compression while 1 represents no compression.

numcolumns
The number of columns (wavelength pixels) in the associated data array. Currently this value will be 1353, 849 or 11 corresponding to the value given in *spm_processingMode*.

numrows
The number of rows (spatial pixels) in the associated data array. This value is typically 1 for normal science operations but can assume any value between 1 and 286.

audit
Records the software/data versioning of this data record. The Level 0 processing reserves *audit[0]*. The value stored is the modified Julian date of the applicable level 0 software. More details can be found in the audit description.

neargatedcbias
The average value of the 4 columns (2,3,4,5) from the near read-out gate. The value is in A/D units.

fargatedcbias
The average value o fthe 4 columns (2,3,4,5) from the far read-out gate. The value is in A/D units.

dark_average[3]
The average value in each dark-row region. Expressed in A/D units.

data
OnyxDataObject object that provides access to the data array of **double[***numrows***][***numcolumns***]** using the ONYX interfaces

darkrow
OnyxDataObject object that provides access to the darkrow data. The data are available only when dumping whole images. It is not normally available during normal scientific programs. The darkrow is an array of **double[***1353***]** when available.

dcbias
OnyxDataObject object that provides access to the entire near and far read-out gate values. The data are only available when dumping whole images. It is not normally available during normal scientific programs. The data are stored as an array of **double[***numrows***][***16***]** using the ONYX interfaces. The first 8 elements of each row at the near read out gate. The last 8 elements of each row from the far read out gate.

crosssection
OnyxDataObject object that provides access to 4 slit cross-sections from the detector. It is only available when the roe mode is 0,1 or 2, and spatial binning or Stegman binning is enabled. The number of columns depends upon the roe mode: roe 0 gives 32 columns, roe 1 gives 16 columns and roe 2 gives 8 columns. Spatial binning provides 8 cross-sections. Stegman binning provides 4

cross-sections. The data are stored as an array of **double[***8 or 4***][***ncolumns***]** using the ONYX interfaces.

crosssectionindex

OnyxDataObject. The index of each column on the CCD in the **crosssection** array. It is only valid for spatial binning or Stegman binning. A valid index will be between 0 and 1352. Spatial binning provides 8 cross-sections. Stegman binning provides 4 cross-sections. The data are stored as an array of **double[***8 or 4***][***ncolumns***]** using the ONYX interfaces

overflowcounter

OnyxDataObject. An array of bytes of the same dimensions as the **data** field. It indicates the number of saturated (overflowed) pixels contributing to spatially and wavelength binned data elements. A value of zero implies there were no saturated pixels contributing to the corresponding data element. The array may be a NULL array which implies there were no overflows in this image. The array is only available for images which were spatially and/or wavelength binned off-chip onboard OSIRIS. This field was introduced in May 2001 and may not be available in older HDF files. It defaults to NULL in those cases.

OS_L1

Structure used to represent level 1 data from the OSIRIS optical spectrograph.

```
struct OS_L1{
double                  mjd;
nxDWORD                 stw;
double                  exposureTime;
double                  temperature;
double                  tempavg;
double                  opticstemp;
double                  straptemp;
nxWORD                  mode;
nxWORD                  scienceprog;
nxBYTE                  roe;
nxBYTE                  shuttermode;
nxWORD                  spm_baserow;
nxWORD                  spm_numrows;
nxBYTE                  spm_processingMode;
nxWORD                  targetIndex;
nxDWORD                 exceptions;
nxDWORD                 processingflags;
nxWORD                  numcolumns;
nxWORD                  numrows;
ONYX_VERSION_STRUCT     audit;
IOnyxArray*             data;
IOnyxArray*             error;
IOnyxArray*             flags;
IOnyxArray*             crosssection;
IOnyxArray*             crosssectionindex;
};
```

Entries

mjd

The UTC expressed as Modified Julian Date at time at the start of the OS exposure. The time is derived from the satellite time word which has an intrinsic resolution of 1/16th of a second.

stw

The Satellite time word associated with the *mjd*. Note that the Satellite Time Word is not guaranteed to be monotonic for the duration of the mission. i.e. it may reset back to zero depending upon platform requirements.

exposureTime

The OS exposure time in seconds. The exposure time is corrected for any constant offsets inherent to the read-out-electronics.

temperature

The temperature of the Optical Spectrograph CCD in Celsius. This value is determined from a single read of the A/D converter at the beginning of the exposure.

tempavg
The running average temperature of the OS CCD in Celsius. The running average is derived from the average of temperature measurements over the previous "??" seconds. The value is derived from the analysis of OSIRIS housekeeping data.

opticstemp
The temperature of the OSIRIS optics unit in Celsius. The value is derived from the analysis of OSIRIS housekeeping data.

straptemp
The strap temperature of the OSIRIS optics unit in Celsius. The value is derived from the analysis of OSIRIS housekeeping data. This value is linearly combined with the opticstemp to get an effective temperature used for dark current estimates.

mode
The unique id code of the OSIRIS imaging mode used to collect this data.

scienceprog
The unique id code of the OSIRIS science mode used to collect this data. The value is related to the overall scientific goal of the current set of measurements. However the value may change during any given satellite scan as several OSIRIS *scienceprog* values are associated with one scientific goal. All calibration programs which would normally be excluded from standard level 2 processing are required to set the most significant bit (Bit 15) to 1. All atmospheric science programs which would be processed by level 2 algorithms are required to have bit 15 set to 0.

roe
The configuration code of the OSIRIS OS CCD read-out-electronics. The pre-launch values are defined as follows:
0 = 32x1353. No on-chip binning.
1 = 16x1353. On-chip binning of 2 reduces 32 rows to 16.
2 = 8x1353. On-chip binning of 4 reduces 32 rows to 8.
3 = 286x1353. Not available in level 1. Used for engineering.
4 = 143x1353. Not available in Level 1. Used for engineering.
7 = 2x1353. Obsolete and unsupported mode.

shuttermode
Indicates the mode of the OSIRIS OS shutter.
0 = Close,Open,Close
1 = Closed
2 = Open

spm_baserow
The base row used by OSIRIS in the Science Processing Module to bin the data off-chip but before transmission to ground.

spm_numrows
The number of rows, spatially binned in the OSIRIS Science Processing Module.

spm_processingMode

The OSIRIS Science Processing Mode. The valid values are; 0 not binned (1353 x n), 1 = spatial binning (1353 x 1), 2 = McDade & Stegman binning (849x1), 3 = Llewellyn & Evans binning (11x32).

targetIndex
The real-time ACDC target index at the start of the exposure.

exceptions
32 bits of flags used to determine various exceptions that have occurred in processing. The most significant bit, bit 31, indicates severity. If bit 31 is set then the record has a serious problem and is probably unusable. If bit 31 is clear then the record has exceptions but may be usable depending upon context. All bit fields (except bit 31) are currently t.b.d. All definitions require consultation with level1 and level 2 OS processing groups.

processingflags
32 bits of flags used to indicate which processing steps have been applied to the data. For example data collected with the shutter closed do not have the dark current removed while all other data do have the dark current removed. All bit fields are currently t.b.d. All definitions require consultation with level 1 and level 2 OS processing groups.

numcolumns
The number of columns (wavelength pixels) in the associated data array. Currently this value will be 1353, 849 or 11 corresponding to the value given in *spm_processingMode*.

numrows
The number of rows (spatial pixels) in the associated data array. This value is typically 1 for normal science operations but can assume any value between 1 and 286.

audit
Records the software/data versioning of this data record. The standard level 1 data processing will guarantee that the audit value only changes when OSIRIS is powered off. The Level 0 processing reserves *audit[0]* and the Level 1 processing reserves *audit[1]*. The value stored is the modified Julian date of the applicable level 0 or 1 software. More details can be found in the audit description.

data
OnyxDataObject object that provides access to the data array of **double[***numrows***][***numcolumns***]** using the ONYX interfaces

error
OnyxDataObject object that provides access to the error array of **double[***numrows***][***numcolumns***]** using the ONYX interfaces.

flags
OnyxDataObject object that provides access to the pixel flags array of **nxBYTE[***numrows***][***numcolumns***]** using the ONYX interfaces. Each nxBYTE provides 8 bits for t.b.d. exception flags specific to this pixel. A value of zero means the pixel is good. Any bit set in the array indicates earlier processing has identified an anomaly with the associated pixel. The values are defined below.

crosssection

OnyxDataObject object that provides access to a few slit cross-sections from the detector. It is only available when the **roe** mode is 0,1 or 2, spatial binning is enabled and wavelength binning is disabled. The number of columns depends upon the roe mode: roe 0 gives 32 columns, roe 1 gives 16 columns and roe 2 gives 8 columns.. The data are stored as an array of **double[***n***][***ncolumns***]** using the ONYX interfaces.

crosssectionindex

OnyxDataObject object The index of each column on the CCD in the **crosssectionindex** array. A valid index will be between 0 and 1352. The data are stored as an array of **nxWORD[***n***][***ncolumns***]** using the ONYX interfaces.

Exception Flags

OSIEX_SERIOUS (0x80000000)

Flags a serious error in the data and indicates that the data should be discarded.

OSIEX_OS_DUMPRAW (0x00000001)

flags that this data is a diagnostic "full R.O.E." data dump. It is a copy of an identical image

OSIEX_OS_BADDMA (0x00000004)

flags that the OS DMA channel failed in the OSIRIS firmware and that the data are probably useless.

OSIEX_OS_BADUTC (0x00000008)

flags that the **mjd** field may be inaccurate by an indeterminate amount.

Pixel Flags

The following bit values are defined for each element of the **flags** array.

OSPIX_FLAG_SEVERE (0x80)

The pixel has a severe error and is probably unsuitable for normal scientific analysis. This flag may appear in conjuction with other pixels.

OSPIX_FLAG_DATAMISSING (0x01)

The pixel has no associated data. This is typically used for regions not sent to the ground, e.g. the order sorter region from pixel 514 to 649. The flag will normally appear in conjuction with **OSPIX_FLAG_SEVERE.**

OSPIX_FLAG_OVERFLOW (0x02)

Flags that at least one constitute CCD pixel had overflowed in the construction of this data element. It is normally recommended that the user discard this pixel element. This flag will normally appear in conjuction with **OSPIX_FLAG_SEVERE.**

OSPIX_FLAG_OUTLIER (0x04)

Flags that this pixel appears to be an outlier probably due to a radiation hit. Proceed with caution.

OSPIX_FLAG_UNCALIBRATABLE (0x08)

Flags that the pixel was uncalibratable for some reason. It may or may not occur with flag **OSPIX_FLAG_SEVERE.** If it does not occur with **OSPIX_FLAG_SEVERE** then the calibration software has adjusted the error bars on this data point to properly account for the larger uncertainty in the measurement.

OSPIX_FLAG_RADIATIONHIT (0x10)

Flags that this pixel appears to be an outlier probably due to a radiation hit. Proceed with caution.

OSPIX_FLAG_NEGATIVEVALUE (0x20)

Flags that this pixel is negative. These pixels cause considerable annoyance for many applications that takes the log of radiance. Testing for this flag allows class Osiris_SpectrographScan to easily filter out negative value exposures.

Return to Structures and Enumerations

ODIN_SCAN_DIAGNOSTICS

Structure used to store L2 pre-processing archived diagnostic information. One entry is created for each scan analyzed. Note that not all scan entries in the OSIRIS database will have corresponding entries in the archived diagnostic database.

struct ODIN_SCAN_DIAGNOSTICS{

nxDWORD	*ScanNumber*;
IOnyxArray*	*albedo;*
IOnyxArray*	*albedowavelengths;*
IOnyxArray*	*cloudalt;*
IOnyxArray*	*cloudextinction;*
IOnyxArray*	*clouddetectedtype;*
double	*TH_offset*;
double	*MaxACSAltError*;
double	*MaxACSTimeGap*;
nxWORD	*MinStarsInACSFOV*;
double	*MoonDistToACSFOV*;

};

Entries

ScanNumber

A number that uniquely identifies this scan entry. This number is guaranteed to be identical to the corresponding scan number in the ODIN_SCAN_ENTRY structure.

albedo

An IOnyxArray expressing the albedo at the wavelengths specified in *albedowavelengths*. This array is returned by **OSAlbedoCalculationScan.**

albedowavelengths

The wavelengths at which the albedo is determined. This array is returned by **OSAlbedoCalculationScan.**

clouddetected

Returns an array of flags indicating the possible presence of a cloud in the field of view, with one entry for each tangent height in the scan. This is the array returned by **OSCloudDetectionScan**. Flags defined as follows:

0 – no cloud
1 – 'thin' cloud
2 – 'thick' cloud
-9999 – calculation failed.

cloudalt

Returns an array of tangent heights corresponding to the *clouddetection* and *cloudextinction* arrays in meters. This is the array returned by **OSCloudDetectionScan** converted to meters.

cloudextinction

Returns an array of calculated cloud extinctions in km^{-1}for each of the tangent heights in array *cloudalt.* This is the array returned by **OSCloudDetectionScan**.

TH_offset
The altitude offset in meters calculated using an RSAS algorithm. This is the value returned by **OSTangentOffsetScan.**

MaxACSAltError
Returns the maximum altitude error in meters reported by the ACS. This is the value returned by **OSACSDiagnosticScan**.

MaxACSTimeGap
Returns the maximum time gap between ACS measurements reported in the att/oat files. This is the value returned by **OSACSDiagnosticScan**.

MinStarsInACSFOV
Returns the minimum stars in the ACS FOV during the scan. This is the value returned by **OSACSDiagnosticScan**.

MoonDistToACSFOV
Returns the angular distance of the ACS ST2 from the center of the moon. This is the value returned by **OSACSDiagnosticScan**.

ODIN_SCAN_ENTRY

A structure used to represent one scan. A normal aeronomy scan is either the upward going or downward going section of the satellite nod. However a special scan type is reserved for stare mode and another type of scan is reserved for everything not covered by scanning or staring (like astronomy).

```
struct ODIN_SCAN_ENTRY{
nxDWORD              ScanNumber;
double               StartMJD;
double               EndMJD;
nxWORD               ACSState;
nxWORD               ACSConfig;
double               MinAltitude;
double               MaxAltitude;
double               ScanRate;
nxSHORT              Direction;
}
```

Entries

ScanNumber
: A number that uniquely identifies this scan entry. The first scan of an orbit is 1000 times the orbit number. The scan number increments by 1 for every entry in the orbit. IDL users should note that this number must be stored in a 32 bit integer.

StartMJD
: The start time of this scan entry expressed as a modified Julian date.

EndMJD
: The end time of this scan expressed as a Modified Julian Date. The end time of one scan is normally identical to the start time of the next scan. Occasionally, this is not true especially when striding across orbit boundaries or when stallite attitude information is missing.

ACSState
: This indicates the current state of the ACS system. It is one of the following values: limb scanning 1, limb-staring 2, other 3. We do not support astronomy modes.

ACSConfig
: This field is not properly implemented. It was intended to indicate the type of scanning being performed by the ACS system. It should be ignored.

MinAltitude
: The minimum tangent altitude in km of the spacecraft control frame during the scan. Note this is approximately 7 km below the OSIRIS spectrograph tangent altitude.

MaxAltitude

The maximum tangent altitude in km of the spacecraft control frame during the scan. Note this is approximately 7 km below the OSIRIS spectrograph tangent altitude.

ScanRate

The scanning rate for up or down scans. It is normally 0.75 km/s. It is undefined if ACSSTATE <> 1.

Direction

The scan direction. This number is only valid if (ACSState < 3). 1 is up, -1 is down and 0 is stare.

Return to Structures and Enumerations

OSIRIS_ECMWF

A structure used to store the ECMWF temperature and density profiles for a scan.

```
struct OSIRIS_ECMWF{
double              mjdofscan;
double              mjd;
double              latitude;
double              longitude;
nxDWORD             scannumber;
nxDWORD             numaltitudes;
IOnyxArray*         altitudes;
IOnyxArray*         density;
IOnyxArray*         temperature;
}
```

Entries

mjdofscan
This is the mjd at the beginning of the scan and is primarily used for database lookup purposes.

mjd
The time that represents the geographic location of the corresponding scan. This time is chosen to be the instant when the OSIRIS spectrograph's tangent point is at 30 km +/- 0.01 km. This time is expressed as a modified Julian date.

latitude
The geodetic latitude of the spectrograph tangent point at time mjd. The ECMWF profile has been extracted at this latitude. The value is expressed in degrees.

longitude
The geodetic longitude of the spectrograph tangent point at time mjd. The ECMWF profile has been extracted at this longitude. The value is expressed in degrees.

scannumber
The scan number associated with this record. Normally this number will be identical to the scan number retrieved from the Level 1 Services at time mjd; it is possible it can slightly disagree if the attitude for this orbit has been reprocessed since creation of this record.

numaltitude
The number of elements in the altitudes, density and temperature arrays

altitudes
The array of heights at which density and temperature are produced. The heights are in meters above the geoid, typically at 500 meter resolution. Note that this is substantially finer than the ECMWF height sampling.

density
The density profile extracted from the ECMWF model. The density is in molecules/cm3. The profile frequently has missing values at the top and bottom

of the profile. Missing values are expressed as a negative number (-9999999.0) and should be ignored by the user.

temperature

The temperature profile extracted from the ECMWF model. The temperature is in Kelvin. The profile frequently has missing values at the top and bottom o fthe profile. Missing values are expressed as a negative number (-9999999.0) and should be ignored by the user.

Return to Structures and Enumerations

QUATERNION

The **QUATERNION** structure is used to store the 4 component quaternion generated by the SSC attitude solution.

typedef double[4] QUATERNION;

Return to Structures and Enumerations

ONYX_VERSION_STRUCT

The primary function of OSIRIS Version Control is to provide anaudit trail for both the end user and the software developer. The OSIRIS database provides a simple structure, **ONYX_VERSION_STRUCT**, that provides this information. The structure is simply an array of 4 double precision numbers. Each number represents one section of the OSIRIS software:

Array[0]

Corresponds to the version of the OSIRIS level 0 decoding software (DLL). The double precision number is the time the level 0 Decode software was compiled and is expressed as a Modified Julian Date. The personnel responsible for level 0 processing will maintain a simple text file indicating which compiled versions are used during normal processing.

Array[1]

Corresponds to the version of the OSIRIS level 0-1 algorithms applied to the associated data. The double precision number is the time the level 0-1 software was compiled and is expressed as a Modified Julian Date. In practice this means a single instant in time will be associated with all of the level 0 to 1 algorithms: changing the code in one algorithm will change the version date of all of the algorithms.

Array[2]

Is not used by level 0-1

Array[3]

Is not used by level 0-1.

Return to Structures and Enumerations

OSIRIS LEVEL 1 API INFORMATION

This document is a summary of the software services that will be provided as part of the OSIRIS level 1 data products

Software Installation	Overview of software installation
Environment Variables	Overview of required environment variables
Software Usage	Overview of software usage
Software Linking	Instructions on linking software to projects
Level 1 Database Overview	Quick description of the Odin/OSIRIS database
Ephemeris Calculations	Description of ephemeris software

Required Environment Variables

Exported Environment Variable	***Windows***	***Unix***	***Brief Description***
LD_LIBRARY	No	Yes	Loader search path for shareable objects
LD_LIBRARY_PATH	No	Yes	Loader search path for shareable objects
ODINCDBDIR	Yes (manual)	Yes	Directory path for the ODIN Calibration Database files
ODINFLIGHTDIR	Yes (Manual)	Yes	Directory path for the ODIN flight duration files.
ODINORBITDIR	Yes (Manual)	Yes	Directory search path for the ODIN orbit duration files.
JPLEPH	Yes (installed)	Yes	Full path to the JPL DE200 binary ephemeride file.

Software Installation

Windows users can obtain pre-compiled binaries from our web site. Linux users or Windows users requiring access to the source code and headers can download tar-balls from our web site. The details of the Windows and Linux installations can be found on the web site

Software Usage

The code is developed using C++ and we expect C programs will be compiled with the C++ compiler.

We give a code snippet below that outlines the basic skeleton usage of the software

```
#define NX_INITGUID   // only define in one file of each program
#include "odin.h"

int main()
{
   IonyxDatabase* dbase;
   HRESULT        status;
   nxLogConsole   logger;

   CoInitialize( NULL );
   logger.AddRef();
   InitializeLevel1Services(&logger);

   ... do things

   UninitializeLevel1Services();
   CoUninitialize();
   return 0;
}
```

The following details should be noted:

1. You must have #define NX_INITGUID in one and only one of your source files. Failure to omit this may result in the linker not finding the various GUIDs such as **IID_IOnyxDatabase** or **IID_nxLog**.
2. The whole of the osiris/odin project may be included by placing #include odin.h in your source files. You must call **CoInitialize(NULL)** at the beginning of your program
3. You must call **CoUninitialize()** at the end of your program
4. The Level 1 services communicate errors through the **InxLog*** interface. You should pass a pointer to such an object before using the services so you can detect errors. Note that statically allocated loggers must have an additional call to **AddRef** to ensure they never attempt to self-destruct themselves when released by the Level1 Services or Onyx software.

NOTE: The OSIRIS Level 1 services, Onyx software and HDF libraries are not thread safe. You should ensure that only one thread of execution ever calls these utilities. In addition all level 1 service call and Onyx calls must reside within the same thread of execution.

Return to Odin/OSIRIS Level 1 API

Odin/OSIRIS Level 1 Database Overview

This section will provide an overview of the Odin/OSIRIS Level 1 database. The Odin/OSIRIS database will ultimately consist of several tens of thousands of files and may be distributed to several platforms. Portability of individual files is addressed through the consistent use of HDF. The Level 1 API reads and writes records from the HDF files using the onyx software developed by University of Saskatchewan.

The database as a single entity is glued together by the Odin/OSIRIS Level 1 services API. The overall paradigm adopted for the API is to treat the database as a single entity and alleviate the user from the burden of joining files together.

The Odin/OSIRIS level 1 API adopts the following policy for locating files.

1. Each database file will belong to one of three possible groups: **Orbital**, **Flight** or **Calibration**.
2. **Calibration** files will be located in one of the directories specified by environment variable `ODINCDBDIR`. The API will search the directories in the order specified in the environment variable. The intent is that all OSIRIS calibration database (CDB) products will be into this directory tree. It is expected that the entire calibration database will be on-line at all processing sites.
3. **Flight** files will be located in one of the directories specified by environment variable `ODINFLIGHTDIR`. The API will search the directories in the order specified in the environment variable. The intent is that the Flight directories will contain database metadata which are used as master indices for the rest of the database. The most notable example is the start and stop time of each orbit.
4. **Orbital** files follow a specific naming convention and contain all of the data for exactly one orbit. Each orbit is assigned a unique number that steadily increases during the mission (maximum value is about 10,000 for a two year mission). Each **Orbital** file will be located in one of the sub-directories of the directories specified by environment variable `ODINORBITDIR`. The API will search the directories in the order specified in the environment variable. In addition, the API will search a specific sub-directory of each directory. The API generates a sub-directory name by masking out the bottom 8 bits of the orbit number and writing the value to a 4 digit hexadecimal string with explicit leading zeroes. Hence orbit 690 (decimal) is 0x02B2 (hexadecimal) and the subdirectory is "0200". The search order is to search the parent directory, the sub-directory and then search the next directory on the search path. The API will also search sub-directories labeled "diskxx" where xx is a number starting with 1 and ascending upwards (eg, 1,2,3). No gaps are allowed in the ascending sequence.

This policy is reasonably flexible. It allows all data files to be located across multiple disk partitions. It also allows users to search their own personal directories before searching standard directories. The usage of orbital sub-directories avoids generating directories with thousands of entries, which can slow many systems. The extra level of hashing provided by the sub-directories should keep the system very responsive as no directory will have more than a thousand (or so) files.

Return to ODIN/OSIRIS Level 1 API

Ephemeris Calculations

The Odin/OSIRIS Level 1 API calculate ephemeris data using the U.S. Naval Observatory's NOVAS software which in turn calls the JPL Lunar and Planetary Ephemerides. We assume that the JPL DE200 ephemerides for the period 1980 to 2040 AD are available at each site. Each site is responsible for the installation of this software although we can provide limited assistance. To help software portability we demand that all sites define an environment variable JPLEPH with the intent that this environment variable will indicate the full path to the binary DE200 ephemeride file.

Version 2.0 of the NOVAS software has been used while the JPL ephemeris code is only available as Fortran. The NOVAS software is compiled "as is" but the JPL code must be modified for each platform so it can access the JPL DE200 ephemerides.

For the Windows NT version of the JPL ephemeris we have done the following:

1. Broken file testeph.f so that the main testeph program module is separated from the subroutines. The subroutines are stored in a file called pleph.f. The main test program is stored in testeph.f
2. The NREC variable is set to 4 in functions FSIZERx
3. Implemented function FSIZER3 in the subroutine STATE
4. Fetch the Direct Access Ephemeride Filename from a Environment variable called JPLEPH
5. Included a Microsoft Fortran attribute in subroutine PLEPH that exports the subroutine PLEPH as a "C" subroutine compatible with definition in NOVAS (ie. underscore prefix, lowercase, no stack decoration, i.e. _pleph)
6. Built the subroutines as a Windows DLL and the DLL placed on the system search path, i.e. environment variable PATH.
7. Verified the integrity of the build by running the testeph program.

All of these changes are "*permitted*" by the JPL software installation.

The NOVAS software uses TDT and TDB time scales. The Odin level 1 services will convert UTC to TDT and TDB to an accuracy no better than 2 seconds. This will introduce pointing errors of the order of hundredths or thousandths or arc seconds. A requirement for higher accuracy would introduce significant maintenance of time scale conversion tables. Details of time scales and conversion can be found in section L of the Astronomical Almanac.

The NOVAS software can be obtained from http://www.usno.navy.mil

In earlier releases of this document we noted that the value of **const double f** in subroutine **terra** was inconsistent with the IAU 1976 Geoid as used by SSC. After discussion with personnel from the USNO we have established that the value of f = 0.00335281 used in the NOVAS software is the correct value and we use the novas code as is.

We note that the NOVAS software has a namespace conflicts with the *nxlib* package. In particular odin.h invokes nxlib.h which includes nxmath.h which defines macro TWOPI. This macro conflicts with a definition in novas. We have disabled the nxmath version of TWOPI when compiling the novas software to ensure the highest degree of compatibility with novas. Both versions specify TWOPI to more than 15 decimal places.

Return to Odin/OSIRIS Level 1 API

Source Code Control

The Odin/OSIRIS level1 services API is now kept under source code control. We use Subversion at https://vidarr.usask.ca/svn/Repos_OSIRIS/. A low security, read-only account odin (with password hugin) can be used to access the repository

We shall release new versions of the Odin/OSIRIS level 1 services on specific dates and these dates will be tracked below. This document will be regarded as the master authority

Release Date	***Software RCS Label***	***Comments***	***Version ID***	***Document Version***
1999-09-24	V1999_09_24	Alpha	0x0000	1.6
2002-01-23	V2002_01_23	Beta	0x0001	2.4
2002-03-06		Beta		2.5
2002-06-24	V2002_06_24	Beta	V 1.03	2.6
2003-06-27		Released	V 1.06	2.8
2005-01-14		Released	V 1.08	2.9

Areas still requiring definition or clarification

- Ray tracing code in **LOSTangentPoint** and **LOSEntrancePoints** is yet to be implemented.
- isrectangular in **DefineFOVSearch** is yet to be implemented
- Define operating modes
- Define science programs
- Define ACDC modes
- **GetScienceProgram** and **GetOperatingMode** just return dummy values
- Description of filename scheme and directory structure

Document Update History

Version 1.4

1. Added function **GetGeoFromOrbitAngle**

Version 1.5

1. Removed field **spm_normalization** from structure OS_L1 as new firmware has eliminated this issue.
2. Removed changes to Earth flattening factor in NOVAS software as USNO clarified the accuracy.
3. Changed definition of **GetFOVSize** to account for rectangular field of view.
4. Changed **GetStarsInFOV** to account for rectangular field of view
5. Changed **GetPlanetInFOV** to account for rectangular field of view
6. Changed **CFToECI** to account for light aberration effects.
7. Added enumeration **ECI_REFERENCE_FRAME**
8. Added functions **InitializeLevel1Services** and **UninitializeLevel1Services**.
9. Changed **enum ODIN_POINTING_FRAME** so it indicates it orientation with respect to the spacecraft control frame.

Version 1.6

1. Added section of Source Code Control
2. Added section on Software installation
3. Added section Environment Variables
4. Added Section on level 1 Database Overview
5. Added **InitializeLevel1Services**
6. Added **UninitializeLevel1Services**
7. Added **GetLevel1Version**
8. Added **GetInstrumentXandYECI**
9. Added **GetStarsInInstrumentFOV**
10. Added GetPlanetInInstrumentFOV

Version 1.7

1. Added STW To UTC conversion functions, **StwLocateResetEpoch, StwUsesFixedResetEpoch** and **StwToUtc.**

Version 1.8

1. Changed shutter definition in OS_L1 so it is in accordance with data files.
2. Changed description of *scienceprog* in OS_L1 so it has no constraints on the upper bit.
3. Introduced the *slixsect* and *slixsectcolumnindex* fields into the OS_L1 structure. These record the CCD slit cross sections voted on in December 2000.
4. Added definition of OS_L0.

Version 1.9

1. Changed definition of OS_L0

Version 2.0

1. New linux installation procedure.
2. Added section on Software Usage.
3. Added section on Software Linking.
4. Added **CDBGen_OS_PointSpread**.
5. Added **CDBGen_OS_ReferenceSpectrum**.
6. Changed CDBGen functions to accommodate modes that have more than one row of data.
7. Added **GetScanInfo**.
8. Made multiple changes to make documentation consistent with software.

Version 2.1

1. Removed highres option from **CDBGen_OS_ReferenceSpectrum**.
2. Added **GetSolarAngles**.

Version 2.2

1. Changed **CDBGEN_OS_FlatFieldResponsivity**
2. Added **CDBGEN_OS_XSectionFlatField**
3. Changed definition of **OS_L1**

Version 2.3

1. Changed **CDBGen_OS_Wavelength**
2. Changed **GetScanInfo**
3. Changed almost all of the CDGBGEN_*** functions.

Version 2.4

Lots of tidying up.
Added ECItoOrbitalPlane

Version 2.5

- Modified CDBGEN_OS_ReferenceSpectrum interface definition.
- Added CDBGen_Molecule_CrossSection

Version 2.6

- Added GEOToECI, ECIToGEo, GeodeticToGeo, GeoToGeodetic OrbitalPlaneToECI and defined typedef GEOVECTOR

Version 2.7

- Changed error in documentation of enum ODIN_ISNTRUMENT

Version 2.8

- Minor updates to reflect supported compilers and document history.

Version 2.9

- Added description of ODIN_SCAN_ENTRY

Version 2.10

- Changed dark current description in CDBGEN_OS_DARKCURRENT

Version 2.11

- Added function GetOsirisEcmwf

Version 2.12

- Added descriptions of the audit fields

Version 2.13

- Added GetOSSlitCurvature
- Added GetPixelCFUnitVectorExt and deprecated GetPixelCFUnitVector

Version 2.14

- Added Matlab documentation

Version 2.15

- Modified CDBGEN_OS_REFERENCESPECTRUM so it returns Photons/cm2/sec/ster/nm rather than Watts/m**2/nm

Version 2.16

- Added function GetAttitudeError to source code and documentation

Version 2.17

- Modified Matalb Onyx code so it reads the audit field correctly
- Added support for flat-field calibration using atmospheric calibration

Version 2.18

- Added GetScanDiagnostics
- Added ST2_X_BORE and ST2_Y_BORE so we can get the control frame coordinates of star tracker 2.

Version 2.19

- Added strap temperature to OS_L1 structure

Version 2.20

- Updated document to match software version 2.02. Changes mostly in return values of CDBGEN_XXXX functions

Version 2.21

- Removed ancient installation instructions.

Version 2.23

- Added OSIRISAvgTemperature
- Added BlankOutRadiationHitPixels
- Added GetUVPSF
- Added a couple of pixel flag exceptions.