

**162-WP-005-001**

# **HDF-EOS Interface Based on HDF5**

## **Volume 2: Function Reference Guide**

### **White Paper**

***White paper - Not intended for formal review  
or Government approval.***

**December 1999**

Prepared Under Contract NAS5-60000

#### **RESPONSIBLE ENGINEER**

Larry Klein /s/

12/16/99

David Wynne, Alex Muslimov, Abe Taaheri,                      Date  
Ray Milburn, Larry Klein  
EOSDIS Core System Project

#### **SUBMITTED BY**

Robert Plante /s/

12/16/99

Robert Plante, Manager, Science Office                      Date  
EOSDIS Core System Project

Raytheon Systems Company  
Upper Marlboro, Maryland

This page intentionally left blank.

# Preface

---

This document is a Users Guide for HDF-EOS (Hierarchical Data Format - Earth Observing System) library tools. The version described in this document is HDF-EOS Version 3.0. The software is based on HDF5, a new version of HDF provided by NCSA. HDF5 is a complete rewrite of the earlier HDF4 version, containing a different data model and user interface. HDF-EOS V3.0 incorporates HDF5, and keeps the familiar HDF4-based interface. There are a few exceptions and these exceptions are described in this document. Note that the contents of this document describe a prototype library, which is not yet operational. Release of an operational version is subject to NASA approval, but is expected in the summer of 2000.

HDF is the scientific data format standard selected by NASA as the baseline standard for EOS. This Users Guide accompanies Version 3 software, which is available to the user community on the EDHS1 server. This library is aimed at EOS data producers and consumers, who will develop their data into increasingly higher order products. These products range from calibrated Level 1 to Level 4 model data. The primary use of the HDF-EOS library will be to create structures for associating geolocation data with their associated science data. This association is specified by producers through use of the supplied library. Most EOS data products which have been identified, fall into categories of point, grid or swath structures, the latter two of which are implemented in the current version of the library. Services based on geolocation information will be built on HDF-EOS structures. Producers of products not covered by these structures, e.g. non-geolocated data, can use the standard HDF libraries.

In the ECS (EOS Core System) production system, the HDF-EOS library will be used in conjunction with SDP (Science Data Processing) Toolkit software. The primary tools used in conjunction with HDF-EOS library will be those for metadata handling, process control and status message handling. Metadata tools will be used to write ECS inventory and granule specific metadata into HDF-EOS files, while the process control tools will be used to access physical file handles used by the HDF tools. (SDP Toolkit Users Guide for the ECS Project, June 1999, 333-CD-500-001).

HDF-EOS is an extension of NCSA (National Center for Supercomputing Applications) HDF and uses HDF library calls as an underlying basis. Version 5-1.2.0 of HDF is used. The library tools are written in the C language and a FORTRAN interface is provided. The current version contains software for creating, accessing and manipulating Grid and Swath structures. This document includes overviews of the interfaces, and code examples. EOSView, the HDF-EOS viewing tool, has been revised to accommodate the current version of the library.

Technical Points of Contact within EOS are:

Larry Klein, larry@eos.hitc.com  
David Wynne, davidw@eos.hitc.com

An email address has been provided for user help:

[pgstlkit@eos.hitc.com](mailto:pgstlkit@eos.hitc.com)

Any questions should be addressed to:

Data Management Office  
The ECS Project Office  
Raytheon Systems Company  
1616 McCormick Drive  
Upper Marlboro, MD 20774-5301

## Abstract

---

This document will serve as the user's guide to the prototype HDF-EOS file access library based on HDF5. HDF refers to the scientific data format standard selected by NASA as the baseline standard for EOS, and HDF-EOS refers to EOS conventions for using HDF. This document will provide information on the use of the two interfaces included in this version – Swath, and Grid – including overviews of the interfaces, and code examples. This document should be suitable for use by data producers and data users alike.

**Keywords:** HDF-EOS, HDF5, Metadata, Standard Data Format, Standard Data Product, Disk Format, Grid, Swath, Projection, Array, Browse

This page intentionally left blank

# **Contents**

---

## **Preface**

## **Abstract**

### **1. Introduction**

1.1	Purpose .....	1-1
1.2	Organization .....	1-1
1.3	Swath Data .....	1-1
1.3.1	The Swath Data Interface .....	1-1
1.3.2	List of SW API Routines.....	1-2
1.4	Grid Data .....	1-3
1.4.1	The Grid Data Interface .....	1-3
1.4.2	List of Grid API ROUTINES.....	1-3
1.5	GCTP Usage .....	1-5
1.5.1	GCTP Projection Codes .....	1-5
1.5.2	UTM Zone Codes .....	1-6
1.5.3	GCTP Spheroid Codes.....	1-6
1.5.4	GCTP Projection Parameters.....	1-7

### **2. Function Reference**

2.1	Format .....	2-1
2.1.1	Swath Interface Functions.....	2-1
2.1.2	Grid Interface Functions.....	2-67
2.1.3	HDF-EOS Utility Routines .....	2-123

## **List of Tables**

1-3. Summary of the Swath Interface.....	1-2
1-4. Summary of the Grid Interface.....	1-4
1-5. Projection Transformation Package Projection Parameters .....	1-7

## **Abbreviations and Acronyms**

# 1. Introduction

---

## 1.1 Purpose

The *HDF-EOS Software Reference Guide for the ECS Project* was prepared under the Earth Observing System Data and Information System (EOSDIS) Core System (ECS), Contract (NAS5-60000).

This software reference guide is intended for use by anyone who wishes to use the HDF-EOS library to create or read EOS data products. Users of this document will include EOS instrument team science software developers and data product designers, DAAC personnel, and end users of EOS data products such as scientists and researchers.

## 1.2 Organization

This paper is organized as follows:

- Section 1      Introduction - Presents Scope and Purpose of this document
- Section 2      Function Reference
- Abbreviations and Acronyms

## 1.3 Swath Data

The SW (*Swath*) interface consists of routines for storing, retrieving, and manipulating data in swath data sets. This interface is tailored to support time-ordered data such as satellite swaths (which consist of a time-ordered series of scanlines), or profilers (which consist of a time-ordered series of profiles). See the Users' Guide, Volume 1 that accompanies this document for more information.

### 1.3.1 The Swath Data Interface

All C routine names in the swath data interface have the prefix “SW” and the equivalent FORTRAN routine names are prefixed by “sw.” The SW routines are classified into the following categories:

- **Access routines** initialize and terminate access to the SW interface and swath data sets (including opening and closing files).
- **Definition** routines allow the user to set key features of a swath data set.
- **Basic I/O** routines read and write data and metadata to a swath data set.
- **Inquiry** routines return information about data contained in a swath data set.
- **Subset** routines allow reading of data from a specified geographic region.

### 1.3.2 List of SW API Routines

The SW function calls are listed below in Table 1-3 and are described in detail in Section 2 of this document. The listing in Section 2 is in alphabetical order.

**Table 1-3. Summary of the Swath Interface (1 of 2)**

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Access	SWopen	swopen	Opens or creates HDF file in order to create, read, or write a swath	2-44
	SWcreate	swcreate	Creates a swath within the file	2-6
	Swattach	swattach	Attaches to an existing swath within the file	2-2
	SWdetach	swdetach	Detaches from swath interface	2-24
	SWclose	swclose	Closes file	2-4
Definition	SWdefdim	swdefdim	Defines a new dimension within the swath	2-13
	SWdefdimmap	swdefmap	Defines the mapping between the geolocation and data dimensions	2-14
	SWdefidxmap	swdefimap	Defines a non-regular mapping between the geolocation and data dimension	2-18
	SWdefgeofield	swdefgfd	Defines a new geolocation field within the swath	2-16
	SWdefdatafield	swdefdfld	Defines a new data field within the swath	2-11
	SWdefcomp	swdefcomp	Defines a field compression scheme	2-9
Basic I/O	SWwritefield	swwrfl	Writes data to a swath field	2-65
	SWreadfield	swrdfl	Reads data from a swath field.	2-54
	SWwriteattr	swwrattr	Writes/updates attribute in a swath	2-63
	SWreadattr	swrdattr	Reads attribute from a swath	2-53
	SWsetfillvalue	swsetfill	sets fill value for the specified field	2-60
	SWgetfillvalue	swgetfill	Retrieves fill value for the specified field	2-31
Inquiry	SWinqdims	swinqdims	Retrieves information about dimensions defined in swath	2-37
	SWinqmaps	swinqmaps	Retrieves information about the geolocation relations defined	2-40
	SWinqidxmaps	swinqimaps	Retrieves information about the indexed geolocation/data mappings defined	2-39
	SWinqgeofields	swinqgfls	Retrieves information about the geolocation fields defined	2-38
	SWinqdatafields	swinqdfld	Retrieves information about the data fields defined	2-35
	SWinqdatatype	swinqdtype	Retrieves information about data type of a field	2-36
	SWinqattrs	swinqattrs	Retrieves number and names of attributes defined	2-34
	SWnentries	swnentries	Returns number of entries and descriptive string buffer size for a specified entity	2-43
	SWdiminfo	swdiminfo	Retrieve size of specified dimension	2-25
	SWmapinfo	swmapinfo	Retrieve offset and increment of specified geolocation mapping	2-42
	SWidxmapinfo	swimapinfo	Retrieve offset and increment of specified geolocation mapping	2-33
	SWattrinfo	swattninfo	Returns information about swath attributes	2-3
	SWfieldinfo	swfldinfo	Retrieve information about a specific geolocation or data field	2-29
	SWcompinfo	swcompinfo	Retrieve compression information about a field	2-5
Ragged Arrays	SWinqswath	swinqswath	Retrieves number and names of swaths in file	2-41
	Swregionindex	swregidx	Returns information about the swath region ID	2-56
	SWupdateidxmap	swupimap	Update map index for a specified region	2-61
	SWradefine	swradefine	Defines ragged array	2-48
Ragged Arrays	SWraopen	swraopen	Opens ragged array	2-49
	SWrawrite	swrawrite	Writes data to the ragged array	2-52

**Table 1-3. Summary of the Swath Interface (2 of 2)**

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
	SWraread	swraread	Reads out data from the ragged array	2-50
	SWraclose	swraclose	Closes the ragged array	2-47
Subset	SWgeomapinfo	swgmapinfo	Retrieves type of dimension mapping when first dimension is geodim	2-32
	SWdefboxregion	swdefboxreg	Define region of interest by latitude/longitude	2-7
	SWregioninfo	swreginfo	Returns information about defined region	2-58
	SWextractregion	swextreg	read a region of interest from a field	2-28
	SWdeftimeperiod	swdeftmeper	Define a time period of interest	2-19
	SWperiodinfo	swperinfo	Retuns information about a defined time period	2-45
	SWextractperiod	swextper	Extract a defined time period	2-27
	SWdefvrtregion	swdefvrtreg	Define a region of interest by vertical field	2-21
	SWdupregion	swdupreg	Duplicate a region or time period	2-26

## 1.4 Grid Data

The GD (*Grid*) interface consists of routines for storing, retrieving, and manipulating data in grid data sets. This interface is designed to support data that has been stored in a rectilinear array based on a well defined and explicitly supported projection. See the Users' Guide, Volume 1 that accompanies this document for more details.

### 1.4.1 The Grid Data Interface

All C routine names in the grid data interface have the prefix “GD” and the equivalent FORTRAN routine names are prefixed by “gd.” The GD routines are classified into the following categories:

- **Access routines** initialize and terminate access to the GD interface and grid data sets (including opening and closing files).
- **Definition** routines allow the user to set key features of a grid data set.
- **Basic I/O** routines read and write data and metadata to a grid data set.
- **Inquiry** routines return information about data contained in a grid data set.
- **Subset** routines allow reading of data from a specified geographic region.

### 1.4.2 List of Grid API ROUTINES

The GD function calls are listed below in Table 1-4 and are described in detail in Section 2 of this document. The listing in Section 2 is in alphabetical order.

**Table 1-4. Summary of the Grid Interface (1 of 2)**

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Access	GDopen	gdopen	Creates a new file or opens an existing one	2-108
	GDcreate	gdcreate	Creates a new grid in the file	2-72
	GDattach	gdattach	Attaches to a grid	2-68
	GDdetach	gddetach	Detaches from grid interface	2-92
	GDclose	gdclose	Closes file	2-70
Definition	GDdeforigin	gddeforigin	Defines origin of grid	2-81
	GDdefdim	gddefdim	Defines dimensions for a grid	2-78
	GDdefproj	gddefproj	Defines projection of grid	2-83
	GDdefpixreg	gddefpixreg	Defines pixel registration within grid cell	2-82
	GDdeffield	gddeffld	Defines data fields to be stored in a grid	2-79
	GDdefcomp	gddefcomp	Defines a field compression scheme	2-76
Basic I/O	GDwritefield	gdwrfld	Writes data to a grid field.	2-121
	GDreadfield	gdrdfld	Reads data from a grid field	2-114
	GDwriteattr	gdwrattr	Writes/updates attribute in a grid.	2-119
	GDreadattr	gdrdattr	Reads attribute from a grid	2-113
	GDsetfillvalue	gdsetfill	sets fill value for the specified field	2-118
	GDgetfillvalue	gdgetfill	Retrieves fill value for the specified field	2-98
Inquiry	GDinqdims	gdinqdims	Retrieves information about dimensions defined in grid	2-104
	GDinqfields	gdinqflds	Retrieves information about the data fields defined in grid	2-105
	GDinqattrs	gdinqattrs	Retrieves number and names of attributes defined	2-102
	GDnentries	gdnentries	Returns number of entries and descriptive string buffer size for a specified entity	2-107
	GDgridinfo	gdgridinfo	Returns dimensions of grid and X-Y coordinates of corners	2-101
	GDprojinfo	gdprojinfo	Returns all GCTP projection information	2-112
	GDdiminfo	gddiminfo	Retrieves size of specified dimension.	2-93
	GDcompinfo	gdcompinfo	Retrieve compression information about a field	2-71
	GDfieldinfo	gdfldinfo	Retrieves information about a specific geolocation or data field in the grid	2-96
	GDinqdatatype	gdinqdtype	Retrieves information about data type of a field	2-103
	GDinqgrid	gdinqgrid	Retrieves number and names of grids in file	2-106
	GDattrinfo	gdattrinfo	Returns information about grid attributes	2-69
	GDorigininfo	gdorginfo	Return information about grid origin	2-110
	GDpixreginfo	gdpreginfo	Return pixel registration information for given grid	2-111
	GDdefboxregion	gddefboxreg	Define region of interest by latitude/longitude	2-75
	GDregioninfo	gdreginfo	Returns information about a defined region	2-116

**Table 5-1. Summary of the Grid Interface (2 of 2)**

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Subset	GDextractregion	gdextrreg	read a region of interest from a field	2-95
	GDdeftimeperiod	gddeftmep	Define a time period of interest	2-87
	GDdefvrtregion	gddefvrtreg	Define a region of interest by vertical field	2-89
	GDgetpixels	gdgetpix	get row/columns for lon/lat pairs	2-99
	GDdupregion	gddupreg	Duplicate a region or time period	2-94
	GDdeftile	gddeftle	Define a tiling scheme	2-85

## 1.5 GCTP Usage

The HDF-EOS Grid API uses the U.S. Geological Survey General Cartographic Transformation Package (GCTP) to define and subset grid structures. This section described codes used by the package.

### 1.5.1 GCTP Projection Codes

The following GCTP projection codes are used in the grid API described in Section 4 below:

GCTP_GEO	( 0 )	Geographic
GCTP_UTM	( 1 )	Universal Transverse Mercator
GCTP_LAMCC	( 4 )	Lambert Conformal Conic
GCTP_PS	( 6 )	Polar Stereographic
GCTP_POLYC	( 7 )	Polyconic
GCTP_TM	( 9 )	Transverse Mercator
GCTP_LAMAZ	( 11 )	Lambert Azimuthal Equal Area
GCTP_HOM	( 20 )	Hotine Oblique Mercator
GCTP_SOM	( 22 )	Space Oblique Mercator
GCTP_GOOD	( 24 )	Interrupted Goode Homolosine
GCTP_ISINUS	( 99 )	Intergerized Sinusoidal Projection*

\* The Intergerized Sinusoidal Projection is not part of the original GCTP package. It has been added by ECS. See *Level-3 SeaWiFS Data Products: Spatial and Temporal Binning Algorithms*. Additional references are provided in Section 2.

Note that other projections supported by GCTP will be adapted for HDF-EOS Version 3 as new user requirements are surfaced. For further details on the GCTP projection package, please refer to Section 6.3.4 and Appendix G of the SDP Toolkit Users Guide for the ECS Project, June 1999, (333-CD-500-001).

### 1.5.2 UTM Zone Codes

The Universal Transverse Mercator (UTM) Coordinate System uses zone codes instead of specific projection parameters. The table that follows lists UTM zone codes as used by GCTP Projection Transformation Package. C.M. is Central Meridian

Zone	C.M.	Range	Zone	C.M.	Range
01	177W	180W-174W	31	003E	000E-006E
02	171W	174W-168W	32	009E	006E-012E
03	165W	168W-162W	33	015E	012E-018E
04	159W	162W-156W	34	021E	018E-024E
05	153W	156W-150W	35	027E	024E-030E
06	147W	150W-144W	36	033E	030E-036E
07	141W	144W-138W	37	039E	036E-042E
08	135W	138W-132W	38	045E	042E-048E
09	129W	132W-126W	39	051E	048E-054E
10	123W	126W-120W	40	057E	054E-060E
11	117W	120W-114W	41	063E	060E-066E
12	111W	114W-108W	42	069E	066E-072E
13	105W	108W-102W	43	075E	072E-078E
14	099W	102W-096W	44	081E	078E-084E
15	093W	096W-090W	45	087E	084E-090E
16	087W	090W-084W	46	093E	090E-096E
17	081W	084W-078W	47	099E	096E-102E
18	075W	078W-072W	48	105E	102E-108E
19	069W	072W-066W	49	111E	108E-114E
20	063W	066W-060W	50	117E	114E-120E
21	057W	060W-054W	51	123E	120E-126E
22	051W	054W-048W	52	129E	126E-132E
23	045W	048W-042W	53	135E	132E-138E
24	039W	042W-036W	54	141E	138E-144E
25	033W	036W-030W	55	147E	144E-150E
26	027W	030W-024W	56	153E	150E-156E
27	021W	024W-018W	57	159E	156E-162E
28	015W	018W-012W	58	165E	162E-168E
29	009W	012W-006W	59	171E	168E-174E
30	003W	006W-000E	60	177E	174E-180W

### 1.5.3 GCTP Spheroid Codes

Clarke 1866 (default)	( 0 )
Clarke 1880	( 1 )
Bessel	( 2 )
International 1967	( 3 )
International 1909	( 4 )
WGS 72	( 5 )
Everest	( 6 )
WGS 66	( 7 )
GRS 1980	( 8 )
Airy	( 9 )
Modified Airy	( 10 )

Modified Everest	(11)
WGS 84	(12)
Southeast Asia	(13)
Australasian National	(14)
Krassovsky	(15)
Hough	(16)
Mercury 1960	(17)
Modified Mercury 1968	(18)
Sphere of Radius 6370997m	(19)

#### 1.5.4 GCTP Projection Parameters

**Table 1-5. Projection Transformation Package Projection Parameters (1 of 2)**

Code & Projection Id	Array Element							
	1	2	3	4	5	6	7	8
0 Geographic								
1 UTM	Lon/Z	Lat/Z						
4 Lambert Conformal C	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
6 Polar Stereographic	SMajor	SMinor			LongPol	TrueScale	FE	FN
7 Polyconic	SMajor	SMinor			CentMer	OriginLat	FE	FN
9 Transverse Mercator	SMajor	SMinor	Factor		CentMer	OriginLat	FE	FN
11 Lambert Azimuthal	Sphere				CentLon	CenterLat	FE	FN
20 Hotin Oblique Merc A	SMajor	SMinor	Factor			OriginLat	FE	FN
20 Hotin Oblique Merc B	SMajor	SMinor	Factor	AziAng	AzmthPt	OriginLat	FE	FN
22 Space Oblique Merc A	SMajor	SMinor		IncAng	AscLong		FE	FN
22 Space Oblique Merc B	SMajor	SMinor	Satnum	Path			FE	FN
24 Interrupted Goode	Sphere							
99 Integerized Sinusoidal	Sphere				CentMer		FE	FN

**Table 1-5. Projection Transformation Package Projection Parameters (2 of 2)**

	Array Element				
Code & Projection Id	9	10	11	12	13
0 Geographic					
1 U T M					
4 Lambert Conformal C					
6 Polar Stereographic					
7 Polyconic					
9 Transverse Mercator					
11 Lambert Azimuthal					
20 Hotin Oblique Merc A	Long1	Lat1	Long2	Lat2	Zero
20 Hotin Oblique Merc B					One
22 Space Oblique Merc A	PSRev	Srat	PFlag		Zero
22 Space Oblique Merc B					One
24 Interrupted Goode					
99 Integerized Sinusoidal	NZone		RFlag		

Where,

- Lon/Z      Longitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
- Lat/Z      Latitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
- Smajor     Semi-major axis of ellipsoid. If zero, Clarke 1866 in meters is assumed.
- Sminor     Eccentricity squared of the ellipsoid if less than zero, if zero, a spherical form is assumed, or if greater than zero, the semi-minor axis of ellipsoid.
- Sphere     Radius of reference sphere. If zero, 6370997 meters is used.
- STDPR1    Latitude of the first standard parallel
- STDPR2    Latitude of the second standard parallel

CentMer	Longitude of the central meridian
OriginLat	Latitude of the projection origin
FE	False easting in the same units as the semi-major axis
FN	False northing in the same units as the semi-major axis
TrueScale	Latitude of true scale
LongPol	Longitude down below pole of map
Factor	Scale factor at central meridian (Transverse Mercator) or center of projection (Hotine Oblique Mercator)
CentLon	Longitude of center of projection
CenterLat	Latitude of center of projection
Long1	Longitude of first point on center line (Hotine Oblique Mercator, format A)
Long2	Longitude of second point on center line (Hotine Oblique Mercator, frmt A)
Lat1	Latitude of first point on center line (Hotine Oblique Mercator, format A)
Lat2	Latitude of second point on center line (Hotine Oblique Mercator, format A)
AziAng	Azimuth angle east of north of center line (Hotine Oblique Mercator, frmt B)
AzmthPt	Longitude of point on central meridian where azimuth occurs (Hotine Oblique Mercator, format B)
IncAng	Inclination of orbit at ascending node, counter-clockwise from equator (SOM, format A)
AscLong	Longitude of ascending orbit at equator (SOM, format A)
PSRev	Period of satellite revolution in minutes (SOM, format A)
SRat	Satellite ratio to specify the start and end point of x,y values on earth surface (SOM, format A -- for Landsat use 0.5201613)
PFlag	End of path flag for Landsat: 0 = start of path, 1 = end of path (SOM, frmt A)
Satnum	Landsat Satellite Number (SOM, format B)
Path	Landsat Path Number (Use WRS-1 for Landsat 1, 2 and 3 and WRS-2 for Landsat 4 and 5.) (SOM, format B)

Nzone	Number of equally spaced latitudinal zones (rows); must be two or larger and even
Rflag	Right justify columns flag is used to indicate what to do in zones with an odd number of columns. If it has a value of 0 or 1, it indicates the extra column is on the right (zero) or left (one) of the projection Y-axis. If the flag is set to 2 (two), the number of columns are calculated so there are always an even number of columns in each zone.

**Notes:**

- Array elements 14 and 15 are set to zero.
- All array elements with blank fields are set to zero.

All angles (latitudes, longitudes, azimuths, etc.) are entered in packed degrees/ minutes/ seconds (DDDDMMSSSS.SS) format.

The following notes apply to the Space Oblique Mercator A projection:

- A portion of Landsat rows 1 and 2 may also be seen as parts of rows 246 or 247. To place these locations at rows 246 or 247, set the end of path flag (parameter 11) to 1--end of path. This flag defaults to zero.
- When Landsat-1,2,3 orbits are being used, use the following values for the specified parameters:
  - Parameter 4 099005031.2
  - Parameter 5 128.87 degrees - (360/251 \* path number) in packed DMS format
  - Parameter 9 103.2669323
  - Parameter 10 0.5201613
- When Landsat-4,5 orbits are being used, use the following values for the specified parameters:
  - Parameter 4 098012000.0
  - Parameter 5 129.30 degrees - (360/233 \* path number) in packed DMS format
  - Parameter 9 98.884119
  - Parameter 10 0.5201613

## 2. Function Reference

---

### 2.1 Format

This section contains a function-by-function reference for each interface in the HDF-EOS library. Each function has a separate page describing it (in some cases there are multiple pages). Each page contains the following information (in order):

- Function name as used in C
- Function declaration in ANSI C format
- Description of each argument
- Purpose of routine
- Description of returned value
- Description of the operation of the routine
- A short example of how to use the routine in C
- The FORTRAN declaration of the function and arguments
- An equivalent FORTRAN example

#### 2.1.1 Swath Interface Functions

This section contains an alphabetical listing of all the functions in the Swath interface. The functions are alphabetized based on their C-language names.

# Attach to an Existing Swath Structure

---

## SWattach

hid\_t SWattach(hid\_t *fid*, char \**swathname*)

<i>fid</i>	IN: Swath file id returned by SWopen
<i>swathname</i>	IN: Name of swath to be attached
Purpose	Attaches to an existing swath within the file.
Return value	Returns the swath handle (swathID) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath file id or swath name.
Description	This routine attaches to the swath using the <i>swathname</i> parameter as the identifier.
Example	In this example, we attach to the previously created swath, "ExampleSwath", within the HDF file, <i>SwathFile.h5</i> , referred to by the handle, <i>fid</i> :

```
swathID = SWattach(fid, "ExampleSwath");
```

The swath can then be referenced by subsequent routines using the handle, *swathID*.

**FORTRAN**      integer function swattach(*fid*,*swathname*)  
                  integer        *fid*  
                  character\*(\*) *swathname*

The equivalent *FORTRAN* code for the example above is:

```
swathid = swattach(fid, "ExampleSwath")
```

# Return Information About a Swath Attribute

---

## **SWattrinfo**

```
herr_t SWattrinfo(hid_t swathID, const char *attrname, H5T_class_t *  
    numbertype, hsize_t *count)
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>attrname</i>	IN: Attribute name
<i>numbertype</i>	OUT: Data type class ID of attribute
<i>count</i>	OUT: Number of attribute elements
Purpose	Returns information about a swath attribute
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise.
Description	This routine returns number type and number of elements (count) of a swath attribute.
Example	In this example, we return information about the <i>ScalarFloat</i> attribute.

```
status = SWattrinfo(swathID, "ScalarFloat", &nt, &count);
```

The *nt* variable will have the value 1 and *count* will have the value 1.

**FORTRAN**

```
integer function swattrinfo(swathid, attrname, ntype, count,)  
integer      swathid  
character(*) attrname  
integer      ntype  
integer *4    count
```

The equivalent *FORTRAN* code for the first example above is:

```
status = swattrinfo(swathid, "ScalarFloat", nt, count);
```

# Close an HDF-EOS File

---

## SWclose

herr\_t SWclose(hid\_t *fid*)

*fid* IN: Swath file id returned by SWopen

Purpose Closes file.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.

Description This routine closes the HDF swath file.

Example

```
status = swclose(fid);
```

FORTRAN integer function swclose(fid)  
integer fid

The equivalent *FORTRAN* code for the example above is:

```
status = swclose(fid)
```

# Retrive Compression Information for Field

---

## SWcompinfo

```
herr_t SWcompinfo(hid_t swathID, char *fieldname, int32_t *compcode, intn  
          compparm[J])
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>fieldname</i>	IN: Fieldname
<i>compcode</i>	OUT: HDF compression code
<i>compparm</i>	OUT: Compression parameters
Purpose	Retrieves compression information about a field.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description	This routine returns the compression code and compression parameters for a given field.
Example	To retreive the compression information about the <i>Opacity</i> field defined in the <i>SWdefcomp</i> section:

```
status = SWcompinfo(swathID, "Opacity", &compcode,  
                  compparm);
```

The *compcode* parameter will be set to 4 and *compparm*[0] to 5.

**FORTRAN**

```
integer function swcompinfo(gridid,fieldname compcode, compparm)  
                  swathid  
                  character*(*) fieldname  
                  integer*4    compcode  
                  integer      compparm
```

The equivalent *FORTRAN* code for the example above is:

```
status = swcompinfo(swathid, 'Opacity', compcode, compparm)
```

The *compcode* parameter will be set to 4 and *compparm*(1) to 5.

# Create a New Swath Structure

---

## SWcreate

hid\_t SWcreate(hid\_t *fid*, const char \**swathname*)

<i>fid</i>	IN: Swath file id returned by SWopen
<i>swathname</i>	IN: Name of swath to be created
Purpose	Creates a swath within the file.
Return value	Returns the swath handle ( <i>swathID</i> ) if successful or FAIL (-1) otherwise.
Description	The swath is created as a Vgroup within the HDF file with the name <i>swathname</i> and class <i>SWATH</i> .
Example	In this example, we create a new swath structure, <i>ExampleSwath</i> , in the previously created file, <i>SwathFile.h5</i> .

```
swathID = SWcreate(fid, "ExampleSwath");
```

The swath structure is referenced by subsequent routines using the handle, *swathID*.

**FORTRAN**      integer function swcreate(*fid,swathname*)  
                  integer        *fid*  
                  character\*(\*) *swathname*

The equivalent *FORTRAN* code for the example above is:

```
swathid = swcreate(fid, "ExampleSwath")
```

# Define a Longitude-Latitude Box Region for a Swath

---

## SWdefboxregion

```
int32_t SWdefboxregion(hid_t swathID, float64 cornerlon[], float64 cornerlat[],  
int32_t mode)
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>cornerlon</i>	IN: Longitude in decimal degrees of box corners
<i>cornerlat</i>	Latitude in decimal degrees of box corners
<i>mode</i>	Cross Track inclusion mode
Purpose	Defines a longitude-latitude box region for a swath.
Return value	Returns the swath region ID if successful or FAIL (-1) otherwise.
Description	This routine defines a longitude-latitude box region for a swath. It returns a swath region ID which is used by the <i>SWextractregion</i> routine to read all the entries of a data field within the region. A cross track is within a region if 1) its midpoint is within the longitude-latitude "box" (HDFE_MIDPOINT), or 2) either of its endpoints is within the longitude-latitude "box" (HDFE_ENDPOINT), or 3) any point of the cross track is within the longitude-latitude "box" (HDFE_ANYPOINT), depending on the inclusion mode designated by the user. All elements within an included cross track are considered to be within the region even though a particular element of the cross track might be outside the region. The swath structure must have both <i>Longitude</i> and <i>Latitude</i> (or <i>Colatitude</i> ) fields defined

Note: Users who are defining subset regions involving scenes with overlaps should add a call to the routine in *SWupdatescene* after calling this routine in order to get correctly defined region.

Example	In this example, we define a region bounded by the 3 degrees longitude, 5 degrees latitude and 7 degrees longitude, 12 degrees latitude. We will consider a cross track to be within the region if its midpoint is within the region.
---------	---

```
cornerlon[0] = 3.;  
cornerlat[0] = 5.;  
cornerlon[1] = 7.;  
cornerlat[1] = 12.;
```

```
regionID = SWdefboxregion(swathID, cornerlon, cornerlat,  
    HDFE_MIDPOINT);
```

**FORTRAN**    integer\*4 function swdefboxreg(*swathid*, *cornerlon*, *cornerlat*, *mode*)  
                Integer        *swathid*  
                real\*8        *cornerlon*  
                real\*8        *cornerlat*  
                integer\*4      *mode*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_MIDPOINT=0)  
  
cornerlon(1) = 3.  
  
cornerlat(1) = 5.  
  
cornerlon(2) = 7.  
  
cornerlat(2) = 12.  
  
regionid = swdefboxreg(swathid, cornerlon, cornerlat,  
    HDFE_MIDPOINT)
```

# Set Swath Field Compression

---

## SWdefcomp

herr\_t SWdefcomp(hid\_t *swathID*, int32\_t *compcode*, intn \**compparm*)

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>compcode</i>	IN: HDF compression code
<i>compparm</i>	IN: Compression parameters (if applicable)
Purpose	Sets the field compression for all subsequent field definitions.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description	This routine sets the HDF field compression for subsequent swath field definitions. The compression does not apply to one-dimensional fields. The compression schemes currently supported are:  (HDFE_COMP_DEFLATE=4) and no compression (HDFE_COMP_NONE = 0, the default). Deflate compression requires a single integer compression parameter in the range of one to nine with higher values corresponding to greater compression. Compressed fields are written using the standard SWwritefield routine, however, the entire field must be written in a single call. Any portion of a compressed field can then be accessed with the SWreadfield routine. Compression takes precedence over merging so that multi-dimensional fields that are compressed are not merged. The user should refer to the HDF Reference Manual for a fuller explanation of the compression schemes and parameters.
Example	Suppose we wish to compress the <i>Pressure</i> using run length encoding, the <i>Opacity</i> field using deflate compression, the <i>Spectra</i> field with skipping Huffman compression, and use no compression for the <i>Temperature</i> field.  status = SWdefcomp(swathID, 4, NULL);  status = SWdefdatafield(swathID, "Pressure", "Track,Xtrack", NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);  compparm[0] = 5;  status = SWdefcomp(swathID, 0, compparm);  status = SWdefdatafield(swathID, "Opacity", "Track,Xtrack", NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);  status = SWdefcomp(swathID, 0, NULL);

```
status = SWdefdatafield(swathID, "Temperature",  
"Track,Xtrack", NULL, H5T_NATIVE_FLOAT, HDFE_AUTOMERGE);
```

Note that the HDFE\_AUTOMERGE parameter will be ignored in the *Temperature* field definition.

**FORTRAN**    integer function swdefcomp(*swathid*, *compcode*, *compparm*)  
              integer        *swathid*  
              integer\*4      *compcode*  
              integer        *compparm*

The equivalent *FORTRAN* code for the example above is:

```
integer*4 HDFE_COMP_NONE  
  
parameter (HDFE_COMP_NONE=0)  
  
integer*4 HDFE_COMP_DEFLATE  
  
parameter (HDFE_COMP_DEFLATE=4)  
  
integer HDFE_NATIVE_FLOAT  
  
parameter (HDFE_NATIVE_FLOAT = 1)  
  
integer compparm(5)  
  
status = swdefcomp(swathid, HDFE_COMP_DEFLATE, compparm)  
  
status = swdefdfld(swathid, "Pressure", "Track,Xtrack",  
H5T_NATIVE_FLOAT, HDFE_NOMERGE)  
  
compparm(1) = 5  
  
status = swdefcomp(swathid, HDFE_COMP_NONE, compparm)  
  
status = swdefdfld(swathid, "Opacity", "Track,Xtrack",  
H5T_NATIVE_FLOAT, HDFE_NOMERGE)
```

# Define a New Data Field Within a Swath

---

## SWdefdatafield

```
herr_t SWdefdatafield(hid_t swathID, char *fieldname, char *dimlist, char  
*maxdimlist, hid_t numbertype, int32_t merge)
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>fieldname</i>	IN: Name of field to be defined
<i>dimlist</i>	IN: The list of data dimensions defining the field
<i>maxdimlist</i>	IN: The list of maximum data dimensions defining the field
<i>numbertype</i>	IN: The number type of the data stored in the field
<i>merge</i>	IN: Merge code (HDFE_NOMERGE (0) - no merge, HDFE_AUTOMERGE (1) -merge)
Purpose	Defines a new data field within the swath.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list.
Description	This routine defines geolocation fields to be stored in the swath. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in C order, that is, the last dimension is incremented first. The API will attempt to merge into a single object those fields that share dimensions and in case of multidimensional fields, numbertype. Two and three dimensional fields will be merged into a single three-dimensional object if the last two dimensions (in C order) are equal. If the merge code for a field is set to HDF_NOMERGE (0), the API will not attempt to merge it with other fields. Because merging breaks the one-to-one correspondence between HDF-EOS fields and HDF SDS arrays, it should not be set if the user wishes to access the HDF-EOS field directly using HDF routines or, for example, to create an HDF attribute corresponding to the field. If maximum dimensions are the same as input dimensions, the "NULL" should be passed as a fourth parameter. In case of unlimited dimension (appendable field) the corresponding dimension should be defined by calling SWdefdim() with the H5S_UNLIMITED as third parameter.
Example	In this example, we define a three dimensional data field named <i>Spectra</i> with dimensions <i>Bands</i> , <i>DataTrack</i> , and <i>DataXtrack</i> :

```
status = SWdefdatafield(swathID, "Spectra",
    "Bands,DataTrack,DataXtrack", NULL, H5T_NATIVE_FLOAT,
    HDFE_AUTOMERGE);
```

Note: To assure that the fields defined by SWdefdatafield are properly established in the file, the swath should be detached (and then reattached) before writing to any fields.

**FORTRAN**

```
integer function swdefdfld(swathid, fieldname, dimlist,
maxdimlist,numbertype,merge)
    integer          swathid
    character*(*)   fieldname
    character*(*)   dimlis
    character*(*)   maxdimlist
    integer          numbertype
    integer          merge
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_NATIVE_FLOAT=1)
parameter (HDFE_AUTOMERGE=1)

status = swdefdfld(swathid, "Spectra",
    "DataXtrack,DataTrack,Bands", "", HDFE_NATIVE_FLOAT,
    HDFE_AUTOMERGE)
```

# Define a New Dimension Within a Swath

---

## SWdefdim

herr\_t SWdefdim(hid\_t *swathID*, char \**dimname*, int32\_t *dim*)

<i>swathID</i>	IN: swath returned by Swcreate or SWattach
<i>fieldname</i>	IN: Name of dimension to be defined
<i>dim</i>	IN: The size of the dimension
Purpose	Defines a new dimension within the swath.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is an improper swath id.
Description	This routine defines dimensions that are used by the field definition routines (described subsequently) to establish the size of the field.
Example	In this example, we define a track geolocation dimension, <i>GeoTrack</i> , of size 2000, a cross track dimension, <i>GeoXtrack</i> , of size 1000 and two corresponding data dimensions with twice the resolution of the geolocation dimensions:

```
status = SWdefdim(swathID, "GeoTrack", 2000);
status = SWdefdim(swathID, "GeoXtrack", 1000);
status = SWdefdim(swathID, "DataTrack", 4000);
status = SWdefdim(swathID, "DataXtrack", 2000);
status = SWdefdim(swathID, "Bands", 5);
```

To specify an unlimited dimension which can be used to define an appendable array, the dimension value should be set to zero or equivalently, H5S\_UNLIMITED:

```
status = SWdefdim(swathID, "Unlim", H5S_UNLIMITED);
integer function swdefdim(swathid,fieldname,dim)
integer      swathid
character(*)  fieldname
integer*4     dim
```

The equivalent *FORTRAN* code for the first example above is:

```
status = swdefdim(swathid, "GeoTrack", 2000)
```

The equivalent *FORTRAN* code for the unlimited dimension example above is:

```
parameter (H5S_UNLIMITED=0)
status = swdefdim(swathid, "Unlim", H5S_UNLIMITED)
```

# Define Mapping Between Geolocation and Data Dimensions

---

## SWdefdimmap

```
herr_t SWdefdimmap(hid_t swathID, char *geodim, char *datadim, int32_t offset,  
                    int32_t increment)
```

<i>swathID</i>	IN:	Swath id returned by SWcreate or SWattach
<i>geodim</i>	IN:	Geolocation dimension name
<i>datadim</i>	IN:	Data dimension name
<i>offset</i>	IN:	The offset of the geolocation dimension with respect to the data dimension
<i>increment</i>	IN:	The increment of the geolocation dimension with respect to the data dimension
Purpose	Defines monotonic mapping between the geolocation and data dimensions.	
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is incorrect geolocation or data dimension name.	
Description	Typically the geolocation and data dimensions are of different size (resolution). This routine established the relation between the two where the offset gives the index of the data element (0-based) corresponding to the first geolocation element and the increment gives the number of data elements to skip for each geolocation element. If the geolocation dimension begins "before" the data dimension, then the offset is negative. Similarly, if the geolocation dimension has higher resolution than the data dimension, then the increment is negative.	
Example	In this example, we establish that (1) the first element of the <i>GeoTrack</i> dimension corresponds to the first element of the <i>DataTrack</i> dimension and the data dimension has twice the resolution as the geolocation dimension, and (2) the first element of the <i>GeoXtrack</i> dimension corresponds to the second element of the <i>DataXtrack</i> dimension and the data dimension has twice the resolution as the geolocation dimension:	

```
status = SWdefdimmap(swathID, "GeoTrack", "DataTrack", 0,  
                     2);  
  
status = SWdefdimmap(swathID, "GeoXtrack", "DataXtrack", 1,  
                     2);
```

**FORTRAN**    integer function  
              swdefmap(*swathid,geodim,datadim,offset,increment*)  
              integer        *swathid*  
              character\*(\*) *geodim*  
              character\*(\*) *datadim*  
              integer\*4      *offset*  
              integer\*4      *increment*

The equivalent *FORTRAN* code for the second example above is:

```
status = swdefmap(swathid, "GeoTrack", "DataTrack", 0, 2)  
status = swdefmap(swathid, "GeoXtrack", "DataXtrack", 1, 2)
```

# Define a New Geolocation Field Within a Swath

---

## SWdefgeofield

```
herr_t SWdefgeofield(hid_t swathID, char *fieldname, char *dimlist, char  
*maxdimlist, hid_t numbertype, int32_t merge)
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>fieldname</i>	IN: Name of field to be defined
<i>dimlist</i>	IN: The list of geolocation dimensions defining the field
<i>maxdimlist</i>	IN: The list of maximum geolocation dimensions defining the field
<i>numbertype</i>	IN: The number type of the data stored in the field
<i>merge</i>	IN: Merge code (HDFE_NOMERGE (0) - no merge, HDFE_AUTOMERGE (1) -merge)
Purpose	Defines a new geolocation field within the swath.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list.
Description	This routine defines geolocation fields to be stored in the swath. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in C order, that is, the last dimension is incremented first. The API will attempt to merge into a single object those fields that share dimensions and in case of multidimensional fields, numbertype. Two and three dimensional fields will be merged into a single three-dimensional object if the last two dimensions (in C order are equal). If the merge code for a field is set to 0, the API will not attempt to merge it with other fields. Fields using the unlimited dimension will not be merged. Because merging breaks the one-to-one correspondence between HDF-EOS fields and HDF SDS arrays, it should not be set if the user wishes to access the HDF-EOS field directly using HDF routines or, for example, to create an HDF attribute corresponding to the field. If maximum dimensions are the same as input dimensions, the "NULL" should be passed as a fourth parameter. In case of unlimited dimension (appendable field) the corresponding dimension should be defined by calling SWdefdim() with the H5S_UNLIMITED as third parameter.

**Example** In this example, we define the geolocation fields, *Longitude* and *Latitude* with dimensions *GeoTrack* and *GeoXtrack* and containing 4 byte floating point numbers. We allow these fields to be merged into a single object:

```
status = SWdefgeofield(swathID, "Longitude",
"GeoTrack,GeoXtrack", NULL,
H5T_NATIVE_FLOAT,HDFE_AUTOMERGE);

status = SWdefgeofield(swathID,"Latitude", "GeoTrack,
GeoXtrack", NULL, H5T_NATIVE_FLOAT, HDFE_AUTOMERGE);
```

Note: To assure that the fields defined by SWdefgeofield are properly established in the file, the swath should be detached (and then reattached) before writing to any fields.

**FORTRAN**

```
integer function swdefgfld(swathid, fieldname, dimlist, maxdimlist,
numbertype, merge)
integer          swathid
character*(*)   fieldname
character*(*)   dimlist
character*(*)   maxdimlist
integer          numbertype
integer          merge
```

The equivalent *FORTRAN* code for the first example above is:

```
parameter (HDFE_NATIVE_FLOAT=1)
parameter (HDFE_AUTOMERGE=1)

status = swdefgfld(swathid, "Longitude",
"GeoXtrack,GeoTrack", "", HDFE_NATIVE_FLOAT,HDFE_AUTOMERGE)
```

The dimensions are entered in *FORTRAN* order with the first dimension incremented first.

# Define Indexed Mapping Between Geolocation and Data Dimension

---

## SWdefidxmap

```
herr_t SWdefidxmap(hid_t swathID, char *geodim, char *datadim, int32_t  
index[]),
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>geodim</i>	IN: Geolocation dimension name
<i>datadim</i>	IN: Data dimension name
<i>index</i>	IN: The array containing the indices of the data dimension to which each geolocation element corresponds.

Purpose      Defines a non-regular mapping between the geolocation and data dimension.

Return value      Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is incorrect geolocation or data dimension name.

Description      If there does not exist a regular (linear) mapping between a geolocation and data dimension, then the mapping must be made explicit. Each element of the index array, whose dimension is given by the geolocation size, contains the element number (0-based) of the corresponding data dimension.

Example      In this example, we consider the (simple) case of a geolocation dimension, *IdxGeo* of size 5 and a data dimension *IdxData* of size 8.

```
int32 index[5] = {0,2,3,6,7};  
  
status = SWdefidxmap(swathID, "IdxGeo", "IdxData", index);
```

In this case the 0th element of *IdxGeo* will correspond to the 0th element of *IdxData*, the 1st element of *IdxGeo* to the 2nd element of *IdxData*, etc.

FORTRAN      integer function

```
swdefimap(swathid, geodim, datadim, index)  
integer        swathid  
character(*)    geodim  
character(*)    datadim  
integer*4       index (*)
```

The equivalent *FORTRAN* code for the example above is:

```
status = swidefmap(swathid, "IdxGeo", "IdxData", index)
```

Note: The *index* array should be 0-based.

# Define a Time Period of Interest

---

## SWdeftimeperiod

```
int32_t SWdeftimeperiod(hid_t swathID, float64 starttime , float64 stoptime  
                      int32_t mode)
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>starttime</i>	IN: Start time of period
<i>stoptime</i>	IN: Stop time of period
<i>mode</i>	IN: Cross Track inclusion mode
Purpose	Defines a time period for a swath.
Return value	Returns the swath period ID if successful or FAIL (-1) otherwise.
Description	This routine defines a time period for a swath. It returns a swath period ID which is used by the <i>SWextractperiod</i> routine to read all the entries of a data field within the time period. A cross track is within a time period if 1) its midpoint is within the time period "box", or 2) either of its endpoints is within the time period "box", or 3) any point of the cross track is within the time period "box", depending on the inclusion mode designated by the user. All elements within an included cross track are considered to be within the time period even though a particular element of the cross track might be outside the time period. The swath structure must have the <i>Time</i> field defined
Example	In this example, we define a time period with a start time of 35232487.2 and a stop time of 36609898.1. We will consider a cross track to be within the time period if either one of the time values at the endpoints of a cross track are within the time period.

```
starttime = 35232487.2;  
stoptime = 36609898.1;  
periodID = SWdeftimeperiod(swathID, starttime, stoptime,  
                           HDFE_ENDPOINT);
```

FORTRAN	integer	function swdeftmeper( <i>swathid</i> , <i>starttime</i> , <i>stoptime</i> , <i>mode</i> )
	integer*4	<i>swathid</i>
	real*8	<i>starttime</i>
	real*8	<i>stoptime</i>
	integer*4	<i>mode</i>

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_ENDPOINT=1)

starttime = 35232487.2

stoptime = 36609898.1

periodID = swdeftmeper(swathID, starttime, stoptime,
HDFE_ENDPOINT)
```

# Define a Vertical Subset Region

---

## SWdefvrtrregion

```
int32_t SWdefvrtrregion(hid_t swathID, int32_t regionID, char *vertObj, float64 range[J])
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>regionID</i>	IN: Region (or period) id from previous subset call
<i>vertObj</i>	IN: Dimension or field to subset by
<i>range</i>	IN: Minimum and maximum range for subset
Purpose	Subsets on a <b>monotonic</b> field or contiguous elements of a dimension.
Return value	Returns the swath region ID if successful or FAIL (-1) otherwise.
Description	Whereas the <i>SWdefboxregion</i> and <i>SWdeftimeperiod</i> routines perform subsetting along the “Track” dimension, this routine allows the user to subset along any dimension. The region is specified by a set of minimum and maximum values and can represent either a dimension index (case 1) or field value range(case 2). In the second case, the field must be one-dimensional and the values must be <b>monotonic</b> (strictly increasing or decreasing) in order that the resulting dimension index range be contiguous. (For the current version of this routine, the second option is restricted to fields with number type: INT16, INT32, FLOAT32, FLOAT64.) This routine may be called after <i>SWdefboxregion</i> or <i>SWdeftimeperiod</i> to provide both geographic or time and “vertical” subsetting . In this case the user provides the id from the previous subset call. (This same id is then returned by the function.) This routine may also be called “stand-alone” by setting the input id to HDFE_NOPREVSUB (-1).

This routine may be called up to eight times with the same region ID. It this way a region can be subsetted along a number of dimensions.

The *SWregioninfo* and *SWextractregion* routines work as before, however because there is no mapping performed between geolocation dimensions and data dimensions the field to be subsetted, (the field specified in the call to *SWregioninfo* and *SWextractregion*) must contain the dimension used explicitly in the call to *SWdefvrtrregion* (case 1) or the dimension of the one-dimensional field (case 2).

**Example** Suppose we have a field called *Pressure* of dimension *Height* (= 10) whose values increase from 100 to 1000. If we desire all the elements with values between 500 and 800, we make the call:

```
range[0] = 500.;  
range[1] = 800.;  
regionID = SWdefvrtrregion(swathID, HDFE_NOPREVSUB,  
"Pressure", range);
```

The routine determines the elements in the *Height* dimension which correspond to the values of the *Pressure* field between 500 and 800.

If we wish to specify the subset as elements 2 through 5 (0 - based) of the *Height* dimension, the call would be:

```
range[0] = 2;  
range[1] = 5;  
regionID = SWdefvrtrregion(swathID, HDFE_NOPREVSUB,  
"DIM:Height", range);
```

The “DIM:” prefix tells the routine that the range corresponds to elements of a dimension rather than values of a field.

In this example, any field to be subsetted must contain the *Height* dimension.

If a previous subset region or period was defined with id, *subsetID*, that we wish to refine further with the vertical subsetting defined above we make the call:

```
regionID = SWdefvrtrregion(swathID, subsetID, "Pressure",  
range);
```

The return value, *regionID* is set equal to *subsetID*. That is, the subset region is modified rather than a new one created.

We can further refine the subset region with another call to the routine:

```
freq[0] = 1540.3;  
freq[1] = 1652.8;  
  
regionID = SWdefvrtrregion(swathID, regionID, "FreqRange",  
freq);
```

**FORTRAN**    integer\*4 function swdefvrtrreg(*swathid*, *regionid*, *vertobj*, *range*)  
              integer          *swathid*  
              integer\*4      *regionid*  
              character(\*)   *vertobj*  
              real\*8        *range*

The equivalent *FORTRAN* code for the examples above is:

```
parameter (HDFE_NOPREVSUB=-1)  
  
range(1) = 500.  
range(2) = 800.  
  
regionid = swdefvrtrreg(swathid, HDFE_NOPREVSUB, "Pressure",  
range)  
  
range(1) = 3          ! Note 1-based element numbers  
range(2) = 6  
  
regionid = swdefvrtrreg(swathid, HDFE_NOPREVSUB,  
"DIM:Height", range)  
  
regionid = swdefvrtrreg(swathid, subsetid, "Pressure", range)  
regionid = swdefvrtrreg(swathid, regionid, "FreqRange", freq)
```

# Detach from a Swath Structure

---

## SWdetach

herr\_t SWdetach(hid\_t *swathID*)

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
Purpose	Detaches from swath interface.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise.
Description	This routine should be run before exiting from the swath file for every swath opened by SWcreate or SWattach.
Example	In this example, we detach the swath structure, <i>ExampleSwath</i> :

```
status = SWdetach(swathID);  
FORTRAN integer function swdetach(swathid)  
integer      swathid
```

The equivalent *FORTRAN* code for the example above is:

```
status = swdetach(swathid)
```

# Retrieve Size of Specified Dimension

---

## SWdiminfo

hsize\_t SWdiminfo(hid\_t *swathID*, char \**dimname*)

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>dimname</i>	IN: Dimension name
Purpose	Retrieve size of specified dimension.
Return value	Size of dimension if successful or FAIL (-1) otherwise. If -1, could signify an improper swath id or dimension name.
Description	This routine retrieves the size of specified dimension.

Example In this example, we retrieve information about the dimension, "GeoTrack":

```
dimsize = SWdiminfo(swathID, "GeoTrack");
```

The return value, *dimsize*, will be equal to 2000.

FORTRAN integer\*4 function swdiminfo(*swathid*,*dimname*)  
integer *swathid*  
character(\*) *dimname*

The equivalent *FORTRAN* code for the example above is:

```
dimsize = swdiminfo(swathid, "GeoTrack")
```

# Duplicate a Region or Period

---

## SWdupregion

int32\_t SWdupregion(int32\_t *regionID*)

<i>regionID</i>	IN: Region or period id returned by SWdefboxregion, SWdeftimeperiod, or SWdefvrtrregion.
Purpose	Duplicates a region.
Return value	Returns new region or period ID if successful or FAIL (-1) otherwise.
Description	This routine copies the information stored in a current region or period to a new region or period and generates a new id. It is usefully when the user wishes to further subset a region (period) in multiple ways.

Example In this example, we first subset a swath with *SWdefboxregion*, duplicate the region creating a new region ID, *regionID2*, and then perform two different vertical subsets of these (identical) geographic subset regions:

```
regionID = SWdefboxregion(swathID, cornerlon, cornerlat,  
                         HDFE_MIDPOINT);  
  
regionID2 = SWdupregion(regionID);  
  
regionID = SWdefvrtrregion(swathID, regionID, "Pressure",  
                           rangePres);  
  
regionID2 = SWdefvrtrregion(swathID, regionID2,  
                           "Temperature", rangeTemp);
```

FORTRAN    integer\*4 swdupreg(*regionid*)  
            integer\*4     *regionid*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_MIDPOINT=0)  
  
regionid = swdefboxreg(swathid, cornerlon, cornerlat,  
                      HDFE_MIDPOINT)  
  
regionid2 = swdupreg(regionid)  
  
regionid = swdefvrtrreg(swathid, regionid, 'Pressure',  
                        rangePres)  
  
regionid2 = swdefvrtrreg(swathid, regionid2, 'Temperature',  
                        rangeTemp)
```

# Read Data from a Defined Time Period

---

## SWextractperiod

herr\_t SWextractperiod(hid\_t *swathID*, hid\_t *periodID*, char \**fieldname*, int32\_t *external\_mode*, void \**buffer*)

<i>periodID</i>	IN: Period id returned by SWdefteimeperiod
<i>fieldname</i>	IN: Field to subset
<i>external_mode</i>	IN: External geolocation mode
<i>buffer</i>	OUT: Data buffer
Purpose	Extracts (reads) from subsetted time period.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise.
Description	This routine reads data into the data buffer from the subsetted time period. Only complete crosstracks are extracted. If the external_mode flag is set to <i>HDFE_EXTERNAL</i> (1) then the geolocation fields and the data field can be in different swaths. If set to <i>HDFE_INTERNAL</i> (0), then these fields must be in the same swath structure.
Example	In this example, we read data within the subsetted time period defined in <i>SWdefteimeperiod</i> from the <i>Spectra</i> field. Both the geolocation fields and the <i>Spectra</i> data field are in the same swath.

```
status = SWextractperiod(SWid, periodID, "Spectra",
                         HDFE_INTERNAL, datbuf);
```

**FORTRAN** integer function swextper(*periodid*, *fieldname*, *external\_mode*, *buffer*)  
integer\*4                   *periodid*  
character\*(\*)              *fieldname*  
integer\*4                   *external\_mode*  
<valid type>              *buffer(\*)*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_INTERNAL=0)

status = swextper(periodid, "Spectra", HDFE_INTERNAL,
                  datbuf)
```

# Read Data from a Geographic Region

---

## SWextractregion

```
herr_t SWextractregion(hid_t swathID, int32_t regionID, char * fieldname,  
int32_t external_mode, void *buffer)
```

*swathID* IN: Swath id returned by SWcreate or SWattach

*regionID* IN: Region id returned by SWdefboxregion

*fieldname* IN: Field to subset

*external\_mode* IN: External geolocation mode

*buffer* OUT: Data buffer

Purpose Extracts (reads) from subsetted region.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.

Description This routine reads data into the data buffer from the subsetted region. Only complete crosstracks are extracted. If the external\_mode flag is set to *HDFE\_EXTERNAL* (1) then the geolocation fields and the data field can be in different swaths. If set to *HDFE\_INTERNAL* (0), then these fields must be in the same swath structure.

Example In this example, we read data within the subsetted region defined in *SWdefboxregion* from the *Spectra* field. Both the geo location fields and the *Spectra* data field are in the same swath.

```
status = SWextractregion(SWid, regionID, "Spectra",  
HDFE_INTERNAL, datbuf);
```

**FORTRAN** integer function swextreg(*swathid*, *regionid*, *fieldname*, *external\_mode*,  
*buffer*)

integer *swathid*

integer\*4 *regionid*

character(\*) *fieldname*

integer\*4 *external\_mode*

<valid type> *buffer*(\*)

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_INTERNAL=0)
```

```
status = swextreg(swathid, regionid, "Spectra", HDFE_INTERNAL,  
datbuf)
```

# Retrieve Information About a Swath Field

---

## SWfieldinfo

```
herr_t SWfieldinfo(hid_t swathID, char *fieldname, int *rank, hsize_t dims[],  
H5T_class_t *numbertype, char *dimlist)
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>fieldname</i>	IN: Fieldname
<i>rank</i>	OUT: Rank of field
<i>dims</i>	OUT: Array containing the dimension sizes of the field
<i>numbertype</i>	OUT: Data type class ID of the field
<i>dimlist</i>	OUT: List of dimensions in field
Purpose	Retrieve information about a specific geolocation or data field in the swath.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. A typical reason for failure is the specified field does not exist.
Description	This routine retrieves information on a specific data field.
Example	In this example, we retrieve information about the <i>Spectra</i> data fields:

```
status = SWfieldinfo(swathID, "Spectra", &rank, dims,  
&numbertype, dimlist)
```

The return parameters will have the following values:

*rank=3, numbertype=1, dims[3]={5,4000,2000} and dimlist="Bands, DataTrack, DataXtrack"*

If one of the dimensions in the field is appendable, then the current value for that dimension will be returned in the *dims* array.

**FORTRAN**    integer function swfldinfo(*swathid, fieldname, rank, dims, numbertype, dimlist*)  
                Integer        *swathid*  
                character\*(\*) *fieldname*  
                integer        *rank*  
                integer\*4        *dims(\*)*  
                integer        *numbertype*  
                character      *dimlist*

The equivalent *FORTRAN* code for the example above is:

```
status = swfldinfo(swathid, "Spectra", rank, dims,  
numbertype, dimlist)
```

The return parameters will have the following values:

*rank=3, numbertype=1, dims[3]={2000,4000,5} and  
dimlist="DataXtrack, DataTrack, Bands"*

Note that the dimensions array and dimension list are in FORTRAN order.

# Get Fill Value for a Specified Field

---

## SWgetfillvalue

herr\_t SWgetfillvalue(hid\_t *swathID*, char \**fieldname*, void \**fillvalue*)

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>fieldname</i>	IN: Fieldname
<i>fillvalue</i>	OUT: Space allocated to store the fill value

Purpose Retrieves fill value for the specified field.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type.

Description It is assumed the number type of the fill value is the same as the field.

Example In this example, we get the fill value for the "Temperature" field:

```
status = SWgetfillvalue(swathID, "Temperature", &tempfill);
```

**FORTRAN** integer function

```
swgetfill(swathid,fieldname,fillvalue)
```

```
integer      swathid
```

```
character*(*)  fieldname
```

```
<valid type>  fillvalue(*)
```

The equivalent *FORTRAN* code for the example above is:

```
status = swgetfill(swathid, "Temperature", tempfill)
```

## Retrieve type of dimension mapping when first dimension is geodim

---

### SWgeomapinfo

herr\_t SWgeomapinfo(hid\_t *swathID*, char \**geodim*)

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>geodim</i>	IN: Dimension name
Purpose	Retrieve type of dimension mapping for a dimension.
Return value	Returns (2) for indexed mapping, (1) for regular mapping, (0) if dimension is not mapped, or FAIL (-1) otherwise.
Description	This routine checks the type of mapping (regular or indexed).
Example	In this example, we retrieve information about the type of mapping between the “IdxGeo” and “IdxData” dimensions, defined by Swdefidxmap.

```
Regmap = SWgeomapinfo(swathID, geodim);
```

We will have regmap = 2 for indexed mapping between the “IdxGeo” and “IdxData” dimensions.

NOTE: If the dimension has been mapped regular and indexed, the function will return a value of 3.

**FORTRAN**      integer function swgmapinfo(*swathid*,*geodim*)  
                  integer       *swathid*  
                 character\*(\*) *geodim*

The equivalent *FORTRAN* code for the example above is:

```
status = swgmapinfo(swathid, geodim)
```

# Retrieve Indexed Geolocation Mapping

---

## SWidxmapinfo

```
int32_t SWidxmapinfo(hid_t swathID, char *geodim, char *datadim, int32_t  
                    index[])
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>geodim</i>	IN: Indexed Geolocation dimension name
<i>datadim</i>	IN: Indexed Data dimension name
<i>index</i>	OUT: Mapping offset
Purpose	Retrieve indexed array of specified geolocation mapping.
Return value	Returns size of indexed array if successful or FAIL (-1) otherwise. A typical reason for failure is the specified mapping does not exist.
Description	This routine retrieves the size of the indexed array and the array of indexed elements of the specified geolocation mapping.
Example	In this example, we retrieve information about the indexed mapping between the "IdxGeo" and "IdxData" dimensions:

```
idxsz = SWidxmapinfo(swathID, "IdxGeo", "IdxData", index);
```

The variable, *idxsz*, will be equal to 5 and *index[5]* = {0,2,3,6,7}.

**FORTRAN**

```
integer*4 function swimapinfo(swathid, geodim, datadim, index)
integer          swathid
character(*)    geodim
character(*)    datadim
integer*4        index(*)
```

The equivalent *FORTRAN* code for the example above is:

```
status = swimapinfo(swathid, "IdxGeo", "IdxData", index)
```

# Retrieve Information Swath Attributes

---

## SWinqattrs

int32\_t SWinqattrs(hid\_t *swathID*, char \**attrlist*, int32\_t \**strbufsize*)

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>attrlist</i>	OUT: Attribute list (entries separated by commas)
<i>strbufsize</i>	OUT: String length of attribute list
Purpose	Retrieve information about attributes defined in swath.
Return value	Number of attributes found if successful or FAIL (-1) otherwise.
Description	The attribute list is returned as a string with each attribute name separated by commas. If <i>attrlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator.

Example      In this example, we retrieve information about the attributes defined in a swath structure. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = SWinqattrs(swathID, NULL, &strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 14.

```
nattr = SWinqattrs(swathID, attrlist, &strbufsize);
```

The variable, *attrlist*, will be set to:

"*attrOne,attr\_2*".

FORTRAN      integer\*4 function swinqattrs(*swathid,attrlist,strbufsize*)  
                integer       *swathid*  
                character\*(\*) *attrlist*  
                integer\*4     *strbufsize*

The equivalent *FORTRAN* code for the example above is:

```
nattr = swinqattrs(swathid, attrlist, strbufsize)
```

# Retrieve Information About Data Fields Defined in Swath

---

## SWinqdatafields

```
int32_t SWinqdatafields(hid_t swathID, char *fieldlist, int32_t rank[],
                        H5T_class_t numbertype[])
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>fieldlist</i>	OUT: Listing of data fields (entries separated by commas)
<i>rank</i>	OUT: Array containing the rank of each data field
<i>numbertype</i>	OUT: Array containing the data type class ID for each data field
Purpose	Retrieve information about all of the data fields defined in swath.
Return value	Number of data fields found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id.
Description	The field list is returned as a string with each data field separated by commas. The <i>rank</i> and <i>numbertype</i> arrays will have an entry for each field. Output parameters set to <i>NULL</i> will not be returned.
Example	In this example we retrieve information about the data fields:

```
nflds = SWinqdatafields(swathID, fieldlist, rank,
                         numbertype);
```

The parameter, *fieldlist*, will have the value:

"Spectra" with *ndim* = 1, *rank*[1]={3}, *numbertype*[1]={1}

**FORTRAN** integer\*4 function swinqdfls(*swathid*, *fieldlist*, *rank*, *numbertype*)  
integer *swathid*  
character(\*) *fieldlist*  
integer\*4 *rank*(\*)  
integer *numbertype*(\*)

The equivalent *FORTRAN* code for the example above is:

```
nflds = swinqdfls(swathid, fieldlist, rank, numbertype)
```

# Retrieve Information About Data Type of a Data Field Defined in Swath

---

## **SWinqdatatype**

```
hid_t SWinqdatatype(hid_t swathID, char *fieldname, intn fieldgroup, hid_t  
                  *datatype, H5T_class_t *classid, H5T_order_t *order, size_t  
                  *size)
```

<i>swathID</i>	IN:	Swath id returned by SWcreate or SWattach
<i>fieldname</i>	IN::	String containing the name of a field
<i>fieldgroup</i>	IN:	Group (Data Field /Geolocation Field ) the data field belongs to
<i>datatype</i>	OUT:	Data type ID of a data field dataset
<i>classid</i>	OUT:	Data type class ID
<i>order</i>	OUT:	Byte order of an atomic datatype
<i>size</i>	OUT:	Size of a datatype (in bytes)
Purpose	Retrieve information about the data type of a specified data field.	
Return value	Status variable (0 if successful or -1 otherwise).	
Description		
Example		

FORTRAN

# Retrieve Information About Dimensions Defined in Swath

---

## SWinqdims

int32\_t SWinqdims(hid\_t *swathID*, char \**dimname*, int32\_t *dims*[])

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>dimname</i>	OUT: Dimension list (entries separated by commas)
<i>dims</i>	OUT: Array containing size of each dimension
Purpose	Retrieve information about all of the dimensions defined in swath.
Return value	Number of dimension entries found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id.
Description	The dimension list is returned as a string with each dimension name separated by commas. Output parameters set to <i>NULL</i> will not be returned.
Example	In this example, we retrieve information about the dimensions defined in the <i>ExampleSwath</i> structure:

ndims = SWinqdims(swathID, dimname, dims);

The parameter, *dimname*, will have the value:

"GeoTrack,GeoXtrack,DataTrack,DataXtrack,Bands,Unlim"

with *ndims* = 6, *dims*[6]={2000,1000,4000,2000,5,0}

**FORTRAN**

```
integer*4 function swinqdims(swathid,dimname,dims)
integer          swathid
character*(*)   dimname
integer*4        dims(*)
```

The equivalent *FORTRAN* code for the example above is:

```
ndims = swingdims(swathid, dimname, dims)
```

# Retrieve Information About Geolocation Fields Defined in Swath

---

## SWinqgeofields

```
int32_t SWinqgeofields(hid_t swathID, char *fieldlist, int32_t rank[],  
                      H5T_class_t numbertype[])
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>fieldlist</i>	OUT: Listing of geolocation fields (entries separated by commas)
<i>rank</i>	OUT: Array containing the rank of each geolocation field
<i>numbertype</i>	OUT: Array containing the data type class ID for each geolocation field
Purpose	Retrieve information about all of the geolocation fields defined in swath.
Return value	Number of geolocation fields found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id.
Description	The field list is returned as a string with each geolocation field separated by commas. The <i>rank</i> and <i>numbertype</i> arrays will have an entry for each field. Output parameters set to <i>NULL</i> will not be returned.
Example	In this example, we retrieve information about the geolocation fields:

```
nflds = SWinqgeofields(swathID, fieldlist, rank,  
                      numbertype);
```

The parameter, *fieldlist*, will have the value: "Longitude,Latitude" with *nflds* = 2, *rank*[2]={2,2}, *numbertype*[2]={1,1}

**FORTRAN**      integer\*4 function swinqgflds(swathid, fieldlist, rank, numbertype)  
                  integer          *swathid*  
                  character\*(\*) *fieldlist*  
                  integer\*4     *rank*(\*)  
                  integer         *numbertype*(\*)

The equivalent *FORTRAN* code for the example above is:

```
nflds = swinqgflds(swathid, fieldlist, rank, numbertype)
```

# Retrieve Information About Indexed Mappings Defined in Swath

---

## SWinqidxmaps

int32\_t SWinqidxmaps(hid\_t swathID, char \*idxmap, int32\_t idxsizes[])

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>idxmap</i>	OUT: Indexed Dimension mapping list (entries separated by commas)
<i>idxsizes</i>	OUT: Array containing the sizes of the corresponding index arrays.
Purpose	Retrieve information about all of the indexed geolocation/data mappings defined in swath.
Return value	Number of indexed mapping relations found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id.
Description	The dimension mapping list is returned as a string with each mapping separated by commas. The two dimensions in each mapping are separated by a slash (/). Output parameters set to <i>NULL</i> , will not be returned.
Example	In this example, we retrieve information about the indexed dimension mappings:  nidxmaps = SWinqidxmaps(swathID, idxmap, idxsizes);  The variable, <i>idxmap</i> , will contain the string:  "IdxGeo/IdxData" with nidxmaps = 1 and idxsizes[1]={5}.
FORTRAN	integer*4 function  swinqimaps(swathid,dimmap,idxsizes)  integer <i>swathid</i>  character(*) <i>dimmap</i>  integer*4 <i>idxsizes(*)</i>

The equivalent *FORTRAN* code for the example above is:

```
nidxmaps = swingimaps(swathid, dimmap, idxsizes)
```

# Retrieve Information About Dimension Mappings Defined in Swath

---

## SWinqmaps

```
int32_t SWinqmaps(hid_t swathID, char *dimmap, int32_t offset[], int32_t increment[])
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>dimmap</i>	OUT: Dimension mapping list (entries separated by commas)
<i>offset</i>	OUT: Array containing the offset of each geolocation relation
<i>increment</i>	OUT: Array containing the increment of each geolocation relation
Purpose	Retrieve information about all of the (non-indexed) geolocation relations defined in swath.
Return value	Number of geolocation relation entries found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id.
Description	The dimension mapping list is returned as a string with each mapping separated by commas. The two dimensions in each mapping are separated by a slash (/). Output parameters set to <i>NULL</i> will not be returned.
Example	In this example, we retrieve information about the dimension mappings in the <i>ExampleSwath</i> structure:

```
nmaps = SWinqmaps(swathID, dimmap, offset, increment);
```

The variable, *dimmap*, will contain the string:

"GeoTrack/DataTrack,GeoXtrack/DataXtrack" with *nmaps* = 2, *offset[2]*={0,1} and *increment[2]*={2,2}.

**FORTRAN** integer\*4 function  
  
swinqmaps(*swathid*,*dimmap*,*offset*,*increment*)  
integer *swathid*  
character(\*) *dimmap*  
integer\*4 *offset(\*)*  
integer\*4 *increment(\*)*

The equivalent *FORTRAN* code for the example above is:

```
nmaps = swingmaps(swathid, dimmap, offset, increment)
```

# Retrieve Swath Structures Defined in HDF-EOS File

---

## SWinqswath

int32\_t SWinqswath(char \*filename, char \*swathlist, int32\_t \*strbufsize)

<i>filename</i>	IN: HDF-EOS filename
<i>swathlist</i>	OUT: Swath list (entries separated by commas)
<i>strbufsize</i>	OUT: String length of swath list
Purpose	Retrieves number and names of swaths defined in HDF-EOS file.
Return value	Number of swaths found if successful or FAIL (-1) otherwise.
Description	The swath list is returned as a string with each swath name separated by commas. If <i>swathlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . If <i>strbufsize</i> is also set to NULL, the routine returns just the number of swaths. Note that <i>strbufsize</i> does not count the null string terminator.

Example      In this example, we retrieve information about the swaths defined in an HDF-EOS file, *HDFEOS.h5*. We assume that there are two swaths stored, *SwathOne* and *Swath\_2*:

```
nswath = SWinqswath("HDFEOS.h5", NULL, strbufsize);
```

The parameter, *nswath*, will have the value 2 and *strbufsize* will have value 16.

```
nswath = SWinqswath("HDFEOS.h5", swathlist, strbufsize);
```

The variable, *swathlist*, will be set to:

“*SwathOne,Swath\_2*”.

FORTRAN      integer\*4 function swingswath(*filename,swathlist,strbufsize*)  
                 character\*(\*) *filename*  
                 character\*(\*) *swathlist*  
                 integer\*4     *strbufsize*

The equivalent *FORTRAN* code for the example above is:

```
nswath = swingswath('HDFEOS.h5', swathlist, strbufsize)
```

# Retrieve Offset and Increment of Specific Dimension Mapping

---

## SWmapinfo

```
herr_t SWmapinfo(hid_t swathID, char *geodim, char *datadim, int32_t *offset,  
                  int32_t *increment)
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>geodim</i>	IN: Geolocation dimension name
<i>datadim</i>	IN: Data dimension name
<i>offset</i>	OUT: Mapping offset
<i>increment</i>	OUT: Mapping increment
Purpose	Retrieve offset and increment of specific monotonic geolocation mapping.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. A typical reason for failure is the specified mapping does not exist.
Description	This routine retrieves offset and increment of the specified geolocation mapping.
Example	In this example, we retrieve information about the mapping between the <i>GeoTrack</i> and <i>DataTrack</i> dimensions:

```
status = SWmapinfo(swathID, "GeoTrack", "DataTrack",  
                   &offset, &increment);
```

The variable *offset* will be 0 and *increment* 2.

**FORTRAN**

```
integer function swmapinfo(swathid, geodim, datadim, offset, increment)  
integer      swathid  
character*(*) geodim  
character*(*) datadim  
integer*4     offset  
integer*4     increment
```

The equivalent *FORTRAN* code for the example above is:

```
status = swmapinfo(swathid, "GeoTrack", "DataTrack",  
                   offset, increment)
```

# Return Number of Specified Objects in a Swath

---

## SWnentries

`int32_t SWnentries(hid_t swathID, int32_t entrycode, int32_t *strbufsize)`

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>entrycode</i>	IN: Entrycode
<i>strbufsize</i>	OUT: String buffer size
Purpose	Returns number of entries and descriptive string buffer size for a specified entity.

Return value Number of entries if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id or entry code.

Description This routine can be called before an inquiry routines in order to determine the sizes of the output arrays and descriptive strings. The string length does not include the NULL terminator.

The entry codes are:

- HDFE\_NENTDIM (0) - Dimensions
- HDFE\_NENTMAP (1) - Dimension Mappings
- HDFE\_NENTIMAP (2) - Indexed Dimension Mappings
- HDFE\_NENTGFLD (3) - Geolocation Fields
- HDFE\_NENTDFLD (4) - Data Fields

Example In this example, we determine the number of dimension mapping entries and the size of the map list string.

```
nmaps = SWnentries(swathID, HDFE_NENTMAP, &bufsz);
```

The return value, *nmaps*, will be equal to 2 and *bufsz* = 39

**FORTRAN** `integer*4 function swnentries(swathid, entrycode, bufsize)`  
`integer swathid`  
`integer*4 entrycode`  
`integer*4 bufsize`

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_NENTMAP=1)  
nmaps = swnentries(swathid, HDFE_NENTMAP, bufsize)
```

# Open HDF-EOS File

---

## SWopen

hid\_t SWopen(char \*filename, **unsigned** flags)

<i>filename</i>	IN: Complete path and filename for the file to be opened
<i>flags</i>	IN: <i>File access flags</i>
Purpose	Opens or creates HDF file in order to create, read, or write a swath.
Return value	Returns the swath file id handle (fid) if successful or FAIL (-1) otherwise.
Description	This routine creates a new file or opens an existing one, depending on the access flag.  Access flags:  H5F_ACC_RDONLY Open for read only. If file does not exist, error H5F_ACC_RDWR Open for read/write. If file does not exist, error H5F_ACC_TRUNC If file exist, truncate it, then open a new file for read/write  H5F_ACC_EXCL Fail if file already exists H5F_ACC_DEBUG Print debug information H5F_ACC_DEFAULT Apply default file access and creation properties

Example In this example, we create a new swath file named, *SwathFile.h5*. It returns the file handle, *fid*.

```
fid = SWopen("SwathFile.h5", H5F_ACC_TRUNC);
```

**FORTRAN** integer function swopen(*filename,flags*)  
character\*(\*) *filename*  
integer *flags*

The access codes should be defined as parameters:

```
parameter (H5F_ACC_TRUNC = 2)
```

The equivalent *FORTRAN* code for the example above is:

```
fid = swopen("SwathFile.h5", H5F_ACC_TRUNC)
```

# Return Information About a Defined Time Period

---

## SWperiodinfo

```
herr_t SWperiodinfo(hid_t swathID, int32_t periodID, char * fieldname,  
H5T_class_t *ntype, int32_t *rank, hsize_t dims[], int32_t *size)
```

<i>swathID</i>	IN:	Swath id returned by SWcreate or SWattach
<i>periodID</i>	IN:	Period id returned by SWdeftimeperiod
<i>fieldname</i>	IN:	Field to subset
<i>ntype</i>	OUT:	Data type class ID of field
<i>rank</i>	OUT:	Rank of field
<i>dims</i>	OUT:	Dimensions of subset period
<i>size</i>	OUT:	Size in bytes of subset period
Purpose	Retrieves information about the subsetted period.	
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise.	
Description	This routine returns information about a subsetted time period for a particular field. It is useful when allocating space for a data buffer for the subset. Because of differences in number type and geolocation mapping, a given time period will give different values for the dimensions and size for various fields.	
Example	In this example, we retrieve information about the time period defined in <i>SWdeftimeperiod</i> for the <i>Spectra</i> field. We use this to allocate space for data in the subsetted time period.	

```
/* Get size in bytes of time period for "Spectra" field*/  
  
status = SWperiodinfo(SWid, periodID, "Spectra", &ntype,  
                      &rank, dims, &size);  
  
/* Allocate space */  
  
datbuf = (float64 *) malloc(size);
```

**FORTRAN**    integer function swperinfo(*swathid*, *periodid*, *fieldname*, *ntype*, *rank*, *dims*,  
              *size*)

              integer                    *swathid*  
              integer\*4                *periodid*  
              character\*(\*)          *fieldname*  
              integer                    *ntype*  
              integer\*4                *rank*  
              integer\*4                *dims(\*)*  
              integer\*4                *size*

The equivalent *FORTRAN* code for the example above is:

```
status = swperinfo(swid, periodid, "Spectra", ntype, rank, dims, size)
```

# Close Swath Ragged Array

---

## SWraclose

herr\_t SWraclose(hid\_t *swathID*, char \**raname*)

*swathID* IN: Swath id returned by SWcreate or SWattach

*raname* IN: Ragged array name

Purpose Closes the ragged array

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or incorrect ragged array name.

Example In this example, we close the ragged array with the name "RA\_1":

```
status = Swraclose(SwathID,"RA_1");
```

**FORTRAN** integer function swraclose(*swathid,raname*)

```
integer swathid
```

```
character*(*) raname
```

The equivalent *FORTRAN* code for the example above is:

```
status = swraclose(swathid, "RA_1")
```

# Define Swath Ragged Array

---

## SWradefine

```
herr_t SWradefine(hid_t swathID, const char *raname, hsize_t chunk_size[],  
                   hid_t numbertype)
```

*swathID* IN: Swath id returned by SWcreate or SWattach

*raname* IN: Ragged array name

*chunk\_size* IN: Array containing the size of each chunk:

*chunk\_size[0]* - length

*chunk\_size[1]* - width

*numbertype* IN: Data type of ragged array data

Purpose Define ragged array in a swath

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or incorrect ragged array name.

Example In this example, we define the ragged array with the name "RA\_1":

```
status = SWradefine(SwathID,"RA_1",chunk_size, H5T_NATIVE_INT);
```

**FORTRAN** integer function swradefine(*swathid*,*raname*,*chunksz*, )

    integer *swathid*

    character(\*) *raname*

    integer\*4 *chunksz*(2)

The equivalent *FORTRAN* code for the example above is:

```
status = swradefine(swathid, "RA_1", chunksz, )
```

# Open Swath Ragged Array

---

## SWraopen

hid\_t SWraopen(hid\_t *swathID*, const char \**raname*)

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>raname</i>	IN: Ragged array name
Purpose	Open ragged array in a swath
Return value	Returns ragged array ID if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or ragged array name

Example In this example, we open the ragged array with the name "RA\_1":

```
raID = SWraopen(SwathID,"RA_1");
```

FORTRAN integer function swraopen(*swathid,raname*)  
integer *swathid*  
character(\*) *raname*

The equivalent *FORTRAN* code for the example above is:

```
raid = swraopen(swathid, "RA_1")
```

# Read Rows from the Swath Ragged Array

---

## SWraread

herr_t SWraread(hid_t <i>swathID</i> , char <i>raname</i> , hsize_t <i>start_row</i> , hsize_t <i>nrows</i> , hid_t <i>numbertype</i> , hsize_t <i>size[]</i> , void ** <i>buf</i> )	
<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>raname</i>	IN: Ragged array name
<i>start_row</i>	IN: Row at which the reading will start
<i>nrows</i>	IN: number of rows to read out
<i>numbertype</i>	IN: Data type of ragged array data
<i>size[]</i>	IN: Array (of <i>nrows</i> elements) containing lengths of rows to read out
<i>buf[]</i>	OUT: Pointers to buffers containing the data
Purpose	Read out data from the ragged array in a swath
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or incorrect ragged array name.
Example	In this example, we read two rows (starting from row 3) from the ragged array with the name "RA_1": <pre>int32_t *buf[2]; for (i =0; i&lt; 2; i++){     buf[i] = (int32_t *)calloc(10+i, sizeof(int32_t *)); } size[0] = 10; size[1] = 11; status = SWraread(SwathID,"RA_1",3,2 H5T_NATIVE_INT, size,(void **)buf);</pre>
FORTRAN	integer function swraread( <i>swathid</i> , <i>raname</i> , <i>start</i> , <i>nrows</i> , <i>ntype</i> , <i>size</i> , <i>buf</i> )     integer <i>swathid</i> character(*) <i>raname</i> integer*4

The equivalent *FORTRAN* code for the example above is:

```
status = swraread(swathid, "RA_1", 3, 2, ,size,buf)
```

# Write Rows to the Swath Ragged Array

---

## SWrawrite

herr_t SWrawrite(hid_t <i>swathID</i> , char <i>raname</i> , hsize_t <i>start_row</i> , hsize_t <i>nrows</i> , hid_t <i>numbertype</i> , hsize_t <i>size[]</i> , void ** <i>buf</i> )	
<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>raname</i>	IN: Ragged array name
<i>start_row</i>	IN: Row at which the writing will start
<i>nrows</i>	IN: number of rows to write to
<i>numbertype</i>	IN: Data type of ragged array data
<i>size[]</i>	IN: Array (of <i>nrows</i> elements) containing lengths of rows to write to
<i>buf[]</i>	IN: Pointers to buffers containing the data
Purpose	Write data to the ragged array in a swath
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or incorrect ragged array name.
Example	In this example, we write two rows (starting from row 3) to the ragged array "RA_1":  size[0] = 10;  size[1] = 11;  status = SWrawrite(SwathID,"RA_1",3,2 H5T_NATIVE_INT, size,(void **)buf);
FORTTRAN	integer function swrawrite( <i>swathid</i> , <i>raname</i> , <i>start</i> , <i>nrows</i> , <i>ntype</i> , <i>size</i> , <i>buf</i> )  integer <i>swathid</i> character(*) <i>raname</i> integer*4 <i>size(*)</i> <valid type> <i>buf(*)</i>

The equivalent *FORTTRAN* code for the example above is:

Parameter(HDFE\_NATIVE\_INT=0)

status = swrawrite(swathid, "RA\_1",3,2, HDFE\_NATIVE\_INT,size,buf)

# Read Swath Attribute

---

## SWreadattr

herr\_t SWreadattr(hid\_t *swathID*, char \**attrname*, void \**dbuf*)

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>attrname</i>	IN: Attribute name
<i>dbuf</i>	OUT: Buffer allocated to hold attribute values
Purpose	Reads attribute from a swath.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type or incorrect attribute name.
Description	The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.
Example	In this example, we read a single precision (32 bit) floating point attribute with the name "ScalarFloat":

```
status = SWreadattr(swathID, "ScalarFloat", &f32);  
FORTRAN integer function swrdattr(swathid,attrname,dbuf)  
integer           swathid  
character*(*) attrname  
<valid type> dbuf(*)
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_NATIVE_FLOAT=1)  
status = swrdattr(swathid, "ScalarFloat", f32)
```

# Read Data From a Swath Field

---

## SWreadfield

herr\_t SWreadfield(hid\_t *swathID*, char \**fieldname*, const hsize\_t *start*[], const hsize\_t *stride*[], const hsize\_t *edge*[], void \**buffer*)

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>fieldname</i>	IN: Name of field to read
<i>start</i>	IN: Array specifying the starting location within each dimension
<i>stride</i>	IN: Array specifying the number of values to skip along each dimension
<i>edge</i>	IN: Array specifying the number of values to read along each dimension
<i>buffer</i>	OUT: Buffer to store the data read from the field
Purpose	Reads data from a swath field.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are improper swath id or unknown fieldname.
Description	The values within <i>start</i> , <i>stride</i> , and <i>edge</i> arrays refer to the swath field (input) dimensions. The output data in <i>buffer</i> is written to contiguously. The default value for <i>stride</i> is 1 and is used if this parameter is set to <i>NULL</i> .
Example	In this example, we read data from the 10th track (0-based) of the <i>Longitude</i> field.

```
float32 track[1000];

hsize_t start[2]={10,1};

hsize_t edge[2]={1,1000};

status = SWreadfield(swathID, "Longitude", start, NULL,
                     edge, track);
```

```

FORTRAN integer function

    swrdfld(swathid, fieldname, start, stride, edge,buffer)
        integer      swathid
        character*(*) fieldname
        integer*4     start(*)
        integer*4     stride(*)
        integer*4     edge(*)
        <valid type> buffer(*)

```

The *start*, *stride*, and *edge* arrays must be defined explicitly, with the *start* array being 0-based.

The equivalent *FORTRAN* code for the example above is:

```

real*4 track(1000)

integer*4 start(2), stride(2), edge(2)

start(1) = 0
start(2) = 10
stride(1) = 1
stride(2) = 1
edge(1) = 1000
edge(2) = 1

status = swrdfld(swathid, "Longitude", start, stride, edge,
                  track);

```

# Define a Longitude-Latitude Box Region for a Swath

---

## SWregionindex

```
int32_t SWregionindex(hid_t swathID, float64 cornerlon[], float64 cornerlat[],  
                      int32_t mode, char *geodim, int32_t idxrange[])
```

<i>swathID</i>	IN:	Swath id returned by SWcreate or SWattach
<i>cornerlon</i>	IN:	Longitude in decimal degrees of box corners
<i>cornerlat</i>	IN:	Latitude in decimal degrees of box corners
<i>mode</i>	IN:	Cross Track inclusion mode
<i>geodim</i>	OUT:	Geolocation track dimension
<i>idxrange</i>	OUT:	The indices of the region in the geolocation track dimension.
Purpose		Defines a longitude-latitude box region for a swath.
Return value		Returns the swath region ID if successful or FAIL (-1) otherwise.
Description		The difference between this routine and SWdefboxregion is the geolocation track dimension name and the range of that dimension are returned in addition to a regionID. Other than that difference they are the same function and this function is used just like SWdefboxregion. This routine defines a longitude-latitude box region for a swath. It returns a swath region ID which is used by the <i>SWextractregion</i> routine to read all the entries of a data field within the region. A cross track is within a region if 1) its midpoint is within the longitude-latitude "box" (HDFE_MIDPOINT), or 2) either of its endpoints is within the longitude-latitude "box" (HDFE_ENDPOINT), or 3) any point of the cross track is within the longitude-latitude "box" (HDFE_ANYPOINT), depending on the inclusion mode designated by the user. All elements within an included cross track are considered to be within the region even though a particular element of the cross track might be outside the region. The swath structure must have both <i>Longitude</i> and <i>Latitude</i> (or <i>Colatitude</i> ) fields defined
Example		In this example, we define a region bounded by the 3 degrees longitude, 5 degrees latitude and 7 degrees longitude, 12 degrees latitude. We will consider a cross track to be within the region if its midpoint is within the region.  cornerlon[0] = 3.; cornerlat[0] = 5.; cornerlon[1] = 7.;

```

cornerlat[1] = 12.;

regionID = SWregionindex(swathID, cornerlon, cornerlat,
                         HDFE_MIDPOINT, geodim, idxrange);

FORTRAN integer*4 function swregidx(swathid, cornerlon, cornerlat, mode, geodim,
                                    idxrange)

```

integer        *swathid*  
               real\*8        *cornerlon*  
               real\*8        *cornerlat*  
               integer\*4     *mode*  
               character\*(\*) *geodim*  
               integer\*4     *idxrange*(\*)

The equivalent *FORTRAN* code for the example above is:

```

parameter (HDFE_MIDPOINT=0)

cornerlon(1) = 3.

cornerlat(1) = 5.

cornerlon(2) = 7.

cornerlat(2) = 12.

regionid = swregidx(swathid, cornerlon, cornerlat,
                     HDFE_MIDPOINT, geodim, idxrange)

```

# Return Information About a Defined Region

---

## SWregioninfo

```
herr_t SWregioninfo(hid_t swathID, int32_t regionID, char * fieldname,  
H5T_class_t *ntype, int32_t *rank, hsize_t dims[], int32_t *size)
```

<i>swathID</i>	IN:	Swath id returned by SWcreate or SWattach
<i>regionID</i>	IN:	Region id returned by SWdefboxregion
<i>fieldname</i>	IN:	Field to subset
<i>ntype</i>	OUT:	Data type class ID of field
<i>rank</i>	OUT:	Rank of field
<i>dims</i>	OUT:	Dimensions of subset region
<i>size</i>	OUT:	Size in bytes of subset region
Purpose	Retrieves information about the subsetted region.	
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise.	
Description	This routine returns information about a subsetted region for a particular field. It is useful when allocating space for a data buffer for the region. Because of differences in number type and geolocation mapping, a given region will give different values for the dimensions and size for various fields.	
Example	In this example, we retrieve information about the region defined in <i>SWdefboxregion</i> for the <i>Spectra</i> field. We use this to allocate space for data in the subsetted region.	

```
/* Get size in bytes of region for "Spectra" field*/  
  
status = SWregioninfo(SWid, regionID, "Spectra", &ntype,  
                      &rank, dims, &size);  
  
/* Allocate space */  
  
datbuf = (float64 *) malloc(size);
```

**FORTRAN**    integer function swreginfo(*swathid*, *regionid*, *fieldname*, *ntype*, *rank*, *dims*,  
                  *size*)

                integer                       *swathid*  
                integer\*4                      *regionid*  
                character\*(\*)              *fieldname*  
                integer                      *ntype*  
                integer\*4                     *rank*  
                integer\*4                     *dims(\*)*  
                integer\*4                     *size*

The equivalent *FORTRAN* code for the example above is:

```
status = swreginfo(swid, regionid, "Spectra", ntype, rank,  
                  dims, size)
```

# Set Fill Value for a Specified Field

---

## SWsetfillvalue

herr\_t SWsetfillvalue(hid\_t *swathID*, char \**fieldname*, void \**fillvalue*)

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>fieldname</i>	IN: Fieldname
<i>fillvalue</i>	IN: Pointer to the fill value to be used
Purpose	Sets fill value for the specified field.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type.
Description	The fill value is placed in all elements of the field which have not been explicitly defined. The field must have 2 or more dimensions.
Example	In this example, we set a fill value for the "Temperature" field:

```
tempfill = -999.0;  
  
status = SWsetfillvalue(swathID, "Temperature", &tempfill);
```

**FORTRAN** integer function  
  
swsetfill(*swathid*,*fieldname*,*fillvalue*)  
integer *swathid*  
character(\*) *fieldname*  
<valid type> *fillvalue*(\*)

The equivalent *FORTRAN* code for the example above is:

```
status = swsetfill(swathid, "Temperature", -999.0)
```

# Update map index for a specified region

---

## **SWupdateidxmap**

```
int32_t SWupdateidxmap(hid_t swathID, int32_t regionID, int32_t indexin[],  
                      int32_t indexout[], int32_t indices[])
```

<i>swathID</i>	IN: Swath id returned by SWcreate or Swattach.
<i>regionID</i>	IN: Region id returned by Swdefboxregion.
<i>indexin</i>	IN: The array containing the indices of the data dimension to which each geolocation element corresponds.
<i>indexout</i>	OUT: The array containing the indices of the data dimension to which each geolocation corresponds in the subsetted region. The indexout set to NULL, will not be returned.
<i>indices</i>	OUT: The array containing the indices for start and stop of region.
Purpose	Retrieve indexed array of specified geolocation mapping for a specified region.
Return value	Returns size of updated indexed array if successful or FAIL (-1) otherwise. A typical reason for failure is the specified mapping does not exist.
Description	Theis routine retrieves the size of he indexed array and the array of indexed elements of the specified geolocation mapping for the specified region.
Example	<p>In this example, we retrieve information about the indexed mapping between the “IdxGeo” and “IdxData” dimensions, defined by Swdefboxregion:</p> <pre>/* Get size of index_region array */ idxsz = SWupdateidxmap(swathID, regionID, index, NULL, indices); /* Allocate memory for index_region */ index_region = (int32*)malloc(sizeof(int32) * idxsz); /* Get the array index_region */ idxsz = Swupdateidxmap(swathID, regionID, index, index_region,                       indices);</pre>
FORTRAN	integer*4 function swupimap(swathid, regionid, indexin, indexout) integer swathid integer*4 regionid

```
integer*4 indexin(*)  
integer*4 indexout(*)  
integer*4 indices(2)
```

The equivalent FORTRAN code for the example above is:  
status = swupdateidxmap(swathid, regionid, index, index\_region, indices)

Note: The indexed arrays should be 0-based.

# Write/Update Swath Attribute

---

## SWwriteattr

```
herr_t SWwriteattr(hid_t swathID, char *attrname, hid_t ntype, hsize_t count[],  
void *datbuf)
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>attrname</i>	IN: Attribute name
<i>ntype</i>	IN: Data type of attribute
<i>count</i>	IN: Number of values to store in attribute
<i>datbuf</i>	IN: Attribute values
Purpose	Writes/Updates attribute in a swath.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type.
Description	If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call.
Example	In this example, we write a single precision (32 bit) floating point number with the name "ScalarFloat" and the value 3.14:

```
cnt[0] = 1;  
  
f32 = 3.14;  
  
status = SWwriteattr(swathid, "ScalarFloat",  
H5T_NATIVE_FLOAT, cnt, &f32);
```

We can update this value by simply calling the routine again with the new value:

```
f32 = 3.14159;  
  
status = SWwriteattr(swathid, "ScalarFloat",  
H5T_NATIVE_FLOAT, cnt, &f32);
```

```
FORTRAN integer function swwrattr(swathid, attrname, ntype, count, datbuf)
integer      swathid
character*(*) attrname
integer       ntype
integer*4     count(*)
<valid type> datbuf(*)
```

The equivalent *FORTRAN* code for the first example above is:

```
parameter (DFNT_FLOAT32=5)
f32 = 3.14
status = swwrattr(swathid, "ScalarFloat", DFNT_FLOAT32, 1,
f32)
```

# Write Data to a Swath Field

---

## SWwritefield

```
herr_t SWwritefield(hid_t swathID, char *fieldname, const hsize_t start[], const hsize_t stride[], const hsize_t edge[], void *data)
```

<i>swathID</i>	IN: Swath id returned by SWcreate or SWattach
<i>fieldname</i>	IN: Name of field to write
<i>start</i>	IN: Array specifying the starting location within each dimension (0-based)
<i>stride</i>	IN: Array specifying the number of values to skip along each dimension
<i>edge</i>	IN: Array specifying the number of values to write along each dimension
<i>data</i>	IN: Values to be written to the field
Purpose	Writes data to a swath field.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or unknown fieldname.
Description	The values within <i>start</i> , <i>stride</i> , and <i>edge</i> arrays refer to the swath field (output) dimensions. The input data in the <i>data</i> buffer is read from contiguously. The default value for <i>stride</i> is 1 is used if this parameter is set to <i>NULL</i> . It is the users responsibility to make sure the data buffer contains sufficient entries to write to the field. Note that the data buffer for a compressed field must be the size of the entire field as incremental writes are not supported by the underlying HDF5 routines.

Example In this example, we write data to the *Longitude* field.

```
float32 longitude [2000][1000];  
hsize_t start[2] = {10,0};  
hsize_t edge[2] = {2000, 1000};  
/* Define elements of longitude array */  
status = SWwritefield(swathID, "Longitude", start, NULL,  
edge, longitude);
```

We now update Track 10 (0 - based) in this field:

```
float32 newtrack[1000];

hssize_t start[2]={0,10};

hsizen_t edge[2]={1,1000};

/* Define elements of newtrack array */

status = SWwritefield(swathID, "Longitude", start, NULL,
                      edge, newtrack);
```

**FORTRAN** integer function

```
swwrfld(swathid,fieldname,start,stride,edge,data)

integer      swathid
character(*)  fieldname
integer*4     start(*)
integer*4     stride(*)
integer*4     edge(*)
<valid type> data(*)
```

The *start*, *stride*, and *edge* arrays must be defined explicitly, with the *start* array being 0-based.

The equivalent *FORTRAN* code for the example above is:

```
real*4 longitude(1000,2000)

integer*4 start(2), stride(2), edge(2)

start(1) = 0
start(2) = 10
stride(1) = 1
stride(2) = 1
edge(1) = 1000
edge(2) = 2000

status = swwrfld(swathid, "Longitude", start, stride, edge,
                  longitude);
```

We now update Track 10 (0 - based) in this field:

```
real*4 newtrack(1000)
```

```
integer*4 start(2), stride(2), edge(2)

start(1) = 10

start(2) = 0

stride(1) = 1

stride(2) = 1

edge(1) = 1000

edge(2) = 1

status = swwrfld(swathid, "Longitude", start, stride, edge,
newtrack)
```

### 2.1.2 Grid Interface Functions

This section contains an alphabetical listing of all the functions in the Grid interface. The functions are alphabetized based on their C-language names.

# Attach to an Existing Grid Structure

---

## GDattach

hid\_t GDattach(hid\_t *fid*, char \**gridname*)

<i>fid</i>	IN: Grid file id returned by GDopen
<i>gridname</i>	IN: Name of grid to be attached
Purpose	Attaches to an existing grid within the file.
Return value	Returns the grid handle( <i>gridID</i> ) if successful or FAIL(-1) otherwise. Typical reasons for failure are improper grid file id or grid name.
Description	This routine attaches to the grid using the <i>gridname</i> parameter as the identifier.
Example	In this example, we attach to the previously created grid, "ExampleGrid", within the HDF file, <i>GridFile.h5</i> , referred to by the handle, <i>fid</i> :

```
gridID = GDattach(fid, "ExampleGrid");
```

The grid can then be referenced by subsequent routines using the handle, *gridID*.

**FORTRAN**

```
integer      function gdattach(fid, gridname)
integer      fid
character(*) gridname
```

The equivalent *FORTRAN* code for the example above is:

```
gridid = gdattach(fid, "ExampleGrid")
```

# Return Information About a Grid Attribute

---

## GDattrinfo

```
herr_t GDattrinfo(hid_t gridID, char *attrname, H5T_class_t *numbertype,
                  hsize_t *count)
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>attrname</i>	IN: Attribute name
<i>numbertype</i>	OUT: Data type class ID of attribute
<i>count</i>	OUT: Number of total bytes in attribute
Purpose	Returns information about a grid attribute
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise.
Description	This routine returns number type and number of elements (count) of a grid attribute.
Example	In this example, we return information about the <i>ScalarFloat</i> attribute.

```
status = GDattrinfo(pointID, "ScalarFloat", &nt, &count);
```

The *nt* variable will have the value 1 and *count* will have the value 4.

## FORTRAN

```
integer function gdattrinfo(gridid, attrname, ntype, count,)
integer      gridid
character(*) attrname
integer       ntype
integer*4     count
```

The equivalent *FORTRAN* code for the first example above is:

```
status = gdattrinfo(pointid, "ScalarFloat", nt, count);
status = GDdefproj(GDid_som, GCTP_SOM, NULL, NULL, projparm);
```

## Related Documents

An Album of Map Projections, USGS Professional Paper 1453, Snyder and Voxland, 1989

Map Projections - A Working Manual, USGS Professional Paper 1395, Snyder, 1987

# Close an HDF-EOS File

---

## GDclose

herr\_t GDclose(hid\_t *fid*)

*fid* IN: Grid file id returned by GDopen

Purpose Closes file.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise.

Description This routine closes the HDF grid file.

Example

```
status = GDclose(fid);
```

FORTRAN integer function gdclose(int32 *fid*)  
integer *fid*

The equivalent *FORTRAN* code for the example above is:

```
status = gdclose(fid)
```

# Retrive Compression Information for Field

---

## GDcompinfo

herr\_t GDcompinfo(hid\_t *gridID*, char \**fieldname*, char \**compcode*, intn *compparm*[*J*])

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>fieldname</i>	IN: Fieldname
<i>compcode</i>	OUT: HDF compression code
<i>compparm</i>	OUT: Compression parameters
Purpose	Retrieves compression information about a field.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description	This routine returns the compression code and compression parameters for a given field.
Example	To retreive the compression information about the <i>Opacity</i> field defined in the <i>GDdefcomp</i> section:

```
status = GDcompinfo(gridID, "Opacity", compcode, compparm);
```

The *compcode* parameter will be set to 4 and *compparm*[0] to 5.

**FORTRAN** integer function gdcompinfo(*gridid*,*fieldname* *compcode*, *compparm*)  
integer *gridid*  
character\*(\*) *fieldname*  
character\*(\*) *compcode*  
integer *compparm*(\*)

The equivalent *FORTRAN* code for the example above is:

```
status = gdcompinfo(gridid, 'Opacity', compcode, compparm)
```

The *compcode* parameter will be set to 4 and *compparm*(1) to 5.

# Create a New Grid Structure

---

## GDcreate

```
hid_t GDcreate(hid_t fid, const char *gridname, int32_t xdimsize, int32_t ydimsize, float64 upleftpt[], float64 lowrightpt[])
```

<i>fid</i>	IN: Grid file id returned by GDopen
<i>gridname</i>	IN: Name of grid to be created
<i>xdimsize</i>	IN: Number of columns in grid
<i>ydimsize</i>	IN: Number of rows in grid
<i>upleftpt</i>	IN: Location, of upper left corner of the upper left pixel
<i>lowrightpt</i>	IN: Location, of lower right corner of the lower right pixel
Purpose	Creates a grid within the file.
Return value	Returns the grid handle( <i>gridID</i> ) or FAIL(-1) otherwise.
Description	The grid is created as a Vgroup within the HDF file with the name <i>gridname</i> and class <i>GRID</i> . This routine establishes the resolution of the grid, ie, the number of rows and columns, and it's location within the complete global projection through the <i>upleftpt</i> and <i>lowrightpt</i> arrays. These arrays should be in meters for all GCTP projections other than the Geographic Projection, which should be in packed degree format. q.v. below.
Example	In this example, we create a UTM grid bounded by 54 E to 60 E longitude and 20 N to 30 N latitude. We divide it into 120 bins along the x-axis and 200 bins along the y-axis

```
uplft[0]=210584.50041;
uplft[1]=3322395.95445;
lowrgt[0]=813931.10959;
lowrgt[1]=2214162.53278;
xdim=120;
ydim=200;

gridID = GDcreate(fid, "UTMGrid", xdim, ydim, uplft,
lowrgt);
```

The grid structure is then referenced by subsequent routines using the handle, *gridID*.

The *xdim* and *ydim* values are referenced in the field definition routines by the reserved dimensions: *XDim* and *YDim*.

For the Polar Stereographic, Goode Homolosine and Lambert Azimuthal projections, we have established default values in the case of an entire hemisphere for the first projection, the entire globe for the second and the entire polar or equitorial projection for the third. Thus, if we have a Polar Stereographic projection of the Northern Hemisphere then the *uplft* and *lowrgt* arrays can be replaced by *NULL* in the function call.

In the case of the Geographic projection (linear scale in both longitude latitude), the *upleftpt* and *lowrightpt* arrays contain the longitude and latitude of these points in packed degree format (DDDDMMMSSS.SS).

Note:

**upleftpt** - Array that contains the X-Y coordinates of the upper left corner of the upper left pixel of the grid. First and second elements of the array contain the X and Y coordinates respectively. The upper left X coordinate value should be the lowest X value of the grid. The upper left Y coordinate value should be the highest Y value of the grid.

**lowrightpt** - Array that contains the X-Y corrdinates of the lower right corner of the lower right pixel of the grid. First and second elements of the array contain the X and Y coordinates respectively. The lower right X coordinate value should be the highest X value of the grid. The lower right Y coordinate value should be the lowest Y value of the grid.

If the projection id geographic (i.e., projcode=0) then the X-Y coordinates should be specified in degrees/minutes/seconds (DDDDMMMSSS.SS) format. The first element of the array holds the longitude and the second element holds the latitude. Latitudes are from -90 to +90 and longitudes are from -180 to +180 (west is negative).

For all other projection types the X-Y coordinates should be in **meters** in double precision. These coordinates have to be computed using the GCTP software with the same projection parameters that have been specified in the projparm array. For UTM projections use the same zone code and its sign (positive or negative) while computing both upper left and lower right corner X-Y coordinates irrespective of the hemisphere.

To convert lat/long to x-y coordinates, it is also possible to use SDP Toolkit routines: PGS\_GCT\_Init() or PGS\_GCT\_Proj(). More information is contained in the *SDP Toolkit Users Guide for the ECS Project (333-CD-500-001)*.

```
FORTRAN integer function gdcreate(fid, gridname, xdimsize, ydimsize, upleftpt,  
lowrightpt)  
    integer      fid  
    character*(*) gridname  
    integer*4     xdimsize  
    interger*4    ydimsize  
    real*8       upleftpt(*)  
    real*8       lowrightpt(*)
```

The equivalent *FORTRAN* code for the example above is:

```
gridid = gdcreate(fid, "UTMGrid", xdim, ydim, uplft,  
                  lowrgt)
```

The default values for the Polar Stereographic and Goode Homolosine can be designated by setting all elements in the *uplft* and *lowrgt* arrays to 0.

# Define Region of Interest by Latitude/Longitude

---

## GDdefboxregion

hid\_t GDdefboxregion(hid\_t *gridID*, float64 *cornerlon[]*, float64 *cornerlat[]*)

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>cornerlon</i>	IN: Longitude in decimal degrees of box corners
<i>cornerlat</i>	IN: Latitude in decimal degrees of box corners
Purpose	Defines a longitude-latitude box region for a grid.
Return value	Returns the grid region ID if successful or FAIL (-1) otherwise.
Description	This routine defines a longitude-latitude box region for a grid. It returns a grid region ID which is used by the <i>GExtractregion</i> routine to read all the entries of a data field within the region.
Example	In this example, we define the region to be the first quadrant of the Northern hemisphere.

```
cornerlon[0] = 0.;  
cornerlat[0] = 90.;  
cornerlon[1] = 90.;  
cornerlat[1] = 0.;  
  
regionID = GDdefboxregion(GDid, cornerlon, cornerlat);
```

**FORTRAN**      integer function gddefboxreg(*gridid*, *cornerlon*, *cornerlat*)  
                integer       *gridid*  
                real\*8       *cornerlon*  
                real\*8       *cornerlat*

The equivalent *FORTRAN* code for the example above is:

```
cornerlon(1) = 0.  
cornerlat(1) = 90.  
cornerlon(2) = 90.  
cornerlat(2) = 0.  
  
regionid = gddefboxreg(gridid, cornerlon,cornerlat)
```

# Set Grid Field Compression

---

## GDdefcomp

herr\_t GDdefcomp(hid\_t *gridID*, int32\_t *compcode*, intn *compparm*[])

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>compcode</i>	IN: Compression method
<i>compparm</i>	IN: Compression parameters (if applicable)
Purpose	Sets the field compression for all subsequent field definitions.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description	This routine sets the HDF field compression for subsequent grid field definitions. The compression does not apply to one-dimensional fields. The compression methods currently supported are: deflate (HDFE_COMP_DEFLATE=4) and no compression (HDFE_COMP_NONE = 0, the default). Deflate compression requires a single integer compression parameter in the range of one to nine with higher values corresponding to greater compression. Compressed fields are written using the standard <i>GDwritefield</i> routine, however, the entire field must be written in a single call. If this is not possible, the user should consider tiling. See <i>GDdeftile</i> for further information. Any portion of a compressed field can then be accessed with the <i>GDreadfield</i> routine. Compression takes precedence over merging so that multi-dimensional fields that are compressed are not merged. The user should refer to the HDF Reference Manual for a fuller explanation of the compression schemes and parameters.

Example	Suppose we wish to compress the <i>Opacity</i> field using deflate compression, and use no compression for the <i>Temperature</i> field.
---------	--

```
status = GDdeffield(gridID, "Pressure", "YDim,XDim", NULL,
H5T_NATIVE_FLOAT, HDFE_NOMERGE);

compparm[0] = 5;

status = GDdefcomp(gridID, HDFE_COMP_DEFLATE, compparm);

status = GDdefcomp(gridID, HDFE_COMP_NONE, NULL);

status = GDdeffield(gridID, "Temperature", "YDim,XDim",
NULL, DFNT_FLOAT32, HDFE_AUTOMERGE);
```

Note that the HDFE\_AUTOMERGE parameter will be ignored in the *Temperature* field definition.

**FORTRAN**    integer function gddefcomp(*gridid*, *compcode*, *compparm*)  
              integer        *gridid*  
              integer        *compcode*  
              integer        *compparm*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_COMP_NONE=0)  
parameter (HDFE_COMP_DEFLATE=4)  
integer compparm(5)  
compparm(1) = 5  
status = gddefcomp(gridid, HDFE_COMP_DEFLATE, compparm)  
status = gddeffld(gridid, "Opacity", "YDim,XDim",  
HDFE_NATIVE_FLOAT, HDFE_NOMERGE)  
status = gddefcomp(gridid, HDFE_COMP_NONE, compparm)  
status = gddeffld(gridid, "Temperature", "YDim,XDim",  
H5T_NATIVE_FLOAT, HDFE_AUTOMERGE)
```

# Define a New Dimension Within a Grid

---

## GDdefdim

herr\_t GDdefdim(hid\_t *gridID*, char \**dimname*, int32\_t *dim*)

<i>gridID</i>	IN:	Grid id returned by GDcreate or GDattach
<i>dimname</i>	IN:	Name of dimension to be defined
<i>dim</i>	IN:	The size of the dimension
Purpose	Defines a new dimension within the grid.	
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reason for failure is an improper grid id.	
Description	This routine defines dimensions that are used by the field definition routines (described subsequently) to establish the size of the field.	
Example	In this example, we define a dimension, <i>Band</i> , with size 15.	

```
status = GDdefdim(gridID, "Band", 15)
```

To specify an unlimited dimension which can be used to define an appendable array, the dimension value should be set to H5S\_UNLIMITED:

```
status = GDdefdim(gridID, "Unlim", H5S_UNLIMITED);  
FORTRAN integer function gddefdim(gridid, fieldname, dim)  
integer gridid  
character*(*) fieldname  
integer*4 dim
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (UNLIMITED=0)  
status = gddefdim(gridid, "Band", 15)  
status = gddefdim(gridid, "Unlim", UNLIMITED)
```

# Define a New Data Field Within a Grid

---

## GDdeffield

```
herr_t GDdeffield(hid_t gridID, char *fieldname, char *dimlist, char *maxdimlist,  
                   hid_t numbertype, int32_t merge)
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>fieldname</i>	IN: Name of field to be defined
<i>dimlist</i>	IN: The list of data dimensions defining the field
<i>maxdimlist</i>	IN: The list of data maximum dimensions defining the field
<i>numbertype</i>	IN: The data type of the data stored in the field
<i>merge</i>	IN: Merge code (HDFE-NOMERGE (0) - no merge, HDFE_AUTOMERGE (1) -merge)
Purpose	Defines a new data field within the grid.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reason for failure is an unknown dimension in the dimension list.
Description	This routine defines data fields to be stored in the grid. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in C order, that is, the last dimension is incremented first. The API will attempt to merge into a single object those fields that share dimensions and in case of multidimensional fields, numbertype. Two and three dimensional fields will be merged into a single three-dimensional object if the last two dimensions (in C order are equal). If the merge code for a field is set to 0, the API will not attempt to merge it with other fields. Fields using the unlimited dimension will not be merged. Because merging breaks the one-to-one correspondence between HDF-EOS fields and HDF SDS arrays, it should not be set if the user wishes to access the HDF-EOS field directly using HDF routines or, for example, to create an HDF attribute corresponding to the field. If maximum dimensions are the same as input dimensions, the “NULL” should be passed as a fourth parameter. In case of unlimited dimension (appendable field) the corresponding dimension should be defined by calling GDdefdim() with the H5S_UNLIMITED as third parameter.
Example	In this example, we define a grid field, <i>Temperature</i> with dimensions <i>XDim</i> and <i>YDim</i> (as established by the <i>GDcreate</i> routine) containing 4-

byte floating point numbers and a field, *Spectra*, with dimensions *XDim*, *YDim*, and *Bands*:

```
status = GDdeffield(gridID, "Temperature", "YDim,XDim",
NULL, H5T_NATIVE_FLOAT, HDFE_AUTOMERGE);

status = GDdeffield(gridID, "Spectra", "Bands,YDim,XDim",
NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);

FORTRAN integer function gddeffld(gridid,fieldname, dimlist, numbertype, merge)

integer      gridid
character(*) fieldname
character(*) dimlist
character(*) maxdimlist
integer       numbertype
integer*4     merge
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_NATIVE_FLOAT= 1)
parameter (HDFE_NOMERGE=0)
parameter (HDFE_AUTOMERGE=1)

status = gddeffld(gridid, "Temperature", "XDim,YDim", NULL,
HDFE_NATIVE_FLOAT, HDFE_AUTOMERGE)

status = gddeffld(gridid, "Spectra", "XDim,YDim,Bands", " ",
HDFE_NATIVE_FLOAT, HDFE_NOMERGE)
```

The dimensions are entered in *FORTRAN* order with the first dimension incremented first.

# Define the Origin of the Grid Data

---

## GDdeforigin

herr\_t GDdeforigin(hid\_t *gridID*, int32\_t *origincode*)

*gridID* IN: Grid id returned by GDcreate or GDattach

*origincode* IN: Location of the origin of the grid data

Purpose Defines the origin of the grid data

Return Value Returns SUCCEED(0) if successful or FAIL(-1) otherwise

Description The routine is used to define the origin of the grid data. This allows the user to select any corner of the grid as the origin.

Origin Codes:

HDFE\_GD\_UL(0) Upper Left corner of grid

HDFE\_GD\_UR(1) Upper Right corner of grid

HDFE\_GD\_LL(2) Lower Left corner of grid

HDFE\_GD\_LR(3) Lower Right corner of grid

Example In this example we define the origin of the grid to be the Lower Right corner:

```
status = GDdeforigin(gridID, HDFE_GD_LR);
```

FORTRAN integer function gddeforg(*gridid*, *origincode*)

integer *gridid*

integer\*4 *origincode*

The equivalent *FORTRAN* code for the above example is :

```
parameter (HDFE_GD_LR=3)
```

```
status = gddeforg(gridid, HDFE_GD_LR)
```

# Define a Pixel Registration Within a Grid

---

## GDdefpixreg

herr\_t GDdefpixreg(hid\_t *gridID*, int32\_t *pixreg*)

*gridID* IN: Grid id returned by GDcreate or GDattach

*pixreg* IN: Pixel registration

Purpose Defines pixel registration within grid cell

Return Value Returns SUCCEED(0) if successful or FAIL(-1) otherwise.

Description This routine is used to define whether the pixel center or pixel corner (as defined by the *GDdeforigin* routine) is used when requesting the location (longitude and latitude) of a given pixel.

Registration Codes:

HDFE\_CENTER (0) (Default) Center of pixel cell

HDFE\_CORNER (1) Corner of a pixel cell

Example In this example, we define the pixel registration to be the corner of the pixel cell:

```
status = GDdefpixreg(gridID, HDFE_CORNER);
```

**FORTRAN** integer function gddefpreg(*gridid*, *pixreg*)

integer *gridid*

integer\*4 *pixreg*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HDFE_CORNER=1)
```

```
status = gddefpreg(gridid, HDFE_CORNER)
```

# Define Grid Projection

---

## GDdefproj

```
herr_t GDdefproj(hid_t gridID, int32_t projcode, int32_t zonecode, int32_t spherocode, float64 projparm[])
```

*gridID* IN: Grid id returned by GDcreate or GDattach

*projcode* IN: GCTP projection code

*zonecode* IN: GCTP zone code used by UTM projection

*spherocode* IN: GCTP spheroid code

*projparm* IN: GCTP projection parameter array

Purpose Defines projection of grid

Return Value Returns SUCCEED(0) if successful or FAIL(-1) otherwise

Description Defines the GCTP projection and projection parameters of the grid.

Example In this example, we define a Universal Transverse Mercator (UTM) grid bounded by 54 E - 60 E longitude and 20 N - 30 N latitude – UTM zonecode 40, using default spheroid (Clarke 1866), spherocode = 0

```
spherocode = 0;  
zonecode = 40;  
  
status = GDdefproj(gridID, GCTP_UTM, zonecode, spherocode,  
NULL);
```

In this next example we define a Polar Stereographic projection of the Northern Hemisphere (True scale at 90 N, 0 Longitude below pole) using the International 1967 spheroid.

```
spherocode = 3;  
  
for (i = 0; i < 13; i++) projparm[i] = 0;  
  
/* Set Long below pole & true scale in DDDMMSSS.SSS form */  
  
projparm[5] = 90000000.00;  
  
status = GDdefproj(gridID, GCTP_PS, NULL, spherocode,  
projparm);
```

Finally we define a Geographic projection. In this case neither the zone code, sphere code or the projection parameters are used.

```

status = GDdefproj(gridID, GCTP_GEO, NULL, NULL, NULL)

FORTRAN integer function gddefproj(gridid, projcode, zonecode, spherecode,
                                   projparm)

integer      gridid
integer*4    projcode
integer*4    zonecode
integer*4    spherecode
real*8      projparm(*)

```

The equivalent FORTRAN code for the examples above is:

```

parameter (GCTP_UTM=1)

spherecode = 0

zonecode = 40

status = gddefproj(gridid, GCTP_UTM, zonecode, spherecode,
                    dummy)

parameter (GCTP_PS=6)

spherecode = 6

do i=1,13

    projparm(i) = 0

enddo

projparm(6) = 90000000.00

status = gddefproj(gridid, GCTP_PS, dummy, spherecode,
                    projparm)

parameter (GCTP_GEO=0)

status = gddefproj(gridid, GCTP_GEO, dummy, dummy, dummy)

```

Note: projcode, zonecode, spherecode and projection parameter information are listed in Section 1.6, GCTP Usage.

# Define Tiling Parameters

---

## GDdeftile

herr\_t GDdeftile(hid\_t *gridID*, int32\_t *tilecode*, int32\_t *tilerank*, int32\_t *tiledims[]*)

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>tilecode</i>	IN: Tile code: HDF_TILE, HDF_NOTILE (default)
<i>tilerank</i>	IN: The number of tile dimensions
<i>tiledims</i>	IN: Tile dimensions
Purpose	Defines tiling dimensions for subsequent field definitions
Return Value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise
Description	This routine defines the tiling dimensions for fields defined following this function call, analogous to the procedure for setting the field compression scheme using <i>GDdefcomp</i> . The number of tile dimensions and subsequent field dimensions must be the same and the tile dimensions must be integral divisors of the corresponding field dimensions. A tile dimension set to 0 will be equivalent to 1.
Example	We will define four fields in a grid, two two-dimensional fields of the same size with the same tiling, a three-dimensional field with a different tiling scheme, and a fourth with no tiling. We assume that <i>XDim</i> is 200 and <i>YDim</i> is 300.

```
tiledims[0] = 100;  
tiledims[1] = 200;  
  
status = GDdeftile(gridID, HDFE_TILE, 2, tiledims);  
  
status = GDdeffield(gridID, "Pressure", "YDim,XDim", NULL,  
H5T_NATIVE_INT, HDFE_NOMERGE);  
  
status = GDdeffield(gridID, "Temperature", "YDim,XDim",  
NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);  
  
tiledims[0] = 1;  
tiledims[1] = 150;  
tiledims[2] = 100;  
  
status = GDdeftile(gridID, HDFE_TILE, 3, tiledims);
```

```

status = GDdeffield(gridID, "Spectra", "Bands,YDim,XDim",
NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);

status = GDdeftile(gridID, HDFE_NOTILE, 0, NULL);

status = GDdeffield(gridID, "Communities", "YDim,XDim",
NULL, H5T_NATIVE_INT, HDFE_AUTOMERGE);

```

**FORTRAN**

```

integer function gddeftle(gridid, tilecode,tilerank,tiledims)
integer      gridid
integer*4    tilecode
integer*4    tilerank
integer*4    tiledims(*)

```

The equivalent *FORTRAN* code for the example above is:

```

parameter (HDFE_NOTILE=0)

parameter (HDFE_NATIVE_INT = 0)

parameter (HDFE_TILE=1)

tiledims(1) = 200

tiledims(2) = 100

status = gddeftle(gridid, HDFE_TILE, 2, tiledims)

status = gddeffld(gridid, 'Pressure', 'XDim,YDim', " ",
HDFE_NATIVE_INT, HDFE_NOMERGE)

status = gddeffld(gridid, 'Temperature', 'XDim,YDim', " ",
HDFE_NATIVE_FLOAT, HDFE_NOMERGE)

tiledims[1] = 100

tiledims[2] = 150

tiledims[3] = 1

status = gddeftle(gridid, HDFE_TILE, 3, tiledims)

status = gddeffld(gridid, 'Spectra', 'XDim,YDim,Bands', " ",
HDFE_NATIVE_FLOAT, HDFE_NOMERGE)

status = gddeftle(gridid, HDFE_NOTILE, 0, tiledims);

status = gddeffld(gridid, 'Communities', 'XDim,YDim', " ",
HDFE_NATIVE_INT, HDFE_AUTOMERGE)

```

# Define a Time Period of Interest

---

## GDdeftimeperiod

```
hid_t GDdeftimeperiod(hid_t gridID, hid_t periodID, float64 starttime, float64 stoptime)
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>periodID</i>	IN: Period (or region) id from previous subset call
<i>starttime</i>	IN: Start time of period
<i>stoptime</i>	IN: Stop time of period
Purpose	Defines a time period for a grid.
Return value	Returns the grid period ID if successful or FAIL (-1) otherwise.
Description	This routine defines a time period for a grid. It returns a grid period ID which is used by the <i>GDefextractperiod</i> routine to read all the entries of a data field within the time period.. The grid structure must have the <i>Time</i> field defined. This routine may be called after <i>GDefboxregion</i> to provide both geographic and time subsetting . In this case the user provides the id from the previous subset call. (This same id is then returned by the function.) Furthermore it can be called before or after <i>GDefvrtrregion</i> to further refine a region. This routine may also be called “stand-alone” by setting the input id to HDFE_NOPREVSUB (-1).

Example      In this example, we define a time period with a start time of 35232487.2 and a stop time of 36609898.1.

```
starttime = 35232487.2;  
stoptime = 36609898.1;  
  
periodID = GDdeftimeperiod(gridID, HDFE_NOPREVSUB  
starttime, stoptime);
```

If we had previously performed a geographic subset with id, *regionID*, then we could further time subset this region with the call:

```
periodID = GDdeftimeperiod(gridID, regionID, starttime,  
stoptime);
```

Note that *periodID* will have the same value as *regionID*.

```
FORTRAN    integer function gddeftmeper(gridid, periodID, starttime, stoptime)
            integer      gridid
            integer      periodid
            real*8       starttime
            real*8       stoptime
```

The equivalent *FORTRAN* code for the examples above are:

```
parameter (HDFE_NOPREVSUB=-1)
starttime = 35232487.2
stoptime = 36609898.1
periodid = gddeftmeper(gridid, HDFE_NOPREVSUB, starttime,
                      stoptime)
periodid = gddeftmeper(grdid, regionid, starttime, stoptime)
```

# Define a Vertical Subset Region

---

## GDdefvrtrregion

```
int32_t GDdefvrtrregion(hid_t gridID, int32_t regionID, char *vertObj, float64 range[])
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>regionID</i>	IN: Region (or period) id from previous subset call
<i>vertObj</i>	IN: Dimension or field to subset
<i>range</i>	IN: Minimum and maximum range for subset
Purpose	Subsets on a <b>monotonic</b> field or contiguous elements of a dimension.
Return value	Returns the grid region ID if successful or FAIL (-1) otherwise.
Description	Whereas the <i>GDdefboxregion</i> routine subsets along the <i>XDim</i> and <i>YDim</i> dimensions, this routine allows the user to subset along any other dimension. The region is specified by a set of minimum and maximum values and can represent either a dimension index (case 1) or field value range(case 2). In the second case, the field must be one-dimensional and the values must be <b>monotonic</b> (strictly increasing or decreasing) in order that the resulting dimension index range be contiguous. (For the current version of this routine, the second option is restricted to fields with number type: INT16, INT32, FLOAT32, FLOAT64.) This routine may be called after <i>GDdefboxregion</i> to provide both geographic and “vertical” subsetting. In this case the user provides the id from the previous subset call. (This same id is then returned by the function.) This routine may also be called “stand-alone” by setting the input id to HDFE_NOPREVSUB (-1).

This routine may be called up to eight times with the same region ID. It this way a region can be subsetted along a number of dimensions.

The *GDregioninfo* and *GDEXTRACTregion* routines work as before, however the field to be subsetted, (the field specified in the call to *GDregioninfo* and *GDEXTRACTregion*) must contain the dimension used explicitly in the call to *GDdefvrtrregion* (case 1) or the dimension of the one-dimensional field (case 2).

Example Suppose we have a field called *Pressure* of dimension *Height* (= 10) whose values increase from 100 to 1000. If we desire all the elements with values between 500 and 800, we make the call:

```

range[0] = 500.;
range[1] = 800.;

regionID = GDdefvrtrregion(gridID, HDFE_NOPREVSUB,
"Pressure", range);

```

The routine determines the elements in the *Height* dimension which correspond to the values of the *Pressure* field between 500 and 800.

If we wish to specify the subset as elements 2 through 5 (0 - based) of the *Height* dimension, the call would be:

```

range[0] = 2;
range[1] = 5;

regionID = GDdefvrtrregion(gridID, HDFE_NOPREVSUB,
"DIM:Height", range);

```

The “DIM:” prefix tells the routine that the range corresponds to elements of a dimension rather than values of a field.

If a previous subset region or period was defined with id, *subsetID*, that we wish to refine further with the vertical subsetting defined above we make the call:

```
regionID = GDdefvrtrregion(gridID, subsetID, "Pressure",
range);
```

The return value, *regionID* is set equal to *subsetID*. That is, the subset region is modified rather than a new one created.

In this example, any field to be subsetted must contain the *Height* dimension.

**FORTRAN**

```

integer function gddefvrtrreg(gridid, regionid, vertobj, range)
integer      gridid
integer      regionid
character(*) vertobj
real*8       range

```

The equivalent *FORTRAN* code for the examples above is:

```

parameter (HDFE_NOPREVSUB=-1)
range(1) = 500.
range(2) = 800.
regionid = gddefvrtrreg(gridid, HDFE_NOPREVSUB, "Pressure",
range)
range(1) = 3          ! Note 1-based element numbers
range(2) = 6

```

```
regionid = gddefvrtrg(gridid, HDFE_NOPREVSUB, "DIM:Height",  
range)  
regionid = gddefvrtrg(gridid, subsetid, "Pressure", range)
```

# Detach from Grid Structure

---

## GDdetach

herr\_t GDdetach(hid\_t *gridID*)

*gridID* IN: Grid id returned by GDcreate or GDattach

Purpose Detaches from grid interface.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise.

Description This routine should be run before exiting from the grid file for every grid opened by GDcreate or GDattach.

Example In this example, we detach the grid structure, *ExampleGrid*:

```
status = GDdetach(gridID)
```

**FORTRAN** integer function gddetach(gridid)

```
integer      gridid
```

The equivalent *FORTRAN* code for the example above is:

```
status = gddetach(gridid)
```

# Retrieve Size of Specified Dimension

---

## GDdiminfo

hsize\_t GDdiminfo(hid\_t *gridID*, char \**dimname*)

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>dimname</i>	IN: Dimension name
Purpose	Retrieve size of specified dimension.
Return value	Size of dimension if successful or FAIL(-1) otherwise. A typical reason for failure is an improper grid id or dimension name.
Description	This routine retrieves the size of specified dimension.
Example	In this example, we retrieve information about the dimension, "Bands":

```
dimsize = GDdiminfo(gridID, "Bands");
```

The return value, *dimsize*, will be equal to 15

**FORTRAN**      integer\*4 function gddiminfo(*gridid*,*dimname*)  
                  integer          *gridid*  
                  character(\*)  *dimname*

The equivalent *FORTRAN* code for the example above is:

```
dimsize = gddiminfo(gridid, "Bands")
```

# Duplicate a Region or Period

---

## GDdupregion

hid\_t GDdupregion(hid\_t *regionID*)

<i>regionID</i>	IN: Region or period id returned by GDdefboxregion, GDdeftimeperiod, or GDdefvrregion.
Purpose	Duplicates a region.
Return value	Returns new region or period ID if successful or FAIL (-1) otherwise.
Description	This routine copies the information stored in a current region or period to a new region or period and generates a new id. It is usefully when the user wishes to further subset a region (period) in multiple ways.

Example In this example, we first subset a grid with *GDdefboxregion*, duplicate the region creating a new region ID, *regionID2*, and then perform two different vertical subsets of these (identical) geographic subset regions:

```
regionID = GDdefboxregion(gridID, cornerlon, cornerlat);

regionID2 = GDdupregion(regionID);

regionID = GDdefvrregion(gridID, regionID, "Pressure",
rangePres);

regionID2 = GDdefvrregion(gridID, regionID2, "Temperature",
rangeTemp);
```

**FORTRAN**    integer        gddupreg(*regionid*)  
                integer        *regionid*

The equivalent *FORTRAN* code for the example above is:

```
regionid = gddefboxreg(gridid, cornerlon, cornerlat)

regionid2 = gddupreg(regionid)

regionid = gddefvrreg(gridid, regionid, 'Pressure',
rangePres)

regionid2 = gddefvrreg(gridid, regionid2, 'Temperature',
rangeTemp)
```

# Read a Region of Interest from a Field

---

## GDextractregion

```
herr_t GDextractregion(hid_t gridID, hid_t regionID, char *fieldname, void *buffer)
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>regionID</i>	IN: Region (period) id returned by GDdefboxregion (GDdeftimeperiod)
<i>fieldname</i>	IN: Field to subset
<i>buffer</i>	OUT: Data Buffer
Purpose	Extracts (reads) from subsetted region.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise.
Description	This routine reads data into the data buffer from a subsetted region as defined by GDdefboxregion.
Example	In this example, we extract data from the “Temperature” field from the region defined in <i>GDdefboxregion</i> . We first allocate space for the data buffer. The size of the subsetted region for the field is given by the Gdregioninfo routine.

```
datbuf = (float32) calloc(size, 4);  
  
status = GDextractregion(GDDid, regionID, "Temperature",  
datbuf32);
```

**FORTRAN**

integer	function gdextreg(gridid, regionid, fieldname, datbuf)
integer	gridid
integer	regionid
character(*)	fieldname
<valid type>	buffer(*)

The equivalent *FORTRAN* code for the example above is:

```
status = gdextreg(gridid, regionid, "Temperature", datbuf)
```

# Retrieve Information About Data Field in a Grid

---

## GDfieldinfo

```
herr_t GDfieldinfo(hid_t gridID, char *fieldname, int32_t rank, int32_t dims[],  
                    H5T_class_t *numbertype, char *dimlist)
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>fieldname</i>	IN: Fieldname
<i>rank</i>	OUT: Pointer to rank of the field
<i>dims</i>	OUT: Array containing the dimension sizes of the field
<i>numbertype</i>	OUT: Data type class ID of the field
<i>dimlist</i>	OUT: Dimension list
Purpose	Retrieve information about a specific geolocation or data field in the grid.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise. A typical reason for failure is the specified field does not exist.
Description	This routine retrieves information on a specific data field.
Example	In this example, we retrieve information about the <i>Spectra</i> data fields:

```
status = GDfieldinfo(gridID, "Spectra", &rank, dims,  
                      &numbertype, dimlist);
```

The return parameters will have the following values:

*rank*=3, *numbertype*= 1, *dims*[3]={15,200,120} and  
*dimlist*="Bands,YDim,XDim"

FORTRAN    integer function gdfldinfo(*gridid*, *fieldname*, *rank*, *dims*, *numbertype*,  
*dimlist*)  
  
               integer          *gridid*  
               character\*(\*)    *fieldname*  
               integer\*4        *rank*  
               integer\*4        *dims(\*)*  
               integer         *numbertype*  
               character(\*)    *dimlist*

The equivalent *FORTRAN* code for the example above is:

```
status = gdfldinfo(gridid, "Spectra", dims, rank,
numbertype, dimlist)
```

The return parameters will have the following values:

*rank*=3, *numbertype*=5, *dims[3]*={120,200,15} and

*dimlist*="XDim,YDim,Bands"

Note that the dimensions array and the dimension list are in *FORTRAN* order.

# Get Fill Value for Specified Field

---

## GDgetfillvalue

herr\_t GDgetfillvalue(hid\_t *gridID*, char \**fieldname*, void \**fillvalue*)

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>fieldname</i>	IN: Fieldname
<i>fillvalue</i>	OUT: Space allocated to store the fill value
Purpose	Retrieves fill value for the specified field.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are an improper grid id or number type or incorrect fill value.
Description	It is assumed the number type of the fill value is the same as the field.

Example In this example, we get the fill value for the "Temperature" field:

```
status = GDgetfillvalue(gridID, "Temperature", &tempfill);
```

**FORTRAN** integer function gdgetfill(*gridid,fieldname,fillvalue*)  
integer *gridid*  
character(\*) *fieldname*  
<valid type> *fillvalue(\*)*

The equivalent *FORTRAN* code for the example above is:

```
status = gdgetfill(gridid, "Temperature", tempfill)
```

# Get Row/Columns for Specified Longitude/Latitude Pairs

---

## GDgetpixels

```
herr_t GDgetpixels(hid_t gridID, int32_t nLonLat, float64 lonVal[], float64 latVal[], int32_t pixRow[], int32_t pixCol[])
```

<i>gridID</i>	IN:	Grid id returned by GDcreate or GDattach
<i>nLonLat</i>	IN:	Number of longitude/latitude pairs
<i>lonVal</i>	IN:	Longitude values in degrees
<i>latVal</i>	IN:	Latitude values in degrees
<i>pixRow</i>	OUT:	Pixel Rows
<i>pixCol</i>	OUT:	Pixel Columns
Purpose	Returns the pixel rows and columns for specified longitude/latitude pairs.	
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise.	
Description	This routine converts longitude/latitude pairs into (0 - based) pixel rows and columns. The origin is the upper left-hand corner of the grid. This routine is the pixel subsetting equivalent of <i>GDdefboxregion</i> .	
Example	To convert two pairs of longitude/latitude values to rows and columns, make the following call:	

```
lonArr[0] = 134.2;  
latArr[0] = -20.8;  
lonArr[1] = 15.8;  
latArr[1] = 84.6;  
  
status = GDgetpixels(gridID, 2, lonArr, latArr, rowArr,  
colArr);
```

The row and column of the two pairs will be returned in the *rowArr* and *colArr* arrays.

```
FORTRAN integer function gdgetpix(gridid, nlonlat, lonval, latval, pixrow, pixcol)
      integer      gridid
      integer*4    nlonlat
      real*8       lonval
      real*8       latval
      integer*4    pixrow(*)
      integer*4    pixcol(*)
```

The equivalent *FORTRAN* code for the example above is:

```
lonarr(1) = 134.2
latarr(1) = -20.8
lonarr(2) = 15.8
latarr(2) = 84.6
status = gdgetpix(gridid, 2, lonarr, latarr, rowarr, colarr)
```

Note that the row and columns values will be 1 - based.

# Return Information About a Grid Structure

---

## GDgridinfo

```
herr_t GDgridinfo(hid_t gridID, int32_t *xdimsize, int32_t *ydimsize, float64 upleft[], float64 lowright[])
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>xdimsize</i>	OUT: Number of columns in grid
<i>ydimsize</i>	OUT: Number of rows in grid
<i>upleft</i>	OUT: Location, in meters, of upper left corner
<i>lowright</i>	OUT: Location, in meters, of lower right corner
Purpose	Returns position and size of grid
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise
Description	This routine returns the number of rows, columns and the location, in meters, of the upper left and lower right corners of the grid image.
Example	In this example, we retrieve information from a previously created grid with a call to GDattach:

```
status = GDgridinfo(gridID, &xdimsize, &ydimsize, upleft,  
                     lowrgt);
```

**FORTRAN**      integer function gdgridinfo(*gridid*, *xdimsize*, *ydimsize*, *upleft*, *lowright*)  
                  integer       *gridid*  
                  integer\*4   *xdimsize*  
                  integer\*4   *ydimsize*  
                  real\*8      *upleft(\*)*  
                  real\*8      *lowright(\*)*

The equivalent FORTRAN code for the example above is:

```
status = gdgridinfo(gridid, xdimsize, ydimsize, upleft,  
                     lowrgt);
```

# Retrieve Information About Grid Attributes

---

## GDinqattrs

int32\_t GDinqattrs(hid\_t *gridID*, char \**attrlist*, int32 \**strbufsize*)

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>attrlist</i>	OUT: Attribute list (entries separated by commas)
<i>strbufsize</i>	OUT: String length of attribute list
Purpose	Retrieve information about attributes defined in grid.
Return value	Number of attributes found if successful or FAIL (-1) otherwise.
Description	The attribute list is returned as a string with each attribute name separated by commas. If <i>attrlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator.

Example In this example, we retrieve information about the attributes defined in a grid structure. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = GDinqattrs(gridID, NULL, strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 14.

```
nattr = GDinqattrs(gridID, attrlist, strbufsize);
```

The variable, *attrlist*, will be set to:

"*attrOne,attr\_2*".

FORTRAN integer\*4 function gdinqattrs(*gridid,attrlist,strbufsize*)  
integer *gridid*  
character(\*) *attrlist*  
integer\*4 *strbufsize*

The equivalent *FORTRAN* code for the example above is:

```
nattr = gdinqattrs(gridid, attrlist, strbufsize)
```

# Retrieve Information About Data Type of a Data Field Defined in Grid

---

## GDinqdatatype

```
hid_t GDinqdatatype(hid_t swathID, char *fieldname, intn fieldgroup, hid_t  
*datatype, H5T_class_t *classid, H5T_order_t *order, size_t  
*size)
```

<i>gridID</i>	IN:	Grid id returned by GDcreate or GDattach
<i>fieldname</i>	IN::	String containing the name of a field
<i>fieldgroup</i>	IN:	Group (Data Field ) the data field belongs to
<i>datatype</i>	OUT:	Data type ID of a data field dataset
<i>classid</i>	OUT:	Data type class ID
<i>order</i>	OUT:	Byte order of an atomic datatype
<i>size</i>	OUT:	Size of a datatype (in bytes)
Purpose	Retrieve information about the data type of a specified data field.	
Return value	Status variable (0 if successful or -1 otherwise).	
Description		
Example		

FORTRAN

# Retrieve Information About Dimensions Defined in Grid

---

## GDinqdims

int32\_t GDinqdims(hid\_t *gridID*, char \**dimname*, int32\_t *dims*[])

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>dimname</i>	OUT: Dimension list (entries separated by commas)
<i>dims</i>	OUT: Array containing size of each dimension
Purpose	Retrieve information about dimensions defined in grid.
Return value	Number of dimension entries found if successful or FAIL(-1) otherwise. A typical reason for failure is an improper grid id.
Description	The dimension list is returned as a string with each dimension name separated by commas. Output parameters set to <i>NULL</i> will not be returned.

Example To retrieve information about the dimensions, use the following statement:

```
ndim = GDinqdims(gridID, dimname, dims);
```

The parameter, *dimname*, will have the value: "Xgrid,Ygrid,Bands"

with *dims*[3]={120,200,15}

FORTRAN  
integer\*4 function gdinqdims(*gridid*,*dimname*,*dims*)  
integer *gridid*  
character\*(\*) *dimname*  
integer\*4 *dims*(\*)

The equivalent *FORTRAN* code for the example above is:

```
ndim = gdinqdims(gridid, dimname, dims)
```

# Retrieve Information About Data Fields Defined in Grid

---

## GDinqfields

```
int32_t GDinqfields(hid_t gridID, char *fieldlist, int32_t rank[], H5T_class_t numbertype[])
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>fieldlist</i>	OUT: Listing of data fields (entries separated by commas)
<i>rank</i>	OUT: Array containing the rank of each data field
<i>numbertype</i>	OUT: Array containing data type class ID of each data field
Purpose	Retrieve information about the data fields defined in grid.
Return value	Number of data fields found if successful or FAIL(-1) otherwise. A typical reason is an improper grid id.
Description	The field list is returned as a string with each data field separated by commas. The <i>rank</i> and <i>numbertype</i> arrays will have an entry for each field. Output parameters set to <i>NULL</i> will not be returned.
Example	To retrieve information about the data fields, use the following statement:

```
nfld = GDinqfields(gridID, fieldlist, rank, numbertype);
```

The parameter, *fieldlist*, will have the value: "*Temperature,Spectra*"

with *rank[2]={2,3}*, *numbertype[2]={1,1}*

**FORTRAN**      integer\*4 function gdinqflds(*gridid, fieldlist, rank, numbertype*)  
                  integer          *gridid*  
                  character(\*)  *fieldlist*  
                  integer\*4      *rank(\*)*  
                  integer          *numbertype(\*)*

The equivalent *FORTRAN* code for the example above is:

```
nfld = gdinqflds(gridID, fieldlist, rank, numbertype)
```

The parameter, *fieldlist*, will have the value: "*Spectra,Temperature*"

with *rank[2]={3,2}*, *numbertype[2]={1,1}*

# Retrieve Grid Structures Defined in HDF-EOS File

---

## GDinqgrid

int32\_t GDinqgrid(char \* *filename*, char \* *gridlist*, int32\_t \* *strbufsize*)

<i>filename</i>	IN: HDF-EOS filename
<i>gridlist</i>	OUT: Grid list (entries separated by commas)
<i>strbufsize</i>	OUT: String length of grid list
Purpose	Retrieves number and names of grids defined in HDF-EOS file.
Return value	Number of grids found or FAIL (-1) otherwise.
Description	The grid list is returned as a string with each grid name separated by commas. If <i>gridlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . If <i>strbufsize</i> is also set to NULL, the routine returns just the number of grids. Note that <i>strbufsize</i> does not count the null string terminator.
Example	In this example, we retrieve information about the grids defined in an HDF-EOS file, <i>HDFEOS.h5</i> . We assume that there are two grids stored, <i>GridOne</i> and <i>Grid_2</i> :

```
ngrid = GDinqgrid("HDFEOS.h5", NULL, strbufsize);
```

The parameter, *ngrid*, will have the value 2 and *strbufsize* will have value 16.

```
ngrid = GDinqgrid("HDFEOS.h5", gridlist, strbufsize);
```

The variable, *gridlist*, will be set to:

“*GridOne,Grid\_2*”.

**FORTRAN** integer\*4 function gdinqgrid(*filename*,*gridlist*,*strbufsize*)  
character\*(\*) *filename*  
character\*(\*) *gridlist*  
integer\*4      *strbufsize*

The equivalent *FORTRAN* code for the example above is:

```
ngrid = gdinqgrid('HDFEOS.h5', gridlist, strbufsize)
```

# Return Number of Specified Objects in a Grid

---

## GDnentries

`int32_t GDnentries(hid_t gridID, int32_t entrycode, int32_t *strbufsize)`

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>entrycode</i>	IN: Entrycode
<i>strbufsize</i>	OUT: String buffer size
Purpose	Returns number of entries and descriptive string buffer size for a specified entity.

Return value Number of entries if successful or FAIL(-1) otherwise. A typical reason for failure is an improper grid id or entry code.

Description This routine can be called before using the inquiry routines in order to determine the sizes of the output arrays and descriptive strings. The string length does not include the NULL terminator.

The entry codes are: HDFE\_NENTDIM (0) - Dimensions

HDFE\_NENTDFLD (4) - Data Fields

Example In this example, we determine the number of data field entries and the size of the field list string.

```
ndims = GDnentries(gridID, HDFE_NENTDFLD, &bufsz);
```

**FORTRAN** integer\*4 function gdentries(gridid,entrycode,bufsize)  
integer gridid  
integer\*4 entrycode  
integer\*4 bufsize

The equivalent *FORTRAN* code for the example above is:

```
ndims = gdentries(gridid, 4, bufsize)
```

# Open HDF-EOS File

---

## GDopen

hid\_t GDopen(char \*filename, uintn access)

<i>filename</i>	IN: Complete path and filename for the file to be opened
<i>access</i>	IN: H5F_ACC_RDONLY, H5F_ACC_RDWR, H5F_ACC_TRUNC, H5F_ACC_EXCL, H5F_ACC_DEBUG, H5P_DEFAULT
Purpose	Opens or creates HDF file in order to create, read, or write a grid.
Return value	Returns the grid file id handle(fid) if successful or FAIL(-1) otherwise.
Description	This routine creates a new file or opens an existing one, depending on the access parameter.  Access codes:  H5F_ACC_RDONLY Open for read only. If file does not exist, error H5F_ACC_RDWR Open for read/write. If file does not exist, error H5F_ACC_TRUNC If file exists, delete it, then open a new file for read/write  H5F_ACC_EXCL Fail if file already exists  H5F_ACC_DEBUG Print debug information  H5P_DEFAULT Apply default file access and creation properties
Example	In this example, we create a new grid file named, <i>GridFile.h5</i> . It returns the file handle, <i>fid</i> .

FORTRAN

```
fid = GDopen("GridFile.h5", H5F_ACC_TRUNC);
```

```
integer function gdopen(filename, access)
character*(*) filename
integer access
```

The access codes should be defined as parameters:

```
parameter (DFACC_RDONLY=0)
parameter (DFACC_RDWR=1)
parameter (DFACC_TRUNC=2)
```

The equivalent *FORTRAN* code for the example above is:

```
fid = gdopen("GridFile.h5", DFACC_TRUNC)
```

# Return Grid Origin Information

---

## GDorigininfo

herr\_t GDorigininfo(hid\_t *gridID*, int32\_t \**origincode*)

*gridID* IN: Grid id returned by GDcreate or GDattach

*origincode* IN: Origin code

Purpose Retrieve origin code.

Return value Origin code if successful or FAIL (-1) otherwise.

Description This routine retrieves the origin code.

Example In this example, we retrieve the origin code defined in *GDdeforigin*.

```
status = GDorigininfo(gridID, &origincode);
```

The return value, *origincode*, will be equal to 3

FORTRAN integer function gdorginfo(*gridid,origincode*)

```
integer      gridid
```

```
integer*4    origincode
```

The equivalent *FORTRAN* code for the above example is :

```
status = gdorginfo(gridid, origincode)
```

# Return Pixel Registration Information

---

## GDpixreginfo

herr\_t GDpixreginfo(hid\_t *gridID*, int32\_t \**pixregcode*)

*gridID* IN: Grid id returned by GDcreate or GDattach

*pixregcode* IN: Pixel registration code

Purpose Retrieve pixel registration code.

Return value Pixel registration code if successful or FAIL (-1) otherwise.

Description This routine retrieves the pixel registration code.

Example In this example, we retrieve the pixel registration code defined in *GDdefpixreg*.

```
status = GDpixreginfo(gridID, &pixregcode);
```

The return value, *pixregcode*, will be equal to 1

**FORTRAN** integer function gdpreginfo(*gridid,pixregcode*)

```
integer      gridid
```

```
integer*4    pixregcode
```

The equivalent *FORTRAN* code for the above example is :

```
status = gdpreginfo(gridid, pixregcode)
```

# Retrieve Grid Projection Information

---

## GDprojinfo

```
herr_t GDprojinfo(hid_t gridID, int32_t *projcode, int32_t *zonecode, int32_t  
*spheredcode, float64 projparm[])
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>projcode</i>	OUT: GCTP projection code
<i>zonecode</i>	OUT: GCTP zone code used by UTM projection
<i>spheredcode</i>	OUT: GCTP spheroid code
<i>projparm</i>	OUT: GCTP projection parameter array
Purpose	Retrieves projection information of grid
Return Value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise
Description	Retrieves the GCTP projection code, zone code, spheroid code and the projection parameters of the grid
Example	In this example, we are retrieving the projection information from a grid attached to with GDAttached:

```
status = GDprojinfo(gridID, &projcode, &zonecode,  
&spheredcode, projparm);
```

**FORTRAN**

```
integer function gdprojinfo( gridid, projcode, zonecode, spheredcode,  
projparm)  
  
integer      gridid  
integer*4    projcode  
integer*4    zonecode  
integer*4    spheredcode  
real*8      projparm(*)
```

The equivalent FORTRAN code for the example above is:

```
status = gdprojinfo(gridid, projcode, zonecode, spheredcode,  
projparm)
```

# Read Grid Attribute

---

## GDreadattr

herr\_t GDreadattr(hid\_t *gridID*, char \**attrname*, void \**datbuf*)

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>attrname</i>	IN: Attribute name
<i>datbuf</i>	OUT: Buffer allocated to hold attribute values
Purpose	Reads attribute from a grid.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are an improper grid id or number type or incorrect attribute name.
Description	The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.
Example	In this example, we read a single precision (32 bit) floating point attribute with the name "ScalarFloat":

```
status = GDreadattr(gridID, "ScalarFloat", &f32);
```

**FORTRAN**      integer function gdrrdattr(*gridid*, *attrname*,*datbuf*)  
                  integer       *gridid*  
                  character\*(\*) *attrname*  
                  <valid type> *datbuf*(\*)

The equivalent *FORTRAN* code for the example above is:

```
status = gdrrdattr(gridid, "ScalarFloat", f32)
```

# Read Data From a Grid Field

---

## GDreadfield

```
herr_t GDreadfield(hid_t gridID, char fieldname, const hsize_t start[], const  
hsize_t stride[], const hsize_t edge[], void *buffer)
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>fieldname</i>	IN: Name of field to read
<i>start</i>	IN: Array specifying the starting location within each dimension
<i>stride</i>	IN: Array specifying the number of values to skip along each dimension
<i>edge</i>	IN: Array specifying the number of values to write along each dimension
<i>buffer</i>	IN: Buffer to store the data read from the field
Purpose	Reads data from a grid field.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are improper grid id or unknown fieldname.
Description	The values within <i>start</i> , <i>stride</i> , and <i>edge</i> arrays refer to the grid field (input) dimensions. The output data in <i>buffer</i> is written to contiguously. The default value for <i>stride</i> is 1 and is used if this parameter is set to <i>NULL</i> .
Example	In this example, we read data from the 10th row (0-based) of the <i>Temperature</i> field.

```
float32 row[120];  
  
hsize_t start[2]={10,1};  
  
hsize_t edge[2]={1,120};  
  
status = GDreadfield(gridID, "Temperature", start, NULL,  
edge, row);
```

```

FORTRAN integer function

gdrdfld(gridid,fieldname,start,stride,edge,buffer)

integer      gridid
character(*) fieldname
integer*4     start(*)
integer*4     stride(*)
integer*4     edge(*)
<valid type> buffer(*)

```

The *start*, *stride*, and *edge* arrays must be defined explicitly, with the *start* array being 0-based.

The equivalent *FORTRAN* code for the example above is:

```

real*4 row(2000)

integer*4 start(2), stride(2), edge(2)

start(1) = 10
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 2000
edge(2) = 1

status = gdrdfld(gridid, "Temperature", start, stride, edge,
row);

```

# Return Information About a Region

---

## GDregioninfo

```
herr_t GDregioninfo(hid_t gridID, hid_t regionID, char *fieldname,  
                    H5T_class_t *ntype, int32_t *rank, int32_t dims[],  
                    int32_t *size, float64 upleftpt[], float64 lowrightpt[])
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>regionID</i>	IN: Region (period) id returned by GDdefboxregion (GDdeftimeperiod)
<i>fieldname</i>	IN: Field to subset
<i>ntype</i>	OUT: Data type class ID of field
<i>rank</i>	OUT: Rank of field
<i>dims</i>	OUT: Dimensions of subset region
<i>size</i>	OUT: Size in bytes of subset region
<i>upleftpt</i>	OUT: Upper left point of subset region
<i>lowrightpt</i>	OUT: Lower right point of subset region
Purpose	Retrieves information about the subsetted region.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise.
Description	This routine returns information about a subsetted region for a particular field. It is useful when allocating space for a data buffer for the region. Because of differences in number type and geolocation mapping, a given region will give different values for the dimensions and size for various fields. The <i>upleftpt</i> and <i>lowrightpt</i> arrays can be used when creating a new grid from the subsetted region.
Example	In this example, we retrieve information about the region defined in <i>GDdefboxregion</i> for the <i>Temperature</i> field. We use this to allocate space for data in the subsetted region.

```
status = GDregioninfo(GDid, regionID, "Temperature", &ntype,  
                      &rank, dims, &size, upleft, lowright);
```

**FORTRAN**    integer function gdreginfo(*gridid*, *regionid*, *fieldname*, *ntype*, *rank*, *dims*,  
              *size*, *upleftpt*, *lowrightpt*)  
              integer        *gridid*  
              integer        *regionid*  
              character\*(\*) *fieldname*  
              integer        *ntype*  
              integer\*4      *rank*  
              integer\*4      *dims(\*)*  
              integer\*4      *size*  
              real\*8        *upleftpt(\*)*  
              real\*8        *lowrightpt(\*)*

The equivalent *FORTRAN* code for the example above is:

```
status = gdreginfo(gridid, regid, "Spectra", ntype, rank,  
 dims, size, upleftpt, lowrightpt)
```

# Set Fill Value for a Specified Field

---

## GDsetfillvalue

herr\_t GDsetfillvalue(hid\_t *gridID*, char \**fieldname*, void \**fillvalue*)

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>fieldname</i>	IN: Fieldname
<i>fillvalue</i>	IN: Pointer to the fill value to be used
Purpose	Sets fill value for the specified field.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are an improper grid id or number type.
Description	The fill value is placed in all elements of the field which have not been explicitly defined.
Example	In this example, we set a fill value for the "Temperature" field:

```
tempfill = -999.0;  
  
status = GDsetfillvalue(gridID, "Temperature", &tempfill);
```

**FORTRAN** integer function  
*gdsetfill*(*gridid,fieldname,fillvalue*)  
integer *gridid*  
character(\*) *fieldname*  
<valid type> *fillvalue(\*)*

The equivalent *FORTRAN* code for the example above is:

```
status = gdsetfill(gridid, "Temperature", -999.0)
```

# Write/Update Grid Attribute

---

## GDwriteattr

```
herr_t GDwriteattr(hid_t gridID, char *attrname, hid_t ntype, hsize_t count[],  
                    void *datbuf)
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>attrname</i>	IN: Attribute name
<i>ntype</i>	IN: Data type of attribute
<i>count</i>	IN: Number of values to store in attribute
<i>datbuf</i>	IN: Attribute values
Purpose	Writes/Updates attribute in a grid.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are an improper grid id or number type.
Description	If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call.

Example      In this example, we write a single precision (32 bit) floating point number with the name "ScalarFloat" and the value 3.14:

```
f32 = 3.14;  
status = GDwriteattr(gridid, "ScalarFloat",  
H5T_NATIVE_FLOAT, 1,&f32);
```

We can update this value by simply calling the routine again with the new value:

```
f32 = 3.14159;  
status = GDwriteattr(gridid, "ScalarFloat",  
H5T_NATIVE_FLOAT, 1, &f32);
```

```
FORTRAN integer function gdwrattr(gridid, attrname,  
          ntype, count, datbuf)  
        integer      gridid  
        character(*) attrname  
        integer      ntype  
        integer*4     count(*)  
        <valid type> datbuf(*)
```

The equivalent *FORTRAN* code for the first example above is:

```
parameter (DFNT_FLOAT32=5)  
count(1) = 1  
f32 = 3.14  
status = gdwrattr(gridid, "ScalarFloat", DFNT_FLOAT32,  
count, f32)
```

# Write Data to a Grid Field

---

## GDwritefield

```
herr_t GDwritefield(hid_t gridID, char *fieldname, const hssize_t start[], const hsize_t stride[], const hsize_t edge[], void *data)
```

<i>gridID</i>	IN: Grid id returned by GDcreate or GDattach
<i>fieldname</i>	IN: Name of field to write
<i>start</i>	IN: Array specifying the starting location within each dimension (0-based)
<i>stride</i>	IN: Array specifying the number of values to skip along each dimension
<i>edge</i>	IN: Array specifying the number of values to write along each dimension
<i>data</i>	IN: Values to be written to the field
Purpose	Writes data to a grid field.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description	The values within <i>start</i> , <i>stride</i> , and <i>edge</i> arrays refer to the grid field (output) dimensions. The input data in the <i>data</i> buffer is read from contiguously. The default value for <i>stride</i> is 1 and is used if this parameter is set to <i>NULL</i> . Note that the data buffer for a compressed field must be the size of the entire field as incremental writes are not supported by the underlying HDF routines. If this is not possible due to, for example, memory limitations, then the user should consider tiling. See <i>GDdeftile</i> for further information.

Example In this example, we write data to the *Temperature* field.

```
float32 temperature [200][120];
hssize start[2]={0,0};
hsize edge[2] = {200,120};

/* Define elements of temperature array */
status = GDwritefield(gridID, "Temperature", start, NULL,
                      edge, temperature);
```

We now update Row 10 (0 - based) in this field:

```
float32 newrow[2000];
start[2]={0,10};
edge[2]={2000,1};
/* Define elements of newrow array */
```

```
status = GDwritefield(gridID, "Temperature", start, NULL,
                      edge, newrow);
```

**FORTRAN** integer function

```
gdwrfld(gridid,fieldname,start,stride,edge,data)
integer      gridid
character(*) fieldname
integer*4    start(*)
integer*4    stride(*)
integer*4    edge(*)
<valid type> data(*)
```

The *start*, *stride*, and *edge* arrays must be defined explicitly, with the *start* array being 0-based.

The equivalent *FORTRAN* code for the example above is:

```
real*4 temperature(2000,1000)
integer*4 start(2), stride(2), edge(2)
start(1) = 0
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 2000
edge(2) = 1000
status = gdwrfld(gridid, "Temperature", start, stride, edge,
                  temperature)
```

We now update Row 10 (0 - based) in this field:

```
real*4 newrow(2000)
integer*4 start(2), stride(2), edge(2)
start(1) = 10
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 2000
edge(2) = 1
status = gdwrfld(gridid, "Temperature", start, stride, edge,
                  newrow)
```

### **2.1.3 HDF-EOS Utility Routines**

This section contains an alphabetical list of the utility functions. The functions are alphabetized on their C-language names.

# Convert Among Angular Units

---

## EHconvAng

float64 EHconvAng(float64 *inAngle*, intn *code*)

<i>inAngle</i>	IN: Input angle
<i>code</i>	IN: Conversion code
Purpose	Convert among various angular units.
Return value	Returns angle in desired units if successful or FAIL (-1) otherwise.
Description	This routine converts angles between three units, decimal degrees, radians, and packed degrees-minutes-seconds. In the later unit, an angle is expressed as a integral number of degrees and minutes and a float point value of seconds packed as a single float64 number as follows: DDDMMMSSS.SS. The six conversion codes are: HDFE_RAD_DEG (0), HDFE_DEG_RAD (1), HDFE_DMS_DEG (2), HDFE_DEG_DMS (3), HDFE_RAD_DMS (0), and HDFE_DMS_RAD (1), where the first three letter code (RAD - radians, DEG - decimal degrees, DMS - packed degrees-minutes-seconds) corresponds to the input angle and the second to the desired output angular unit.

Example To convert 27.5 degrees to packed format:

```
inAng = 27.5;  
  
outAng = EHconvAng(inAng, HDFE_DEG_DMS);  
“outAng” will contain the value: 27030000.00.
```

FORTRAN real\*8 function ehconvang(*inangle,code*)  
real\*8       *inangle*  
integer       *code*

The equivalent *FORTRAN* code for the example above is:

```
inangle = 27.5  
outangle = ehconvang(inangle,3)
```

# Get HDF-EOS Version String

---

## EHgetversion

herr\_t EHgetversion(hid\_t *fid*, char \**version*)

<i>fid</i>	IN: File id returned by <i>SWopen</i> , <i>GDopen</i> , or <i>PTopen</i> .
<i>version</i>	OUT: HDF-EOS version string
Purpose	Get HDF-EOS version string.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description	This routine returns the HDF-EOS version string of an HDF-EOS file. This designates the version of HFD-EOS that was used to create the file. This string is of the form: “HDFEOS_V <i>maj.min</i> ” where <i>maj</i> is the major version and <i>min</i> is the minor version.

Example To get the HDF-EOS version (assumed to be 2.0) used to create the HDF-EOS file: “SwathFile.h5”:

```
char version[16];  
  
fid = SWopen("SwathFile.h5", H5F_ACC_RDONLY);  
  
status = EHgetversion(fid, version);  
  
“version” will contain the string: “HDFEOS_3.alpha”.
```

FORTRAN integer function ehgetver(*fid*,*version*)

```
integer      fid  
character*(*) version
```

The equivalent *FORTRAN* code for the example above is:

```
character*16 version  
fid = swopen("SwathFile.h5",1)  
status = ehgetver(fid, version)
```

# Get HDF File ids

---

## EHidinfo

herr\_t EHidinfo(hid\_t *fid*, hid\_t \**HDFfid*, hid\_t \**groupID*)

<i>fid</i>	IN: File id returned by <i>SWopen</i> , <i>GDopen</i> , or <i>PTopen</i> .
<i>HDFfid</i>	OUT: HDF file ID (returned by <i>EHopen</i> )
<i>groupID</i>	OUT: HDF group ID (returned by <i>XXcreate</i> )
Purpose	Get HDF file IDs.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description	This routine returns the HDF file ids corresponding to the HDF-EOS file id returned by <i>SWopen</i> , <i>GDopen</i> , or <i>PTopen</i> . These ids can then be used to create or access native HDF structure such as groups, datasets, or HDF attributes within an HDF-EOS file.

### Example

FORTRAN    integer function ehidinfo(*fid,hdffid,sdid*)  
              integer        *fid*  
              integer        *hdffid*  
              integer        *grid*

To retrieve the HDF file id and Group interface id:

```
fid = swopen("SwathFile.h5",1)
status = ehidinfo(fid, hdffid, grid)
```

## Abbreviations and Acronyms

---

AI&T	Algorithm Integration & Test
AIRS	Atmospheric Infrared Sounder
API	application program interface
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer
CCSDS	Consultative Committee on Space Data Systems
CDRL	Contract Data Requirements List
CDS	CCSDS day segmented time code
CERES	Clouds and Earth Radiant Energy System
CM	configuration management
COTS	commercial off-the-shelf software
CUC	constant and unit conversions
CUC	CCSDS unsegmented time code
DAAC	distributed active archive center
DBMS	database management system
DCE	distributed computing environment
DCW	Digital Chart of the World
DEM	digital elevation model
DTM	digital terrain model
ECR	Earth centered rotating
ECS	EOSDIS Core System
EDC	Earth Resources Observation Systems (EROS) Data Center
EDHS	ECS Data Handling System
EDOS	EOSDIS Data and Operations System
EOS	Earth Observing System
EOSAM	EOS AM Project (morning spacecraft series)
EOSDIS	Earth Observing System Data and Information System

EOSPM	EOS PM Project (afternoon spacecraft series)
ESDIS	Earth Science Data and Information System (GSFC Code 505)
FDF	flight dynamics facility
FOV	field of view
ftp	file transfer protocol
GCT	geo-coordinate transformation
GCTP	general cartographic transformation package
GD	grid
GPS	Global Positioning System
GSFC	Goddard Space Flight Center
HDF	hierarchical data format
HITC	Hughes Information Technology Corporation
http	hypertext transport protocol
I&T	integration & test
ICD	interface control document
IDL	interactive data language
IP	Internet protocol
IWG	Investigator Working Group
JPL	Jet Propulsion Laboratory
LaRC	Langley Research Center
LIS	Lightening Imaging Sensor
M&O	maintenance and operations
MCF	metadata configuration file
MET	metadata
MODIS	Moderate-Resolution Imaging Spectroradiometer
MSFC	Marshall Space Flight Center
NASA	National Aeronautics and Space Administration
NCSA	National Center for Supercomputer Applications
netCDF	network common data format

NGDC	National Geophysical Data Center
NMC	National Meteorological Center (NOAA)
ODL	object description language
PC	process control
PCF	process control file
PDPS	planning & data production system
PGE	product generation executive (formerly product generation executable)
POSIX	Portable Operating System Interface for Computer Environments
PT	point
QA	quality assurance
RDBMS	relational data base management system
RPC	remote procedure call
RRDB	recommended requirements database
SCF	Science Computing Facility
SDP	science data production
SDPF	science data processing facility
SGI	Silicon Graphics Incorporated
SMF	status message file
SMP	Symmetric Multi–Processing
SOM	Space Oblique Mercator
SPSO	Science Processing Support Office
SSM/I	Special Sensor for Microwave/Imaging
SW	swath
TAI	International Atomic Time
TBD	to be determined
TDRSS	Tracking and Data Relay Satellite System
TRMM	Tropical Rainfall Measuring Mission (joint US – Japan)
UARS	Upper Atmosphere Research Satellite
UCAR	University Corporation for Atmospheric Research

URL	universal reference locator
USNO	United States Naval Observatory
UT	universal time
UTC	Coordinated Universal Time
UTCF	universal time correlation factor
UTM	universal transverse mercator
VPF	vector product format
WWW	World Wide Web