

170-TP-006-002

HDF-EOS Library Users Guide for the ECS Project Volume 2: Function Reference Guide

Technical Paper

April 1997

Prepared Under Contract NAS5-60000

SUBMITTED BY

<u>Paul Fingerman /s/</u>	<u>4/30/97</u>
Paul Fingerman, ECS CCB Chairman	Date
EOSDIS Core System Project	

Hughes Information Technology Systems
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. HDF is the scientific data format standard selected by NASA as the baseline standard for EOS. This Users Guide accompanies Version 1 software, which is available to the user community on the EDHS1 server. These 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 grid, point or swath structures, 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, May, 1997, 333-CD-003-001).

HDF-EOS is an extension of NCSA (National Center for Supercomputing Applications) HDF and uses HDF library calls as an underlying basis. Version 4.1 r1 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, Point 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
Joel Gales, jgales@eos.hitc.com
David Wynne, davidw@eos.hitc.com
Doug Ilg, dilg@gsfc.nasa.gov

An email address has been provided for user help:
pgstlkit@eos.hitc.com

Any questions should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Systems
1616 McCormick Drive
Upper Marlboro, MD 20774-5372

This page intentionally left blank.

Abstract

This document will serve as the user's guide to the HDF-EOS file access library. 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 three interfaces included in HDF-EOS - Point, 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, Metadata, Standard Data Format, Standard Data Product, Disk Format, Point, Grid, Swath, Projection, Array, Browse

This page intentionally left blank

Change Information Page

List of Effective Pages			
Page Number		Issue	
Title iii through x 1-1 and 1-2 2-1 and 2-162 AB-1 and AB-4		Submitted as Final Submitted as Final Submitted as Final Submitted as Final Submitted as Final	
Document History			
Document Number	Status/Issue	Publication Date	CCR Number
170-TP-006-002	Submitted as Final	April 1997	97-0659

This page intentionally left blank.

Contents

Preface

Abstract

1. Introduction

1.1 Purpose	1-1
1.2 Organization	1-1
1.3 Point Data.....	1-2
1.3.1 PT API Routines.....	1-2
1.3.2 List of PT API Routines	1-2
1.4 Swath Data	1-4
1.4.1 The Swath Data Interface.....	1-4
1.4.2 List of SW API Routines.....	1-4
1.5 Grid Data.....	1-6
1.5.1 The Grid Data Interface.....	1-6
1.5.2 List of Grid API ROUTINES.....	1-6
1.6 GCTP Usage.....	1-8
1.6.1 GCTP Projection Codes	1-8
1.6.2 UTM Zone Codes.....	1-9
1.6.3 GCTP Spheroid Codes	1-9
1.6.4 GCTP Projection Parameters.....	1-10

2. Function Reference

2.1 Format	2-1
2.1.1 Point Interface Functions.....	2-1

List of Tables

1-3. Summary of the Point Interface	1-3
1-4. Summary of the Swath Interface.....	1-5
1-5. Summary of the Grid Interface	1-7
1-6. Projection Transformation Package Projection Parameters.....	1-10

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 Point Data

The PT (*Point*) interface consists of routines for storing, retrieving, and manipulating data in point data sets. This interface is designed to support data that has associated geolocation information, but is not organized in any well defined spatial or temporal way. See the Users' Guide, Volume 1 that accompanies this document for more information.

1.3.1 PT API Routines

All C routine names in the point data interface have the prefix "PT" and the equivalent FORTRAN routine names are prefixed by "pt." The PT routines are classified into the following categories:

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

1.3.2 List of PT API Routines

The PT 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 Point Interface

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Access	PTopen	ptopen	creates a new file or opens an existing one	2-26
	Ptcreate	ptcreate	creates a new point data set and returns a handle	2-6
	PTattach	ptattach	attaches to an existing point data set	2-2
	PTdetach	ptdetach	releases a point data set and frees memory	2-12
	PTclose	ptclose	closes the HDF-EOS file and deactivates the point interface	2-5
Definition	PTdeflevel	ptdeflev	defines a level within the point data set	2-8
	PTdeflinkage	ptdeflink	defines link field to use between two levels	2-10
Basic I/O	PTwritelevel	ptwrlev	writes (appends) full records to a level	2-38
	PTreadlevel	ptrdlev	reads data from the specified fields and records of a level	2-30
	PTupdatelevel	ptuplev	updates the specified fields and records of a level	2-34
	PTwriteattr	ptwrattr	creates or updates an attribute of the point data set	2-36
	PTreadattr	ptrdatr	reads existing attribute of point data set	2-29
Inquiry	PTnlevels	ptnlevs	returns the number of levels in a point data set	2-24
	PTnrecs	ptnrecs	returns the number of records in a level	2-25
	PTnfields	ptnfls	returns number of fields defined in a level	2-23
	PTlevelinfo	ptnlevinfo	returns information about a given level	2-22
	PTlevelindx	ptlevidx	returns index number for a named level	2-21
	PTbcklinkinfo	ptbclinkinfo	returns link field to previous level	2-4
	PTfwdlinkinfo	ptflinkinfo	returns link field to following level	2-15
	PTgetlevelname	ptgetlevname	returns level name given level number	2-16
	PTsizeof	ptsizeof	returns size in bytes for specified fields in a point	2-33
	PTattrinfo	ptattrinfo	returns information about point attributes	2-3
	PTingattrs	ptingattrs	retrieves number and names of attributes defined	2-19
PTingpoint	ptingpoint	retrieves number and names of points in file	2-20	
Utility	PTgetrecnums	ptgetrecnums	returns corresponding record numbers in a related level	2-17
Subset	PTdefboxregion	ptdefboxreg	define region of interest by latitude/longitude	2-7
	PTregioninfo	ptreginfo	returns information about defined region	2-31
	PTregionrecs	ptregrecs	returns # of records and record #s within region	2-32
	PTextractregion	ptextreg	read a region of interest from a set of fields in a single level	2-14
	PTdeftimeperiod	ptdeftmper	define time period of interest	2-11
	PTperiodinfo	ptperinfo	returns information about defined time period	2-27
	PTperiodrecs	ptperrecs	returns # of records and record #s within time period	2-28
	PTextractperiod	ptextper	read a time period from a set of fields in a single level	2-13

1.4 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.

1.4.1 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.

1.4.2 List of SW API Routines

The SW 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 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	2-79
	SWcreate	swcreate	creates a swath within the file	2-43
	SWattach	swattach	attaches to an existing swath within the file	2-39
	SWdetach	swdetach	detaches from swath interface	2-61
	SWclose	swclose	closes file	2-41
Definition	SWdefdim	swdefdim	defines a new dimension within the swath	2-50
	SWdefdimmap	swdefmap	defines the mapping between the geolocation and data dimensions	2-51
	SWdefidxmap	swdefimap	defines a non-regular mapping between the geolocation and data dimension	2-55
	SWdefgeofield	swdefgfld	defines a new geolocation field within the swath	2-53
	SWdefdatafield	swdefdfd	defines a new data field within the swath	2-48
	SWdefcomp	swdefcomp	defines a field compression scheme	2-46
	SWwritegeometa	swwrgmeta	writes field metadata for an existing swath geolocation field	2-94
SWwritedatameta	swwrdmeta	writes field metadata for an existing swath data field	2-90	
Basic I/O	SWwritefield	swwrfld	writes data to a swath field	2-91
	SWreadfield	swrdfld	reads data from a swath field.	2-83
	SWwriteattr	swwrattr	writes/updates attribute in a swath	2-88
	SWreadattr	swrdattr	reads attribute from a swath	2-82
	SWsetfillvalue	swsetfill	sets fill value for the specified field	2-87
	SWgetfillvalue	swgetfill	retrieves fill value for the specified field	2-68
Inquiry	SWinqdims	swinqdims	retrieves information about dimensions defined in swath	2-72
	SWinqmaps	swinqmaps	retrieves information about the geolocation relations defined	2-75
	SWinqidxmaps	swinqimaps	retrieves information about the indexed geolocation/data mappings defined	2-74
	SWinqgeofields	swinqgflds	retrieves information about the geolocation fields defined	2-73
	SWinqdatafields	swinqdfd	retrieves information about the data fields defined	2-71
	SWinqattrs	swinqattr	retrieves number and names of attributes defined	2-70
	SWnentries	swnentries	returns number of entries and descriptive string buffer size for a specified entity	2-78
	SWdiminfo	swdiminfo	retrieve size of specified dimension	2-62
	SWmapinfo	swmapinfo	retrieve offset and increment of specified geolocation mapping	2-77
	SWidxmapinfo	swimapinfo	retrieve offset and increment of specified geolocation mapping	2-69
	SWattrinfo	swattrinfo	returns information about swath attributes	2-40
	SWfieldinfo	swfldinfo	retrieve information about a specific geolocation or data field	2-66
	SWcompinfo	swcompinfo	retrieve compression information about a field	2-42
SWinqswath	swinqswath	retrieves number and names of swaths in file	2-76	

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

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Subset	SWdefboxregion	swdefboxreg	define region of interest by latitude/longitude	2-44
	SWregioninfo	swreginfo	returns information about defined region	2-85
	SWextractregion	swextreg	read a region of interest from a field	2-65
	SWdeftimeperiod	swdeftmep	define a time period of interest	2-56
	SWperiodinfo	swperinfo	returns information about a defined time period	2-80
	SWextractperiod	swextper	extract a defined time period	2-64
	SWdefvrtregion	swdefvrtreg	define a region of interest by vertical field	2-58
	SWdupregion	swdupreg	duplicate a region or time period	2-63

1.5 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.

1.5.1 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.

1.5.2 List of Grid API ROUTINES

The GD function calls are listed below in Table 1-5 and are described in detail in Section 2 of this document. The listing in Section 2 is in alphabetical order.

Table 1-5. 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	2-139
	GDcreate	gdcreate	creates a new grid in the file	2-100
	GDattach	gdattach	attaches to a grid	2-96
	GDdetach	gddetach	detaches from grid interface	2-120
	GDclose	gdclose	closes file	2-98
Definition	GDdeforigin	gddeforigin	defines origin of grid	2-109
	GDdefdim	gddefdim	defines dimensions for a grid	2-106
	GDdefproj	gddefproj	defines projection of grid	2-111
	GDdefpixreg	gddefpixreg	defines pixel registration within grid cell	2-110
	GDdeffield	gddeffld	defines data fields to be stored in a grid	2-107
	GDdefcomp	gddefcomp	defines a field compression scheme	2-104
	GDwritefieldmeta	gdwrmeta	writes metadata for field already existing in file	2-156
Basic I/O	GDwritefield	gdwrfld	writes data to a grid field.	2-154
	GDreadfield	gdrfld	reads data from a grid field	2-144
	GDwriteattr	gdwrattr	writes/updates attribute in a grid.	2-152
	GDreadattr	gdrattr	reads attribute from a grid	2-143
	GDsetfillvalue	gdsetfill	sets fill value for the specified field	2-149
	GDgetfillvalue	gdgetfill	retrieves fill value for the specified field	2-126
	Inquiry	GDinqdims	gdinqdims	retrieves information about dimensions defined in grid
GDinqfields		gdinqflds	retrieves information about the data fields defined in grid	2-134
GDinqattrs		gdinqattrs	retrieves number and names of attributes defined	2-132
GDnentries		gdnentries	returns number of entries and descriptive string buffer size for a specified entity	2-138
GDgridinfo		gdgridinfo	returns dimensions of grid and X-Y coordinates of corners	2-131
GDprojinfo		gdprojinfo	returns all GCTP projection information	2-142
GDdiminfo		gddiminfo	retrieves size of specified dimension.	2-121
GDcompinfo		gdcompinfo	retrieve compression information about a field	2-99
GDfieldinfo		gdfldinfo	retrieves information about a specific geolocation or data field in the grid	2-124
GDinggrid		gdinqgrid	retrieves number and names of grids in file	2-135
GDattrinfo		gdattrinfo	returns information about grid attributes	2-97
GDorigininfo		gdorginfo	return information about grid origin	2-140
GDpixreginfo		gdpreinfo	return pixel registration information for given grid	2-141
Subset	GDdefboxregion	gddefboxreg	define region of interest by latitude/longitude	2-103
	GDregioninfo	gdreginfo	returns information about a defined region	2-147
	GDextractregion	gdextreg	read a region of interest from a field	2-123
	GDdeftimeperiod	gddeftmeper	define a time period of interest	2-115
	GDdefvrtregion	gddefvrtreg	define a region of interest by vertical field	2-117
	GDgetpixels	gdgetpix	get row/columns for lon/lat pairs	2-127
	GDgetpixvalues	gdgetpixval	get field values for specified pixels	2-129
	GDinterpolate	gdinterpolate	perform bilinear interpolation on a grid field	2-136
	GDdupregion	gdidupreg	duplicate a region or time period	2-122

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

Category	Routine Name		Description	Page
	C	FORTRAN		Nos.
Tiling	GDdeftile	gddeftle	define a tiling scheme	2-113
	GDtileinfo	gdtleinfo	returns information about tiling for a field	2-151
	GDsettilecache	gdsettleche	set tiling cache parameters	2-150
	GDreadtile	gdrdtle	read data from a single tile	2-146
	GDwritetile	gdwrtile	write data to a single tile	2-157

1.6 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.

1.6.1 GCTP Projection Codes

The following GCTP projection codes are used in the grid API described in Section 7 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 2 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, May, 1996, (333-CD-003-001).

1.6.2 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

1.6.3 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)
Austrailian National	(14)
Krassovsky	(15)
Hough	(16)
Mercury 1960	(17)
Modified Mercury 1968	(18)
Sphere of Radius 6370997m	(19)

1.6.4 GCTP Projection Parameters

Table 1-6. Projection Transformation Package Projection Parameters (1 of 2)

Code & Projection Id	Array Element							
	1	2	3	4	5	6	7	8
0 Geographic								
1 U T M	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 1-6. Projection Transformation Package Projection Parameters (2 of 2)

Code & Projection Id	Array Element				
	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 (DDDDMMSSS.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 Point Interface Functions

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

Attach to an Existing Point Structure

PTattach

int32 PTattach(int32 *fid*, char **pointname*)

fid IN: Point file id returned by PTopen

pointname IN: Name of point to be attached

Purpose Attaches to an existing point within the file.

Return value Returns the point handle (pointID) if successful and FAIL (-1) otherwise. Typical reasons for failure are an improper point file id or point name.

Description This routine attaches to the point using the *pointname* parameter as the identifier.

Example In this example, we attach to the previously created point, "ExamplePoint", within the HDF file, *PointFile.hdf*, referred to by the handle, *fid*:

```
pointID = PTattach(fid, "ExamplePoint");
```

The point can then be referenced by subsequent routines using the handle, *pointID*.

FORTTRAN integer*4 function ptattach(*fid*,*pointname*)

```
integer*4          fid
```

```
character*(*) pointname
```

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

```
status = ptattach(fid, "ExamplePoint")
```


Return Information About a Point Attribute

PTattrinfo

intn PTattrinfo(int32 *pointID*, char **attrname*, int32 * *numbertype*, int32 **count*)

pointID IN: Point id returned by PTcreate or PTattach

attrname IN: Attribute name

numbertype OUT: Number type of attribute

count OUT: Number of total bytes in attribute

Purpose Returns information about a point attribute

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

Description This routine returns number type and number of elements (*count*) of a point attribute.

Example In this example, we return information about the *ScalarFloat* attribute.

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

The *nt* variable will have the value 5 and *count* will have the value 4.

FORTRAN integer function ptattrinfo(*pointid*, *attrname*, *ntype*, *count*,)
integer*4 *pointid*
character*(*) *attrname*
integer*4 *ntype*
integer*4 *count*

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

```
status = ptattrinfo(pointid, "ScalarFloat",nt,count);
```

Return Linkage Field to Previous Level

PTbcklinkinfo

intn PTbcklinkinfo(int32 *pointID*, int32 *level*, char **linkfield*)

pointID IN: Point id returned by PTcreate or PTattach

level IN: Point level (0-based)

linkfield OUT: Link field

Purpose Returns the linkfield to the previous level.

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

Description This routine returns the linkfield to the previous level.

Example In this example, we return the linkfield connecting the *Observations* level to the previous *Desc-Loc* level. (This levels are defined in the *PTdeflevel* routine.)

```
status = PTbcklinkinfo(pointID2, 1, linkfield);
```

The *linkfield* will contain the string: *ID*.

FORTRAN integer ptblinkinfo(*pointid*, *level*, *linkfield*)

integer*4 *pointid*

integer*4 *level*

character*(*) *linkfield*

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

```
status = ptblinkinfo(pointid2, 0, linkfield)
```

Close an HDF-EOS File

PTclose

intn PTclose(int32 *fid*)

fid IN: Point file id returned by PTopen

Purpose Closes file.

Return value Returns SUCCEEDED (0) if successful and FAIL (-1) otherwise.

Description This routine closes the HDF point file.

Example

```
status = PTclose(fid);
```

FORTTRAN integer ptclose(*fid*)

```
integer*4 fid
```

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

```
status = ptclose(fid)
```

Create a New Point Structure

PTcreate

int32 PTcreate(int32 *fid*, char **pointname*)

fid IN: Point file id returned by PTOpen

pointname IN: Name of point to be created

Purpose Creates a point within the file.

Return value Returns the point handle (*pointID*) if successful and FAIL (-1) otherwise.

Description The point is created as a Vgroup within the HDF file with the name *pointname* and class *POINT*.

Example In this example, we create a new point structure, *ExamplePoint*, in the previously created file, *PointFile.hdf*.

```
pointID = PTcreate(fid, "ExamplePoint");
```

The point structure is then referenced by subsequent routines using the handle, *pointID*.

FORTTRAN integer*4 function ptcreate(*fid*,*pointname*)

```
integer*4    fid
```

```
character*(*) pointname
```

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

```
pointid = ptcreate(fid, "ExamplePoint");
```

```
    Define Region of Interest by Lat/Long.
```

Define Region of Interest by Latitude/Longitude

PTdefboxregion

int32 PTdefboxregion(int32 *pointID*, float64 *cornerlon*[], float64 *cornerlat*[])

pointID IN: Point id returned by PTcreate or PTattach

cornerlon IN: Longitude in decimal degrees of box corners

cornerlat IN: Latitude in decimal degrees of box corners

Purpose Defines a longitude-latitude box region for a point.

Return value Returns the point regionID if successful and FAIL (-1) otherwise.

Description This routine defines an area of interest for a point. It returns a point region ID which is used by the *PTextractregion* routine to read the fields from a level for those records within the area of interest. The point structure must have a level with both a *Longitude* and *Latitude* (or *Colatitude*) field defined

Example In this example, we define an area of interest with (opposite) corners at -145 degrees longitude, -15 degrees latitude and -135 degrees longitude, -8 degrees latitude.

```
cornerlon[0] = -145.;
cornerlat[0] = -15.;
cornerlon[1] = -135.;
cornerlat[1] = -8.;
regionID = PTdefboxregion(pointID, cornerlon, cornerlat);
```

FORTRAN integer*4 function ptdefboxreg(*pointid*, *cornerlon*, *cornerlat*)

```
integer*4    pointid
real*8      cornerlon
real*8      cornerlat
```

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

```
cornerlon(1) = -145.
cornerlat(2) = -15.
cornerlon(1) = -135.
cornerlat(2) = -8.
regionid = ptdefboxreg(pointid, cornerlon, cornerlat)
```

Define a New Level Within a Point

PTdeflevel

intn PTdeflevel(int32 *pointID*, char **levelname*, char **fieldlist*,
int32 *fieldtype*[], int32 *fieldorder*[])

pointID IN: Point id returned by PTcreate or PTattach

levelname IN: Name of level to be defined

fieldlist IN: List of fields in level

fieldtype IN: Array containing field type of each field within level

fieldorder IN: Array containing order of each field within level

Purpose Defines a new level within the point.

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

Description This routine defines a level within the point. A simple point consists of a single level. A point where there is common data for a number of records can be more efficiently stored with multiple levels. The order in which the levels are defined determines the (0-based) level index.

Example Simple Point

In this example, we define a simple single level point, with *levelname*, *Sensor*. The *levelname* should not contain any slashes ("/"). It consists of six fields, *ID*, *Time*, *Longitude*, *Latitude*, *Temperature*, and *Mode* defined in the field list. The *fieldtype* and *fieldorder* parameters are arrays consisting of the HDF number type codes and field orders, respectively. The *Temperature* is an array field of dimension 4 and the *Mode* field a character string of size 4. All other fields are scalars. Note that the order for numerical scalar variables can be either 0 or 1.

```
int32 fieldtype[6] = {DFNT_INT16, DFNT_INT16,  
                    DFNT_FLOAT32,DFNT_FLOAT32, DFNT_FLOAT32,DFNT_CHAR8};  
  
int32 fieldorder[6] = {0,0,0,0,4,4};  
  
char8 *fldlist =  
    "ID,Time,Longitude,Latitude,Temperature,Mode";  
  
status = PTdeflevel(pointID1, "Sensor", fldlist, fieldtype,  
                    fieldorder);
```

Multi-Level Point

In this example, we define a two-level point that describes data from a network of fixed buoys. The first level contains information about each buoy and includes the name (label) of the buoy, its (fixed) longitude and latitude, its deployment date, and an ID that is used to link it to the following level. (The link field is defined in the *PTdeflinkage* routine described later.) The entries within the ID field must be unique. The second level contains the actual measurements from the buoys (rainfall and temperature values) plus the observation time and the ID which relates a given measurement to a particular buoy entry in the previous level. There can be many records in this level with the same ID since there can be multiple measurements from a single buoy. It is advantageous, although not mandatory, to store all records for a particular buoy (ID) contiguously.

Level 0

```
int32 fieldtype0[5] = {DFNT_CHAR8, DFNT_FLOAT64,
                      DFNT_FLOAT64, DFNT_INT32, DFNT_CHAR8};

int32 fieldorder0[5] = {8, 0, 0, 0, 1};

char8 *fldlist0 =
    "Label, Longitude, Latitude, DeployDate, ID";

status = PTdeflevel(pointID2, "Desc-Loc", fldlist0,
                    fieldtype0, fieldorder0);
```

Level 1

```
int32 fieldtype1[4] = {DFNT_FLOAT64, DFNT_FLOAT32,
                      DFNT_FLOAT32, DFNT_CHAR8};

int32 fieldorder1[4] = {0, 0, 0, 1};

char8 *fldlist1 = "Time, Rainfall, Temperature, ID";

status = PTdeflevel(pointID2, "Observations", fldlist1,
                    fieldtype1, fieldorder1);
```

```
FORTRAN integer function ptdeflev(pointid, levelname, fieldlist, fieldtype, fieldorder)
integer*4    pointid
character*(*) levelname
character*(*) fieldlist
integer*4    fieldtype (*)
integer*4    fieldorder (*)
```

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

```
status = ptdeflev(pointid1, "Sensor", fldlist, fieldtype,
                  fieldorder)
```

Define Linkage Field Between Two Levels

PTdeflinkage

intn PTdeflinkage(int32 *pointID*, char **parent*, char **child*, char **linkfield*)

pointID IN: Point id returned by PTcreate or PTattach

parent IN: Name of parent level

child IN: Name of child level

linkfield IN: Name of (common) linkfield

Purpose Defines a linkfield between two (adjacent) levels.

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

Description This routine defines the linkfield between two levels. This field must be defined in both levels.

Example In this example we define the *ID* field as the link between the two levels defined previously in the *PTdeflevel* routine.

```
status = PTdeflinkage(pointID2, "Desc-Loc", "Observations",  
"ID");
```

FORTTRAN integer function ptdeflink(*pointid*, *parent*, *child*, *linkfield*)

```
integer*4    pointid
```

```
character*(*) parent
```

```
character*(*) child
```

```
character*(*) linkfield
```

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

```
status = ptdeflink(pointid2, "Desc-Loc", "Observations",  
"ID")
```


Define Time Period of Interest

PTdeftimeperiod

int32 PTdeftimeperiod(int32 *pointID*, float64 *starttime*, float64 *stoptime*)

pointID IN: Point id returned by PTcreate or PTattach

starttime IN: Start time of period

stoptime IN: Stop time of period

Purpose Defines a time period for a point.

Return value Returns the point periodID if successful and FAIL (-1) otherwise.

Description This routine defines time period for a point. It returns a point period ID which is used by the *PTextractperiod* routine to read the fields from a level for those records within the time period. The point structure must have a level with the *Time* field defined

Example In this example, we define a time period with a start time of 35208757.6 and a stop time of 35984639.2

```
starttime = 35208757.6
```

```
stoptime = 35984639.2
```

```
periodID = PTdeftimeperiod(pointID, starttime, stoptime)
```

FORTRAN integer*4 function swdeftmeper(*pointid*, *starttime*, *stoptime*)

```
integer*4    pointid
```

```
real*8      starttime
```

```
real*8      stoptime
```

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

```
starttime = 35208757.6
```

```
stoptime = 35984639.2
```

```
periodid = ptdeftmeper(pointid, starttime, stoptime)
```

Detach from Point Structure

PTdetach

intn PTdetach(int32 *pointID*)

pointID IN: Point id returned by PTcreate or PTattach

Purpose Detaches from point data set.

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

Description This routine should be run before exiting from the point file for every point opened by PTcreate or PTattach.

Example In this example, we detach the point structure, *ExamplePoint*:

```
status = PTdetach(pointID)
```

FORTTRAN integer ptdetach(*pointid*)

```
integer*4 pointid
```

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

```
status = ptdetach(pointid)
```

Reads Point Records for a Specified Time Period

PTextextractperiod

intn PTextextractperiod(int32 *pointID*, int32 *periodID*, int32 *level*, char **fieldlist*,
VOIDP *buffer*)

pointID IN: Point id

periodID IN: Period id returned by PTdeftimeperiod

level IN: Point level (0-based)

fieldname IN: List of fields to extract

buffer OUT: Data buffer

Purpose Extracts (reads) from subsetted time period.

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

Description This routine reads data from the designated level fields into the data buffer from the subsetted time period.

Example In this example, we read data within the subsetted time period defined in *Ptdeftimeperiod* from the *Time* field.

```
/* Read subsetted data into buffer */  
  
status = PTextextractperiod(pointID, periodID, 0, "Time",  
datbuf);
```

FORTRAN integer function ptextper(*pointid*,*periodid*,*level*,*fieldlist*,*buffer*)

integer*4 *pointid*

integer*4 *periodid*

integer*4 *level*

character*(*) *fieldlist*

<valid type> *buffer*(*)

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

```
status = ptextper(periodid,pointid,0,'Time',datbuf)
```

Reads Point Records for a Specified Geographic Region

PTextextractregion

intn PTextextractregion(int32 *pointID*, int32 *regionID*, int32 *level*, char **fieldlist*,
VOIDP *buffer*)

pointID IN: Point id

regionID IN: Period id returned by PTdefboxregion

level IN: Point level (0-based)

fieldname IN: List of fields to extract

buffer OUT: Data buffer

Purpose Extracts (reads) from subsetted area of interest.

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

Description This routine reads data from the designated level fields into the data buffer from the subsetted area of interest.

Example In this example, we read data within the subsetted area of interest defined in *PTdefboxregion* from the *Longitude* and *Latitude* fields.

```
/* Read subsetted data into buffer */
```

```
status = PTextextractregion(pointID, regionID, 0,  
"Longitude, Latitude", datbuf);
```

FORTRAN integer function ptextreg(*pointid*,*regionid*,*level*,*fieldlist*,*buffer*)

integer*4 *pointid*

integer*4 *regionid*

integer*4 *level*

character*(*) *fieldlist*

<valid type> *buffer*(*)

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

```
status =  
ptextreg(periodid, regionid, 0, "Longitude, Latitude", datbuf)
```

Return Linkage Field to Following Level

PTfwdlinkinfo

intn PTfwdlinkinfo(int32 *pointID*, int32 *level*, char **linkfield*)

pointID IN: Point id returned by PTcreate or PTattach

level IN: Point level (0-based)

linkfield OUT: Link field

Purpose Returns the linkfield to the following level.

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

Description This routine returns the linkfield to the following level.

Example In this example, we return the linkfield connecting the *Desc-Loc* level to the following *Observations* level. (These levels are defined in the *PTdeflevel* routine.)

```
status = PTfwdlinkinfo(pointID2, 1, linkfield);
```

The *linkfield* will contain the string: *ID*.

FORTRAN integer ptflinkinfo(*pointid*, *level*, *linkfield*)

integer*4 *pointid*

integer*4 *level*

character*(*) *linkfield*

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

```
status = ptflinkinfo(pointid2, 0, linkfield)
```

Return Level Name

PTgetlevelname

intn PTgetlevelname(int32 *pointID*, int32 *level*, char **levelname*, int32 **strbufsize*)

pointID IN: Point id returned by PTcreate or PTattach

level IN: Point level (0-based)

linkfield OUT: Level name

strbufsize OUT: String length of level name

Purpose Returns the name of a level given the level number.

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

Description This routine returns the name of a level given the level number (0-based). If the user passes NULL for the *levelname*, the routine will return just the string length of the level name (not counting the null terminator).

Example In this example, we return the level name of the 0th level of the second point defined in the *PTdeflevel* section:

```
status = PTgetlevelname(pointID2, 0, levelname,
&strbufsize);
```

The *levelname* will contain the string: *Desc-Loc* and the *strbufsize* variable will be set to 8.

FORTRAN integer ptgetlevname(*pointid*, *level*, *levelname*, *strbufsize*)

integer*4 *pointid*

integer*4 *level*

character*(*) *levelname*

integer*4 *strbufsize*

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

```
status = ptgetlevname(pointid2, 0, levelname, strbufsize)
```

Return Record Numbers Related to Level

PTgetrecnums

intn PTgetrecnums(int32 *pointID*, int32 *inlevel*, int32 *outlevel*, int32 *inNrec*, int32 *inRecs*[], int32 **outNrec*, int32 *outRecs*[])

pointID IN: Point id returned by PTcreate or PTattach

inlevel IN: Level number of input records(0-based)

outlevel IN: Level number of output records(0-based)

inNrec IN: Number of records in the *inRecs* array

inRecs IN: Array containing the input record numbers.

outNrec OUT: Number of records in the *outRecs* array

outRecs OUT Array containing the output record numbers.

Purpose Returns the record numbers in one level corresponding to a group of records in a different level.

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

Description The records in one level are related to those in another through the link field. These in turn are related to the next. In this way each record in any level is related to others in all the levels of the point structure. The purpose of *PTgetrecnums* is to return the record numbers in one level that are connected to a given set of records in a different level. Note that the two levels need not be adjacent.

Example In this example, we get the record number in the second level that are related to the first record in the first level.

```
nrec = 1;

recs[0] = 0;

inLevel = 0;

outLevel = 1;

status = PTgetrecnums(pointID2, inLevel, outLevel, nrec,
recs, &outNrec, outRecs);
```

FORTTRAN integer
ptgetrecnum(*pointID,inlevel,outlevel,innrec,inrecs,outnrec,outrecs*)
integer*4 *pointid*
integer*4 *inlevel*
integer*4 *outlevel*
integer*4 *innrec*
integer*4 *inrecs*
integer*4 *outnrec*
integer*4 *outnrecs*

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

```
status = ptgetrecnums(pointid2,inlevel,outlevel,nrec,recs,outnrec,outrecs)
```


Retrieve Information About Point Attributes

PTinqattrs

int32 PTinqattrs(int32 *pointID*, char **attrlist*, int32 **strbufsize*)

pointID IN: Point id returned by PTcreate or PTattach

attrlist OUT: Attribute list (entries separated by commas)

strbufsize OUT: String length of attribute list

Purpose Retrieve information about attributes defined in point.

Return value Number of attributes found or (-1) if failure.

Description The attribute list is returned as a string with each attribute name separated by commas. If *attrlist* is set to NULL, then the routine will return just the string buffer size, *strbufsize*. This variable does not count the null string terminator.

Example In this example, we retrieve information about the attributes defined in a point structure. We assume that there are two attributes stored, *attrOne* and *attr_2*:

```
nattr = PTinqattrs(pointID, NULL, strbufsize);
```

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

```
nattr = PTinqattrs(pointID, attrlist, strbufsize);
```

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

```
"attrOne,attr_2".
```

FORTRAN integer*4 function ptingattrs(*pointid*,*attrlist*,*strbufsize*)

integer*4 *pointid*

character*(*) *attrlist*

integer*4 *strbufsize*

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

```
nattr = ptingattrs(pointid, attrlist, strbufsize)
```

Retrieve Point Structures Defined in HDF-EOS File

PTinqpoint

int32 PTinqpoint(char * *filename*, char **pointlist*, int32 **strbufsize*)

pointID IN: HDF-EOS filename

pointlist OUT: Point list (entries separated by commas)

strbufsize OUT: String length of point list

Purpose Retrieves number and names of points defined in HDF-EOS file.

Return value Number of points found or (-1) if failure.

Description The point list is returned as a string with each point name separated by commas. If *pointlist* is set to NULL, then the routine will return just the string buffer size, *strbufsize*. If *strbufsize* is also set to NULL, the routine returns just the number of points. Note that *strbufsize* does not count the null string terminator.

Example In this example, we retrieve information about the points defined in an HDF-EOS file, *HDFEOS.hdf*. We assume that there are two points stored, *PointOne* and *Point_2*:

```
npoint = PTinqpoint("HDFEOS.hdf", NULL, strbufsize);
```

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

```
npoint = PTinqpoint("HDFEOS.hdf", pointlist, strbufsize);
```

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

"PointOne,Point_2".

FORTRAN integer*4 function ptinqpoint(*filename,pointlist,strbufsize*)

character*(*) *filename*

character*(*) *pointlist*

integer*4 *strbufsize*

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

```
npoint = ptinqpoint('HDFEOS.hdf', pointlist, strbufsize)
```

Return Index Number of a Named Level

PTlevelindx

int32 PTlevelindx(int32 *pointID*, char **levelname*)

pointID IN: Point id returned by PTcreate or PTattach

linkfield IN: Level Name

Purpose Returns the level index (0-based) for a given (named) level.

Return value Returns the level index if successful and FAIL (-1) otherwise.

Description This routine returns the level index for a give level specified by name.

Example In this example, we return the level index of the *Observations* level in the multilevel point structure defined in *PTdeflevel*.

```
levindx = PTlevelindex(pointID2, "Observations");
```

The *levindx* variable will have the value 1.

FORTRAN integer*4 ptlevidx (*pointid* *levelname*)

```
integer*4    pointid
```

```
character*(*) levelname
```

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

```
levindx = ptlevidx(pointid2, "Observations")
```

Return Information on Fields in a Given Level

PTlevelinfo

int32 PTlevelinfo(int32 *pointID*, int32 *level*, char **fieldlist*, int32 *fldtype*[], int32 *fldorder*[])

pointID IN: Point id returned by PTcreate or PTattach

level IN: Point level (0-based)

fieldlist OUT: Field names in level

fldtype OUT: Number type of each field

fldorder OUT: Order of each field

Purpose Returns information on fields in a given level.

Return value Returns number of fields if successful and FAIL (-1) otherwise. Typical reasons for failure are an improper point id or level number.

Description This routine returns information about the fields in a given level.

Example In this example we return information about the *Desc-Loc* (1st) level defined previously.

```
nflds = PTlevelinfo(pointID2, 0, fldlist, fldtype,
fldorder);
```

The *fldlist* variable will be set to:
"Time,Longitude,Latitude,Channel,Value".

Then *nflds* = 5, the *fldtype* array = {22,5,5,22,5}, the *fldorder* array = {0,0,0,0,0}, and the *fldlevel* = {2,2,2,2,2}. The last variable is only useful when information on an entire point is requested.

FORTTRAN integer*4 function ptlevinfo(*pointID*, *level*, *fieldlist*, *fldtype*, *fldorder*)

integer*4 *pointid*

integer*4 *level*

character*(*) *fieldlist*

integer*4 *fldtype* (*)

integer*4 *fldorder* (*)

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

```
nflds = ptlevinfo(pointid2, 0, fldlist, fldtype, fldorder)
```

Unlike the C language example, all output parameters must be supplied in the call.

Return Number of Fields Defined in a Level

PTnfields

int32 PTnfields(int32 *pointID*, int32 *level*, int32 **strbufsize*)

pointID IN: Point id returned by PTcreate or PTattach

level IN: Level number (0-based)

strbufsize OUT: Size in bytes of fieldlist for level

Purpose Returns number of fields in a level and the size of the fieldlist.

Return value Returns number of fields if successful and FAIL (-1) otherwise.

Description This routine returns the number of fields in a level and the size of the comma-separated fieldlist. This value does NOT count the null character at the end of the string.

Example In this example we retrieve the number of levels in the 2nd point defined previously:

```
nflds = PTnfields(pointID2, 0, strbufsize);
```

The *nflds* variable will be 5 and the *strbufsize* variable equal to 38.

FORTTRAN integer*4 function ptnflds(*pointid*), *level*, *strbufsize*

integer*4 *pointid*

integer*4 *level*

integer*4 *strbufsize*

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

```
nflds = ptnflds(pointid2, 0, strbufsize)
```

Return Number of Levels in a Point Structure

PTnlevels

int32 PTnlevels(int32 *pointID*)

pointID IN: Point id returned by PTcreate or PTattach

Purpose Returns number of levels in a point.

Return value Returns number of levels if successful and FAIL (-1) otherwise. Typical reasons for failure are an improper point id.

Description This routine returns the number of levels in a point.

Example In this example we retrieve the number of levels in the 2nd point defined previously:

```
nlevels = PTnlevels(pointID2);
```

The *nlevels* variable will be 2.

FORTRAN integer*4 function ptnlevs(*pointid*)

```
integer*4    pointid
```

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

```
nlevels = ptnlevs(pointid2)
```

Return Number of Records in a Given Level

PTnrecs

int32 PTnrecs(int32 *pointID*, int32 *level*)

pointID IN: Point id returned by PTcreate or PTattach

level IN: Level number (0-based)

Purpose Returns number of records in a given level.

Return value Returns number of records in a given level if successful and FAIL (-1) otherwise. Typical reasons for failure are an improper point id or level number.

Description This routine returns the number of records in a given level.

Example In this example we retrieve the number of records in the first level of the 2nd point defined previously:

```
nrecs = PTnrecs(pointID2, 0);
```

FORTTRAN integer*4 function ptnrecs(*pointid*,*level*)

```
integer*4          pointid
```

```
integer*4          level
```

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

```
nrecs = ptnrecs(pointid2, 0)
```

Open HDF-EOS File

PTopen

int32 PTopen(char **fieldname*, intn *access*)

fieldname IN: Complete path and filename for the file to be opened

access IN: DFACC_READ, DFACC_RDWR or DFACC_CREATE

Purpose Opens or creates HDF file in order to create, read, or write a point.

Return value Returns the point file id handle (fid) if successful and FAIL (-1) otherwise.

Description This routine creates a new file or opens an existing one, depending on the access parameter.

Access codes:

DFACC_READ Open for read only. If file does not exist, error

DFACC_RDWR Open for read/write. If file does not exist, create it

DFACC_CREATE If file exist, delete it, then open a new file for read/write

Example In this example, we create a new point file named, *PointFile.hdf*. It returns the file handle, *fid*.

```
fid = PTopen("PointFile.hdf", DFACC_CREATE);
```

FORTRAN integer*4 function ptopen(*filename*, *access*)

character*(*) *filename*

integer *access*

The access codes should be defined as parameters:

parameter (DFACC_READ=1)

parameter (DFACC_RDWR=3)

parameter (DFACC_CREATE=4)

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

```
fid = ptopen("PointFile.hdf", DFACC_CREATE)
```

Note to users of the SDP Toolkit: Please refer to the *SDP Toolkit 5 User Guide for the ECS Project* (333-CD-003-002), Section 6.2.1.2, for information on how to obtain a file name (referred to as a "physical file handle") from within a PGE. See also Section 9 of this document for code examples.

Returns Information About a Time Period

PTperiodinfo

intn PTperiodinfo(int32 *pointID*, int32 *periodID*, int32 *level*, char **fieldlist*,
int32 **size*)

pointID IN: Point id

periodID IN: Period id returned by PTdeftimeperiod

level IN: Point level (0-based)

fieldname IN: List of fields to extract

size OUT: Size in bytes of subset period

Purpose Retrieves information about the subsetted period.

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

Description This routine returns information about a subsetted time period for a particular fieldlist. It is useful when allocating space for a data buffer for the subset.

Example In this example, we get the size of the subsetted time period defined in *PTdeftimeperiod* for the *Time* field.

```
status = PTperiodinfo(pointID, periodID, 0, "Time",  
&size);
```

FORTRAN integer function ptperinfo(*pointid,periodid,level,fieldlist,size*)

integer*4 *pointid*

integer*4 *periodid*

integer*4 *level*

character*(*) *fieldlist*

integer*4 *size*

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

```
status = ptperinfo(pointid,periodid,0,"Time",size)
```

Returns Record Numbers within a Time Period

PTperiodrecs

intn PTperiodrecs(int32 *pointID*, int32 *periodID*, int32 *level*, int32 **nrec*,
int32 *recs*[])

pointID IN: Point id

periodID IN: Period id returned by PTdeftimeperiod

level IN: Point level (0-based)

nrec OUT: Number of records within time period in level

recs OUT: Record numbers of subsetted records in level

Purpose Retrieves record numbers within time period.

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

Description This routine the record numbers within a subsetted time period for a particular level. If the *recs* array is set to NULL, then the routine simply returns the number of records.

Example In this example, we get the number of records and record numbers within the subsetted area of interest defined in *PTdeftimeperiod* for the 0th level.

```
status = PTperiodrecs(pointID, regionID, 0, &nrec, recs);
```

FORTTRAN integer function ptperrrecs(*pointid,periodid,level,nrec,recs*)

integer*4 *pointid*

integer*4 *periodid*

integer*4 *level*

integer*4 *nrec*

integer*4 *recs*(*)

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

```
status = ptperrrecs(pointid,periodid,0,nrec,recs)
```

Read Point Attribute

PTreadattr

intn PTreadattr(int32 *pointID*, char **attrname*, VOIDP *datbuf*)

pointID IN: Point id returned by PTcreate or PTattach

attrname IN: Attribute name

datbuf IN: Buffer allocated to hold attribute values

Purpose Reads attribute from a point.

Return value Returns SUCCEED (0) if successful and FAIL (-1) otherwise. Typical reasons for failure are an improper point 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 = PTreadattr(pointID, "ScalarFloat", &f32);
```

FORTTRAN integer function ptrdattr(*pointid*, *attrname*, *datbuf*)

integer*4 *pointid*

character*(*) *attrname*

<valid type> *datbuf*(*)

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

```
status = ptrdattr(pointid, "ScalarFloat", f32)
```

Read Records From a Point Level

PTreadlevel

intn PTreadlevel(int32 *pointID*, int32 *level*, char *fieldlist*, int32 *nrec*, int32 *recs*[],
VOIDP *buffer*)

pointID IN: Point id returned by PTcreate or PTattach
level IN: Level to read (0-based)
fieldlist IN: List of fields to read
nrec IN: Number of records to read
recs IN: Record number of records to read (0 - based)
buffer OUT: Buffer to store data
Purpose Reads data from a point level.

Return value Returns SUCCEED (0) if successful and FAIL (-1) otherwise. Typical reasons for failure are an improper point id or unknown fieldname.

Description This routine reads data from the specified fields and records of a single level in a point. Sufficient space in the read buffer must be allocated by the user.

Example In this example we read records 0, 2, and 3 from the *Temperature* and *Mode* fields in the first level in the point referred to by the point id, *pointID1*.

```
int32 recs[3] = {0,2,3};  
status = PTreadlevel(pointID1, 0, "Temperature,Mode", 3,  
recs, buffer);
```

FORTRAN integer function
ptrdlev(*pointid,level,fieldlist,nrec,recs,buffer*)
integer*4 *pointid*
integer*4 *level*
character*(*) *fieldlist*
integer*4 *nrec*
integer*4 *recs*(*)
<valid type> *buffer*(*)

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

```
integer*4 recs(10)  
recs(1) = 0  
recs(2) = 2  
recs(3) = 3  
status = ptrdlev(pointid1, 1, "Temperature,Mode", 3, recs,  
buffer)
```

Returns Information About a Geographic Region

PTregioninfo

intn PTregioninfo(int32 *pointID*, int32 *regionID*, int32 *level*, char **fieldlist*, int32 **size*)

pointID IN: Point id

regionID IN: Region id returned by PTdefboxregion

level IN: Point level (0-based)

fieldname IN: List of fields to extract

size OUT: Size in bytes of subset period

Purpose Retrieves information about the subsetted region.

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

Description This routine returns information about a subsetted area of interest for a particular fieldlist. It is useful when allocating space for a data buffer for the subset.

Example In this example, we get the size of the subsetted area of interest defined in *PTdefboxregion* from the *Longitude* and *Latitude* fields.

```
status = PTregioninfo(pointID, regionID, 0,
"Longitude, Latitude", &size);
```

FORTRAN integer function ptreinfo(*pointid*,*regionid*,*level*,*fieldlist*,*size*)

integer*4 *pointid*

integer*4 *regionid*

integer*4 *level*

character*(*) *fieldlist*

integer*4 *size*

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

```
status =
ptreinfo(pointid,regionid,0,"Longitude, Latitude",size)
```

Returns Record Numbers within a Geographic Region

PTregionrecs

intn PTregionrecs(int32 *pointID*, int32 *regionID*, int32 *level*, int32 **nrec*,
int32 *recs*[])

pointID IN: Point id

regionID IN: Region id returned by PTdefboxregion

level IN: Point level (0-based)

nrec OUT: Number of records within geographic region in level

recs OUT: Record numbers of subsetted records in level

Purpose Retrieves record numbers within geographic region.

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

Description This routine the record numbers within a subsetted geographic region for a particular level. If the *recs* array is set to NULL, then the routine simply returns the number of records.

Example In this example, we get the number of records and record numbers within the subsetted area of interest defined in *PTdefboxregion* for the 0th level.

```
status = PTregionrecs(pointID, regionID, 0, &nrec, recs);
```

FORTTRAN integer function ptregrecs(*pointid*,*regionid*,*level*,*nrec*,*recs*)

integer*4 *pointid*

integer*4 *regionid*

integer*4 *level*

integer*4 *nrec*

integer*4 *recs*(*)

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

```
status = ptregrecs(pointid,regionid,0,nrec,recs)
```

Return Information About Fields in a Point

PTsizeof

int32 PTsizeof(int32 *pointID*, char **fieldlist*, int32 *fldlevel*[])

pointID IN: Point id returned by PTcreate or PTattach

fieldlist IN: Field names

fldlevel OUT: Level number of each field

Purpose Returns information on specified fields in point.

Return value Returns size in bytes of specified fields and FAIL (-1) otherwise. Typical reasons for failure are an improper point id or field names.

Description This routine returns information about specified fields in a point regardless of level.

Example In this example we return the size in bytes of the *Label* and *Rainfall* fields in the 2nd point defined in the *PTdeflevel* routine.

```
size = PTsizeof(pointID2, "Label,Rainfall", fldlevel);
```

The *size* variable will be 8 and the *fldlevel* = {1,2}.

FORTRAN integer*4 function ptsizeof(*pointID*, *fieldlist*, *fldlevel*)

integer*4 *pointid*

character*(*) *fieldlist*

integer*4 *fldlevel* (*)

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

```
size = ptsizeof(pointid2, "Label,Rainfall", fldlevel)
```

Update Records in a Point Structure

PTupdatelevel

intn PTupdatelevel(int32 *pointID*, int32 *level*, char *fieldlist*, int32 *nrec*, int32 *recs*[], VOIDP *data*)

pointID IN: Point id returned by PTcreate or PTattach

level IN: Level to update (0-based)

fieldlist IN: List of fields to update

nrec IN: Number of records to update

recs IN: Record number of records to update (0 - based)

data IN: Values to be written to the fields

Purpose Updates (corrects) data to a point level.

Return value Returns SUCCEED (0) if successful and FAIL (-1) otherwise. Typical reasons for failure are an improper point id or unknown fieldname.

Description This routine updates the specified fields and records of a single level.

Example In this example we update records 0, 2, and 3 in the *Temperature* and *Mode* fields in the first level in the point referred to by the point id, *pointID1*.

```
int32 recs[3] = {0,2,3};
```

```
/* Fill Data Buffer */
```

```
status = PTupdatelevel(pointID1, 1, "Temperature,Mode", 3,  
recs, datbuf);
```

The user may update a single record or all records in precisely the same manner as that used in the *PTreadlevel* examples.

FORTRAN integer function
ptuplev(*pointid,level,fieldlist,nrec,recs,buffer*)
integer*4 *pointid*
integer*4 *level*
character*(*) *fieldlist*
integer*4 *nrec*
integer*4 *recs*(*)
<valid type> *buffer*(*)

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

```
integer*4 recs(10)

recs(1) = 0

recs(2) = 2

recs(3) = 3
```

C Fill Data Buffer

```
status = ptuplev(pointidl, 1, "Temperature,Mode", 3, recs,
                 datbuf)
```

Write/Update Point Attribute

PTwriteattr

intn PTwriteattr(int32 *pointID*, char **attrname*, int32 *ntype*, int32 *count*
VOIDP *datbuf*)

pointID IN: Point id returned by PTcreate or PTattach

attrname IN: Attribute name

ntype IN: Number type of attribute

count IN: Number of values to store in attribute

datbuf IN: Attribute values

Purpose Writes/Updates attribute in a point.

Return value Returns SUCCEED (0) if successful and FAIL (-1) otherwise. Typical reasons for failure are an improper point 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 = PTwriteattr(pointid, "ScalarFloat", DFNT_FLOAT32,  
1, &f32);
```

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

```
f32 = 3.14159;
```

```
status = PTwriteattr(pointid, "ScalarFloat", DFNT_FLOAT32,  
1, &f32);
```

FORTRAN integer function ptwrattr(*pointid*, *attrname*,
ntype, *count*, *datbuf*)
integer*4 *pointid*
character*(*) *attrname*
integer*4 *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 = ptwrattr(pointid, "ScalarFloat", DFNT_FLOAT32, 1,
                 f32)
```

Write New Records to a Point Level

PTwritelevel

intn PTwritelevel(int32 *pointID*, int32 *level*, int32 *nrec*, VOIDP *data*)

pointID IN: Point id returned by PTcreate or PTattach

level IN: Level to write (0-based)

nrec IN: Number of records to write

data IN: Values to be written to the field

Purpose Writes (appends) new records to a point level.

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

Description This routine writes (appends) full records to a level. The data in each record must be *packed*. Please refer to the section on Vdatas in the HDF documentation. The input data buffer must be sufficient to fill the number of records designated.

Example In this example we write 5 records to the first level in the point referred to by the point id, *pointID1*.

```
/* Fill Data Buffer */
```

```
status = PTwritelevel(pointID1, 0, 5, datbuf);
```

FORTTRAN integer function
ptwrlev(*pointid,level,nrec,data*)
integer*4 *pointid*
integer*4 *level*
integer*4 *nrec*
<valid type> *data*(*)

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

```
status = ptwrlev(pointid1, 0, 5, datbuf)
```

2.1.2 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

int32 SWattach(int32 *fid*, char **swathname*)

fid IN: Swath file id returned by SWopen

swathname 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 and 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 *swathname* parameter as the identifier.

Example In this example, we attach to the previously created swath, "ExampleSwath", within the HDF file, *SwathFile.hdf*, referred to by the handle, *fid*:

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

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

FORTTRAN integer*4 function swattach(*fid*,*swathname*)

integer*4 *fid*

character*(*) *swathname*

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

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

Return Information About a Swath Attribute

SWattrinfo

intn SWattrinfo(int32 *swathID*, char **attrname*, int32 * *numbertype*, int32 **count*)

swathID IN: Swath id returned by SWcreate or SWattach

attrname IN: Attribute name

numbertype OUT: Number type of attribute

count OUT: Number of total bytes in attribute

Purpose Returns information about a swath attribute

Return value Returns SUCCEED (0) if successful and 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 *ScalarFloat* attribute.

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

The *nt* variable will have the value 5 and *count* will have the value 4.

FORTRAN integer function swattrinfo(*swathid*, *attrname*, *ntype*, *count*,)

integer*4 *swathid*

character*(*) *attrname*

integer*4 *ntype*

integer*4 *count*

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

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

Close an HDF-EOS File

SWclose

intn SWclose(int32 *fid*)

fid IN: Swath file id returned by SWopen

Purpose Closes file.

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

Description This routine closes the HDF swath file.

Example

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

FORTRAN integer function swclose(fid)

```
integer*4 fid
```

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

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

Retreive Compression Information for Field

SWcompinfo

intn SWcompinfo(int32 *swathID*, char **fieldname*, int32 **compcode*, intn *compparm*[])

swathID IN: Swath id returned by SWcreate or SWattach

fieldname IN: Fieldname

compcode OUT: HDF compression code

compparm OUT: Compression parameters

Purpose Retrieves compression information about a field.

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

Description This routine returns the compression code and compression parameters for a given field.

Example To retrieve the compression information about the *Opacity* field defined in the *SWdefcomp* 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*)

integer*4 *swathid*

character*(*) *fieldname*

integer*4 *compcode*

integer *compparm*

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

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

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

Create a New Swath Structure

SWcreate

int32 SWcreate(int32 *fid*, char **swathname*)

ffid IN: Swath file id returned by SWopen

swathname IN: Name of swath to be created

Purpose Creates a swath within the file.

Return value Returns the swath handle (*swathID*) if successful and FAIL (-1) otherwise.

Description The swath is created as a Vgroup within the HDF file with the name *swathname* and class *SWATH*.

Example In this example, we create a new swath structure, *ExampleSwath*, in the previously created file, *SwathFile.hdf*.

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

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

FORTRAN integer*4 function swcreate(*fid*,*swathname*)

```
integer*4 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 SWdefboxregion(int32 *swathID*, float64 *cornerlon*[], float64 *cornerlat*[], int32 *mode*)

swathID IN: Swath id returned by SWcreate or SWattach

cornerlon IN: Longitude in decimal degrees of box corners

cornerlat IN: Latitude in decimal degrees of box corners

mode IN: Cross Track inclusion mode

Purpose Defines a longitude-latitude box region for a swath.

Return value Returns the swath region ID if successful and 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 *SWextractregion* 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 *Longitude* and *Latitude* (or *Colatitude*) 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 = SWdefboxregion(swathID, cornerlon, cornerlat,  
                           HDFE_MIDPOINT);
```

FORTTRAN integer*4 function swdefboxreg(*swathid, cornerlon, cornerlat, mode*)
integer*4 *swathid*
real*8 *cornerlon*
real*8 *cornerlat*
integer*4 *mode*

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

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

Set Swath Field Compression

SWdefcomp

intn SWdefcomp(int32 *swathID*, int32 *comPCODE*, intn *compparm*[])

swathID IN: Swath id returned by SWcreate or SWattach

comPCODE IN: HDF compression code

compparm IN: Compression parameters (if applicable)

Purpose Sets the field compression for all subsequent field definitions.

Return value Returns SUCCEED(0) if successful and 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: run length encoding (HDFE_COMP_RLE = 1), skipping Huffman (HDFE_COMP_SKPHUFF = 3), deflate (gzip) (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 *Pressure* using run length encoding, the *Opacity* field using deflate compression, the *Spectra* field with skipping Huffman compression, and use no compression for the *Temperature* field.

```
status = SWdefcomp(swathID, HDFE_COMP_RLE, NULL);

status = SWdefdatafield(swathID, "Pressure", "Track,Xtrack",
DFNT_FLOAT32, HDFE_NOMERGE);

compparm[0] = 5;

status = SWdefcomp(swathID, HDFE_COMP_DEFLATE, compparm);

status = SWdefdatafield(swathID, "Opacity", "Track,Xtrack",
DFNT_FLOAT32, HDFE_NOMERGE);
```

```

status = SWdefcomp(swathID, HDFE_COMP_SKPHUFF, NULL);

status = SWdefdatafield(swathID, "Spectra",
"Tracks,Track,Xtrack", DFNT_FLOAT32, HDFE_NOMERGE);

status = SWdefcomp(swathID, HDFE_COMP_NONE, NULL);

status = SWdefdatafield(swathID, "Temperature",
"Track,Xtrack", DFNT_FLOAT32, HDFE_AUTOMERGE);

```

Note that the `HDFE_AUTOMERGE` parameter will be ignored in the *Temperature* field definition.

FORTTRAN integer function `swdefcomp(swathid, compcode, compparm)`

```

integer*4    swathid
integer      compcode
integer      compparm

```

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

```

parameter (HDFE_COMP_NONE=0)

parameter (HDFE_COMP_RLE=1)

parameter (HDFE_COMP_SKPHUFF=3)

parameter (HDFE_COMP_DEFLATE=4)

integer    compparm(5)

status = swdefcomp(swathid, HDFE_COMP_RLE, compparm)

status = swdefdfld(swathid, "Pressure", "Track,Xtrack",
DFNT_FLOAT32, HDFE_NOMERGE)

compparm(1) = 5

status = swdefcomp(swathid, HDFE_COMP_DEFLATE, compparm)

status = swdefdfld(swathid, "Opacity", "Track,Xtrack",
DFNT_FLOAT32, HDFE_NOMERGE)

status = swdefcomp(swathid, HDFE_COMP_SKPHUFF, compparm)

status = swdefdfld(swathid, "Spectra", "Bands,Track,Xtrack",
DFNT_FLOAT32, HDFE_NOMERGE)

status = swdefcomp(swathid, HDFE_COMP_NONE, compparm)

status = swdefdfld(swathid, "Temperature", "Track,Xtrack",
DFNT_FLOAT32, HDFE_AUTOMERGE)

```

Define a New Data Field Within a Swath

SWdefdatafield

intn SWdefdatafield(int32 *swathID*, char **fieldname*, char **dimlist*, int32 *numbertype*, int32 *merge*)

swathID IN: Swath id returned by SWcreate or SWattach

fieldname IN: Name of field to be defined

dimlist IN: The list of data dimensions defining the field

numbertype IN: The number type of the data stored in the field

merge 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 and 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 HDFE_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.

Example In this example, we define a three dimensional data field named *Spectra* with dimensions *Bands*, *DataTrack*, and *DataXtrack*:

```
status = SWdefdatafield(swathID, "Spectra",  
                        "Bands,DataTrack,DataXtrack", DFNT_FLOAT32,  
                        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*,
numbertype,*merge*)

integer*4 *swathid*

character*(*) *fieldname*

character*(*) *dimlist*

integer*4 *numbertype*

integer*4 *merge*

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

```
parameter (DFNT_FLOAT32=5)
```

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

```
status = swdefdfld(swathid, "Spectra",  
                  "DataXtrack,DataTrack,Bands", DFNT_FLOAT32,  
                  HDFE_AUTOMERGE)
```

Define a New Dimension Within a Swath

SWdefdim

intn SWdefdim(int32 *swathID*, char **fieldname*, int32 *dim*)

fieldname IN: Name of dimension to be defined

dim IN: The size of the dimension

Purpose Defines a new dimension within the swath.

Return value Returns SUCCEED (0) if successful and 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, *GeoTrack*, of size 2000, a cross track dimension, *GeoXtrack*, 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, *SD_UNLIMITED*:

```
status = SWdefdim(swathID, "Unlim", SD_UNLIMITED);
```

FORTTRAN integer function swdefdim(*swathid*,*fieldname*,*dim*)

```
integer*4    swathid
character*(*) fieldname
integer*4    dim
```

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

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

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

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


Define Mapping Between Geolocation and Data Dimensions

SWdefdimmap

intn SWdefdimmap(int32 *swathID*, char **geodim*, char **datadim*, int32 *offset*, int32 *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 and 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 *GeoTrack* dimension corresponds to the first element of the *DataTrack* dimension and the data dimension has twice the resolution as the geolocation dimension, and (2) the first element of the *GeoXtrack* dimension corresponds to the second element of the *DataTrack* 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*4 *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

intn SWdefgeofield(int32 *swathID*, char **fieldname*, char **dimlist*,
int32 *numbertype*, int32 *merge*)

swathID IN: Swath id returned by SWcreate or SWattach

fieldname IN: Name of field to be defined

dimlist IN: The list of geolocation dimensions defining the field

numbertype IN: The number type of the data stored in the field

merge 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 and 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.

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", DFNT_FLOAT32, HDFE_AUTOMERGE);  
  
status = SWdefgeofield(swathID, "Latitude",  
"GeoTrack,GeoXtrack", DFNT_FLOAT32, 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, numbertype,  
merge)  
integer*4    swathid  
character*(*) fieldname  
character*(*) dimlist  
integer*4    numbertype  
integer*4    merge
```

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

```
parameter (DFNT_FLOAT32=5)  
parameter (HDFE_AUTOMERGE=1)  
status = swdefgfld(swathid, "Longitude",  
                  "GeoXtrack,GeoTrack", DFNT_FLOAT32, HDFE_AUTOMERGE)
```

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

Define Indexed Mapping Between Geolocation and Data Dimension

SWdefidxmap

intn SWdefidxmap(int32 *swathID*, char **geodim*, char **datadim*, int32 *index*[]),

swathID IN: Swath id returned by SWcreate or SWattach
geodim IN: Geolocation dimension name
datadim IN: Data dimension name
index 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 and 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*4 swathid
```

```
character*(*) geodim
```

```
character*(*) datadim
```

```
integer*4 index (*)
```

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

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

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

Define a Time Period of Interest

SWdeftimeperiod

int32 SWdeftimeperiod(int32 *swathID*, float64 *starttime*, float64 *stoptime*
int32 *mode*)

swathID IN: Swath id returned by SWcreate or SWattach

starttime IN: Start time of period

stoptime IN: Stop time of period

mode IN: Cross Track inclusion mode

Purpose Defines a time period for a swath.

Return value Returns the swath period ID if successful and FAIL (-1) otherwise.

Description This routine defines a time period for a swath. It returns a swath period ID which is used by the *SWextractperiod* 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 *Time* 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*4 function swdeftmeper(*swathid*, *starttime*, *stoptime*, *mode*)
integer*4 *swathid*
real*8 *starttime*
real*8 *stoptime*
integer*4 *mode*

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

SWdefvrtregion

int32 SWdefvrtregion(int32 *swathID*, int32 *regionID*, char **vertObj*, float64 *range*[])

swathID IN: Swath id returned by SWcreate or SWattach

regionID IN: Region (or period) id from previous subset call

vertObj IN: Dimension or field to subset by

range IN: Minimum and maximum range for subset

Purpose Subsets on a **monotonic** field or contiguous elements of a dimension.

Return value Returns the swath region ID if successful and FAIL (-1) otherwise.

Description Whereas the *SWdefboxregion* and *SWdeftimeperiod* 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 **monotonic** (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 *SWdefboxregion* or *SWdeftimeperiod* 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. In 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 *SWdefvrtregion* (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 = SWdefvrtregion(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 = SWdefvrtregion(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 = SWdefvrtregion(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 = SWdefvrtregion(swathID, regionID, "FreqRange",
freq);
```

FORTRAN integer*4 function swdefvrtreg(*swathid, regionid, vertobj, range*)
 integer*4 *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 = swdefvrtreg(swathid, HDFE_NOPREVSUB, "Pressure",
range)
range(1) = 3           ! Note 1-based element numbers
range(2) = 6
regionid = swdefvrtreg(swathid, HDFE_NOPREVSUB,
"DIM:Height", range)
regionid = swdefvrtreg(swathid, subsetid, "Pressure", range)
regionid = swdefvrtreg(swathid, regionid, "FreqRange", freq)
```

Detach from a Swath Structure

SWdetach

intn SWdetach(int32 *swathID*)

swathID IN: Swath id returned by SWcreate or SWattach

Purpose Detaches from swath interface.

Return value Returns SUCCEED (0) if successful and 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, *ExampleSwath*:

```
status = SWdetach(swathID);
```

FORTTRAN integer function swdetach(swathid)

```
integer*4          swathid
```

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

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

Retrieve Size of Specified Dimension

SWdiminfo

int32 SWdiminfo(int32 *swathID*, char **dimname*)

swathID IN: Swath id returned by SWcreate or SWattach

dimname IN: Dimension name

Purpose Retrieve size of specified dimension.

Return value Size of dimension. 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.

FORTTRAN integer*4 function swdiminfo(*swathid*,*dimname*)

```
integer*4    swathid
```

```
character*(*) dimname
```

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

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

Duplicate a Region or Period

SWdupregion

int32 SWdupregion(int32 *regionID*)

<i>pointID</i>	IN: Region or period id returned by SWdefboxregion, SWdeftimeperiod, or SWdefvrtregion.
Purpose	Duplicates a region.
Return value	Returns new region or period ID.
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 <i>SWdefboxregion</i> , duplicate the region creating a new region ID, <i>regionID2</i> , and then perform two different vertical subsets of these (identical) geographic subset regions:

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

regionID2 = SWdupregion(regionID);

regionID = SWdefvrtregion(swathID, regionID, "Pressure",
                          rangePres);

regionID2 = SWdefvrtregion(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 = swdefvrtreg(swathid, regionid, 'Pressure',
                      rangePres)

regionid2 = swdefvrtreg(swathid, regionid2, 'Temperature',
                      rangeTemp)
```

Read Data from a Defined Time Period

SWextractperiod

intn SWextractperiod(int32 *periodID*, char * *fieldname*, int32 *external_mode*,
VOIDP *buffer*)

regionID IN: Period id returned by SWdeftimeperiod

fieldname IN: Field to subset

external_mode IN: External geolocation mode

rank OUT: Data buffer

Purpose Extracts (reads) from subsetted time period.

Return value Returns SUCCEED (0) if successful and 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 *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 time period defined in *SWdeftimeperiod* from the *Spectra* field. Both the geolocation fields and the *Spectra* data field are in the same swath.

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

FORTTRAN integer function swextper(*periodid*, *fieldname*, *external_mode*, *buffer*)

integer*4 *periodid*

character*(*) *fieldname*

integer*4 *external_mode*

<valid type> *buffer*(*)

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

```
parameter (HDFE_INTERNAL=0)  
  
status = swextper(periodid, "Spectra", HDFE_INTERNAL,  
datbuf)
```

Read Data from a Geographic Region

SWextractregion

intn SWextractregion(int32 *swathID*, int32 *regionID*, char * *fieldname*, int32 *external_mode*, VOIDP *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 and 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 geolocation 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*4 *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

intn SWfieldinfo(int32 *swathID*, char **fieldname*, int32 **rank*, int32 *dims*[], int32 **numbertype*, char **dimlist*)

swathID IN: Swath id returned by SWcreate or SWattach

fieldname IN: Fieldname

rank OUT: Rank of field

dims OUT: Array containing the dimension sizes of the field

numbertype OUT: Pointer to the numbertype of the field

dimlist 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 and FAIL (-1) if 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 *Spectra* data fields:

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

The return parameters will have the following values:

rank=3, *numbertype*=5, *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*4 *swathid*

character*(*) *fieldname*

integer*4 *rank*

integer*4 *dims*(*)

integer*4 *numbertype*

integer*4 *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*=5, *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

intn SWgetfillvalue(int32 *swathID*, char **fieldname*, VOIDP *fillvalue*)

swathID IN: Swath id returned by SWcreate or SWattach

fieldname IN: Fieldname

fillvalue OUT: Space allocated to store the fill value

Purpose Retrieves fill value for the specified field.

Return value Returns SUCCEED (0) if successful and 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*4 swathid
```

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

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

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

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

Retrieve Indexed Geolocation Mapping

SWidxmapinfo

int32 SWidxmapinfo(int32 *swathID*, char **geodim*, char **datadim*, int32 *index*[])

swathID IN: Swath id returned by SWcreate or SWattach

geodim IN: Indexed Geolocation dimension name

datadim IN: Indexed Data dimension name

index OUT: Mapping offset

Purpose Retrieve indexed array of specified geolocation mapping.

Return value Returns size of indexed array if successful and FAIL (-1) if 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}.

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

integer*4 *swathid*

character*(*) *geodim*

character*(*) *datadim*

integer*4 *index*(*)

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

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

Retrieve Information Swath Attributes

SWinqatrs

int32 SWinqatrs(int32 *swathID*, char **attrlist*, int32 **strbufsize*)

swathID IN: Swath id returned by SWcreate or SWattach

attrlist OUT: Attribute list (entries separated by commas)

strbufsize OUT: String length of attribute list

Purpose Retrieve information about attributes defined in swath.

Return value Number of attributes found or (-1) if failure.

Description The attribute list is returned as a string with each attribute name separated by commas. If *attrlist* is set to NULL, then the routine will return just the string buffer size, *strbufsize*. 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 = SWinqatrs(swathID, NULL, strbufsize);
```

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

```
nattr = SWinqatrs(swathID, attrlist, strbufsize);
```

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

```
"attrOne,attr_2".
```

FORTRAN integer*4 function swinqatrs(*swathid*,*attrlist*,*strbufsize*)

integer*4 *swathid*

character*(*) *attrlist*

integer*4 *strbufsize*

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

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

Retrieve Information About Data Fields Defined in Swath

SWinqdatafields

int32 SWinqdatafields(int32 *swathID*, char **fieldlist*, int32 *rank*[], int32 *numbertype*[])

swathID IN: Swath id returned by SWcreate or SWattach
fieldlist OUT: Listing of data fields (entries separated by commas)
rank OUT: Array containing the rank of each data field
numbertype OUT: Array containing the numbertype of each data field
Purpose Retrieve information about all of the data fields defined in swath.
Return value Number of data fields found. If -1, could signify improper swath id.
Description The field list is returned as a string with each data field separated by commas. The *rank* and *numbertype* arrays will have an entry for each field. Output parameters set to *NULL* 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]={5}

FORTTRAN integer*4 function swinqdflds(*swathid*, *fieldlist*, *rank*, *numbertype*)

integer*4 *swathid*

character*(*) *fieldlist*

integer*4 *rank*(*)

integer*4 *numbertype*(*)

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

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

Retrieve Information About Dimensions Defined in Swath

SWinqdims

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

swathID IN: Swath id returned by SWcreate or SWattach

dimname OUT: Dimension list (entries separated by commas)

dims 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 -1, could signify an improper swath id.

Description The dimension list is returned as a string with each dimension name separated by commas. Output parameters set to *NULL* will not be returned.

Example In this example, we retrieve information about the dimensions defined in the *ExampleSwath* structure:

```
ndims = SWinqdims(swathID, dimname, dims);
```

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

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

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

FORTTRAN integer*4 function swinqdims(*swathid*,*dimname*,*dims*)

```
integer*4    swathid
```

```
character*(*) dimname
```

```
integer*4    dims(*)
```

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

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

Retrieve Information About Geolocation Fields Defined in Swath

SWinqgeofields

int32 SWinqgeofields(int32 *swathID*, char **fieldlist*, int32 *rank*[], int32 *numbertype*[])

swathID IN: Swath id returned by SWcreate or SWattach
fieldlist OUT: Listing of geolocation fields (entries separated by commas)
rank OUT: Array containing the rank of each geolocation field
numbertype OUT: Array containing the numbertype of each geolocation field
Purpose Retrieve information about all of the geolocation fields defined in swath.
Return value Number of geolocation fields found. If -1, could signify an improper swath id.

Description The field list is returned as a string with each geolocation field separated by commas. The *rank* and *numbertype* arrays will have an entry for each field. Output parameters set to *NULL* 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]={5,5}

FORTRAN integer*4 function swinqgfls(swathid, fieldlist, rank, numbertype)

integer*4 *swathid*

character*(*) *fieldlist*

integer*4 *rank*(*)

integer*4 *numbertype*(*)

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

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

Retrieve Information About Indexed Mappings Defined in Swath

SWinqidxmaps

int32 SWinqidxmaps(int32 swathID, char *idxmap, int32 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 -1, could signify 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, *idxmap*, will contain the string:

"IdxGeo/IdxData" with *nidxmaps = 1* and *idxsizes[1]={5}*.

FORTRAN integer*4 function

```
swinqimaps(swathid,dimmap,idxsizes)
```

```
integer*4 swathid
```

```
character*(*) dimmap
```

```
integer*4 idxsizes(*)
```

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

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


Retrieve Information About Dimension Mappings Defined in Swath

SWinqmaps

int32 SWinqmaps(int32 *swathID*, char **dimmap*, int32 *offset*[], int32 *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 -1, could signify 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*4 *swathid*
character*(*) *dimmap*
integer*4 *offset*(*)
integer*4 *increment*(*)

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

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

Retrieve Swath Structures Defined in HDF-EOS File

SWinqswath

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

swathID IN: HDF-EOS filename

swathlist OUT: Swath list (entries separated by commas)

strbufsize OUT: String length of swath list

Purpose Retrieves number and names of swaths defined in HDF-EOS file.

Return value Number of swaths found or (-1) if failure.

Description The swath list is returned as a string with each swath name separated by commas. If *swathlist* is set to NULL, then the routine will return just the string buffer size, *strbufsize*. If *strbufsize* is also set to NULL, the routine returns just the number of swaths. Note that *strbufsize* does not count the null string terminator.

Example In this example, we retrieve information about the swaths defined in an HDF-EOS file, *HDFEOS.hdf*. We assume that there are two swaths stored, *SwathOne* and *Swath_2*:

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

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

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

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

“*SwathOne,Swath_2*”.

FORTTRAN integer*4 function swinqswath(*filename,swathlist,strbufsize*)

character*(*) *filename*

character*(*) *swathlist*

integer*4 *strbufsize*

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

```
nswath = swinqswath('HDFEOS.hdf', swathlist, strbufsize)
```

Retrieve Offset and Increment of Specific Dimension Mapping

SWmapinfo

intn SWmapinfo(int32 *swathID*, char **geodim*, char **datadim*, int32 *offset*, int32 *increment*)

swathID IN: Swath id returned by SWcreate or SWattach

geodim IN: Geolocation dimension name

datadim IN: Data dimension name

offset OUT: Mapping offset

increment OUT: Mapping increment

Purpose Retrieve offset and increment of specific monotonic geolocation mapping.

Return value Returns SUCCEED (0) if successful and FAIL (-1) if 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 *GeoTrack* and *DataTrack* dimensions:

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

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

FORTTRAN integer function swmapinfo(*swathid*, *geodim*, *datadim*, *offset*, *increment*)

integer*4 *swathid*

character*(*) *geodim*

character*(*) *datadim*

integer*4 *offset*

integer*4 *increment*

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

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

Return Number of Specified Objects in a Swath

SWnentries

int32 SWnentries(int32 *swathID*, int32 *entrycode*, int32 **strbufsize*)

swathID IN: Swath id returned by SWcreate or SWattach

entrycode IN: Entrycode

strbufsize OUT: String buffer size

Purpose Returns number of entries and descriptive string buffer size for a specified entity.

Return value Number of entries. If -1, could signify 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*4 *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

int32 SWopen(char *filename, intn access)

<i>filename</i>	IN: Complete path and filename for the file to be opened
<i>access</i>	IN: DFACC_READ, DFACC_RDWR or DFACC_CREATE
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 and FAIL (-1) otherwise.
Description	This routine creates a new file or opens an existing one, depending on the access parameter. Access codes: DFACC_READ Open for read only. If file does not exist, error DFACC_RDWR Open for read/write. If file does not exist, create it DFACC_CREATE If file exist, delete it, then open a new file for read/write
Example	In this example, we create a new swath file named, <i>SwathFile.hdf</i> . It returns the file handle, <i>fid</i> .

```
fid = SWopen("SwathFile.hdf", DFACC_CREATE);
```

FORTRAN integer*4 function swopen(filename, access)
character*(*) filename
integer access

The access codes should be defined as parameters:

```
parameter (DFACC_READ=1)  
parameter (DFACC_RDWR=3)  
parameter (DFACC_CREATE=4)
```

The equivalent FORTRAN code for the example above is:

```
fid = swopen("SwathFile.hdf", DFACC_CREATE)
```

Note to users of the SDP Toolkit: Please refer to the *SDP Toolkit 5 User Guide for the ECS Project* (333-CD-003-002), Section 6.2.1.2, for information on how to obtain a file name (referred to as a “physical file handle”) from within a PGE. See also Section 9 of this document for code examples.

Return Information About a Defined Time Period

SWperiodinfo

intn SWperiodinfo(int32 *swathID*, int32 *periodID*, char * *fieldname*, int32 **ntype*, int32 **rank*, int32 *dims*[], int32 **size*)

swathID IN: Swath id returned by SWcreate or SWattach

periodID IN: Period id returned by SWdeftimeperiod

fieldname IN: Field to subset

ntype OUT: Number type of field

rank OUT: Rank of field

dims OUT: Dimensions of subset period

size OUT: Size in bytes of subset period

Purpose Retrieves information about the subsetted period.

Return value Returns SUCCEED (0) if successful and 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 *SWdeftimeperiod* for the *Spectra* 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*4	<i>swathid</i>
integer*4	<i>periodid</i>
character*(*)	<i>fieldname</i>
integer*4	<i>ntype</i>
integer*4	<i>rank</i>
integer*4	<i>dims</i> (*)
integer*4	<i>size</i>

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

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

Read Swath Attribute

SWreadattr

intn SWreadattr(int32 *swathID*, char **attrname*, VOIDP *datbuf*)

swathID IN: Swath id returned by SWcreate or SWattach

attrname IN: Attribute name

datbuf OUT: Buffer allocated to hold attribute values

Purpose Reads attribute from a swath.

Return value Returns SUCCEED (0) if successful and 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);
```

FORTTRAN integer function swrdattr(*swathid*,*attrname*,*datbuf*)

```
integer*4 swathid
```

```
character*(*) attrname
```

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

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

```
parameter (DFNT_FLOAT32=5)
```

```
status = swrdattr(swathid, "ScalarFloat", f32)
```


Read Data From a Swath Field

SWreadfield

intn SWreadfield(int32 *swathID*, char **fieldname*, int32 *start*[], int32 *stride*[], int32 *edge*[], VOIDP *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 and 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 values for <i>start</i> and <i>stride</i> are 0 and 1 respectively and are used if these parameters are set to <i>NULL</i> . The default values for <i>edge</i> are $(dim - start) / stride$ where <i>dim</i> refers is the size of the dimension.
Example	In this example, we read data from the 10th track (0-based) of the <i>Longitude</i> field.

```
float32 track[1000];

int32 start[2]={10,1}, edge[2]={1,1000};

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

FORTRAN integer function

swordfld(*swathid*, *fieldname*, *start*, *stride*, *edge*, *buffer*)

integer*4 *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 = swordfld(swathid, "Longitude", start, stride, edge,  
                 track);
```

Return Information About a Defined Region

SWregioninfo

intn SWregioninfo(int32 *swathID*, int32 *regionID*, char * *fieldname*, int32 **ntype*, int32 **rank*, int32 *dims*[], int32 **size*)

swathID IN: Swath id returned by SWcreate or SWattach

regionID IN: Region id returned by SWdefboxregion

fieldname IN: Field to subset

ntype OUT: Number type of field

rank OUT: Rank of field

dims OUT: Dimensions of subset region

size OUT: Size in bytes of subset region

Purpose Retrieves information about the subsetted region.

Return value Returns SUCCEED (0) if successful and 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 *SWdefboxregion* for the *Spectra* 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*4	<i>swathid</i>
integer*4	<i>regionid</i>
character*(*) <i>f</i>	<i>ieldname</i>
integer*4	<i>ntype</i>
integer*4	<i>rank</i>
integer*4	<i>dims</i> (*)
integer*4	<i>size</i>

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

intn SWsetfillvalue(int32 *swathID*, char **fieldname*, VOIDP *fillvalue*)

swathID IN: Swath id returned by SWcreate or SWattach

fieldname IN: Fieldname

fillvalue IN: Pointer to the fill value to be used

Purpose Sets fill value for the specified field.

Return value Returns SUCCEED (0) if successful and 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.

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

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

FORTTRAN integer function

```
swsetfill(swathid,fieldname,fillvalue)
```

```
integer*4    swathid
```

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

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

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

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

Write/Update Swath Attribute

SWwriteattr

intn SWwriteattr(int32 *swathID*, char **attrname*, int32 *ntype*, int32 *count*, VOIDP *datbuf*)

swathID IN: Swath id returned by SWcreate or SWattach

attrname IN: Attribute name

ntype IN: Number type of attribute

count IN: Number of values to store in attribute

datbuf IN: Attribute values

Purpose Writes/Updates attribute in a swath.

Return value Returns SUCCEED (0) if successful and 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:

```
f32 = 3.14;  
  
status = SWwriteattr(swathid, "ScalarFloat", DFNT_FLOAT32,  
                    1, &f32);
```

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

```
f32 = 3.14159;  
  
status = SWwriteattr(swathid, "ScalarFloat", DFNT_FLOAT32,  
                    1, &f32);
```

FORTRAN integer function `swrattr(swathid, attrname, ntype, count, datbuf)`
integer*4 *swathid*
character*(*) *attrname*
integer*4 *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 = swrattr(swathid, "ScalarFloat", DFNT_FLOAT32, 1,
f32)
```

Write Field Metadata for an Existing Swath Data Field

SWwritedatameta

intn SWwritedatameta(int32 *swathID*, char **fieldname*, char **dimlist*, int32 *numbertype*)

swathID IN: Swath id returned by SWcreate or SWattach

fieldname IN: Name of field

dimlist IN: The list of data dimensions defining the field

numbertype IN: The number type of the data stored in the field

Purpose Writes field metadata for an existing swath data field.

Return value Returns SUCCEED (0) if successful and FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list.

Description This routine writes field metadata for an existing data field. This is useful when the data field was defined without using the swath API. Note that any entries in the dimension list must be defined through the *SWdefdim* routine before this routine is called.

Example In this example we write the metadata for the “Band_1” data field used in the swath.

```
status = SWwritedatameta(swathID, "Band_1", "GeoTrack,  
GeoXtrack", DFNT_FLOAT32);
```

FORTRAN integer function

```
swwrdmeta(swathid,fieldname,dimlist,numbertype)
```

```
integer*4 swathid
```

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

```
character*(*) dimlist
```

```
integer*4 numbertype
```

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

```
status = swwrdmeta(swathID, "Band_1", "GeoXtrack, GeoTrack",  
DFNT_FLOAT32)
```

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

Write Data to a Swath Field

SWwritefield

intn SWwritefield(int32 *swathID*, char **fieldname*, int32 *start*[], int32 *stride*[], int32 *edge*[], VOIDP *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 and FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or unknown fieldname.

Description The values within *start*, *stride*, and *edge* arrays refer to the swath field (output) dimensions. The input data in the *data* buffer is read from contiguously. The default values for *start* and *stride* are 0 and 1 respectively and are used if these parameters are set to *NULL*. The default values for *edge* are $(dim - start) / stride$ where *dim* refers is the size of the dimension. 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 HDF routines.

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

```
float32 longitude [2000][1000];

/* Define elements of longitude array */

status = SWwritefield(swathID, "Longitude", NULL, NULL,
                     NULL, longitude);
```

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

```
float32 newtrack[1000];  
int32 start[2]={10,0}, edge[2]={1,1000};  
/* Define elements of newtrack array */  
status = SWwritefield(swathID, "Longitude", start, NULL,  
                    edge, newtrack);
```

FORTTRAN integer function
swwrfld(*swathid,fieldname,start,stride,edge,data*)
integer*4 *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 *FORTTRAN* 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 = swrfld(swathid, "Longitude", start, stride, edge,
newtrack)
```

Write Field Metadata to an Existing Swath Geolocation Field

SWwritegeometa

intn SWwritegeometa(int32 *swathID*, char **fieldname*, char **dimlist*, int32 *numbertype*)

swathID IN: Swath id returned by SWcreate or SWattach

fieldname IN: Name of field

dimlist IN: The list of geolocation dimensions defining the field

numbertype IN: The number type of the data stored in the field

Purpose Writes field metadata for an existing swath geolocation field.

Return value Returns SUCCEED (0) if successful and FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list.

Description This routine writes field metadata for an existing geolocation field. This is useful when the data field was defined without using the swath API. Note that any entries in the dimension list must be defined through the *SWdefdim* routine before this routine is called.

Example In this example we write the metadata for the "Latitude" geolocation field used in the swath.

```
status = SWwritegeometa(swathID, "Latitude",  
                        "GeoTrack,GeoXtrack", DFNT_FLOAT32);
```

FORTRAN integer function

```
swwrgmeta(swathid,fieldname,dimlist,numbertype)
```

```
integer*4 swathid
```

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

```
character*(*) dimlist
```

```
integer*4 numbertype
```

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

```
status = swwrgmeta(swathID, "Latitude",  
                  "GeoXtrack,GeoTrack", DFNT_FLOAT32)
```

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

2.1.3 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

int32 GDattach(int32 *fid*, char **gridname*)

fid IN: Grid file id returned by GDopen

gridname IN: Name of grid to be attached

Purpose Attaches to an existing grid within the file.

Return value Returns the grid handle(gridID) if successful and FAIL(-1) otherwise. Typical reasons for failure are improper grid file id or grid name.

Description This routine attaches to the grid using the *gridname* parameter as the identifier.

Example In this example, we attach to the previously created grid, "*ExampleGrid*", within the HDF file, *GridFile.hdf*, referred to by the handle, *fid*:

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

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

FORTTRAN integer*4 function gdattach(*fid*, *gridname*)

```
integer*4    fid
```

```
character*(*) gridname
```

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

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

Return Information About a Grid Attribute

GDattrinfo

intn GDattrinfo(int32 *gridID*, char **attrname*, int32 * *numbertype*, int32 **count*)

gridID IN: Grid id returned by GDcreate or GDattach

attrname IN: Attribute name

numbertype OUT: Number type of attribute

count OUT: Number of total bytes in attribute

Purpose Returns information about a grid attribute

Return value Returns SUCCEED (0) if successful and 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 *ScalarFloat* attribute.

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

The *nt* variable will have the value 5 and *count* will have the value 4.

FORTRAN integer function gdattrinfo(*gridid*, *attrname*, *ntype*, *count*,)

```
integer*4    gridid  
character*(*) attrname  
integer*4    ntype  
integer*4    count
```

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

```
status = gdattrinfo(pointid, "ScalarFloat",nt,count);
```

Close an HDF-EOS File

GDclose

intn GDclose(int32 *fid*)

fid IN: Grid file id returned by GDopen

Purpose Closes file.

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

Description This routine closes the HDF grid file.

Example

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

FORTRAN integer function gdclose(int32 *fid*)

```
integer*4 fid
```

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

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


Retreive Compression Information for Field

GDcompinfo

intn GDcompinfo(int32 *gridID*, char **fieldname*, int32 **compcode*, intn *compparm*[])

gridID IN: Grid id returned by GDcreate or GDattach

fieldname IN: Fieldname

compcode OUT: HDF compression code

compparm OUT: Compression parameters

Purpose Retrieves compression information about a field.

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

Description This routine returns the compression code and compression parameters for a given field.

Example To retrieve the compression information about the *Opacity* field defined in the *GDdefcomp* 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*4 *gridid*

character*(*) *fieldname*

integer*4 *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

int32 GDcreate(int32 *fid*, char **gridname*, int32 *xdimsize*, int32 *ydimsize*, float64 *upleftpt*[], float64 *lowrightpt*[])

fid IN: Grid file id returned by GDopen
gridname IN: Name of grid to be created
xdimsize IN: Number of columns in grid
ydimsize IN: Number of rows in grid
upleftpt IN: Location, of upper left corner of the upper left pixel
lowrightpt IN: Location, of lower right corner of the lower right pixel

Purpose Creates a grid within the file.

Return value Returns the grid handle(*gridID*) and FAIL(-1) otherwise.

Description The grid is created as a Vgroup within the HDF file with the name *gridname* and class *GRID*. 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 *upleftpt* and *lowrightpt* 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 equatorial 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 (DDDMMMSS.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 coordinates 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 is geographic (i.e., projcode=0) then the X-Y coordinates should be specified in degrees/minutes/seconds (DDDMMMSS.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*

FORTRAN integer*4 function gdcreate(*fid*, *gridname*, *xdimsize*, *ydimsiz*e, *upleftpt*,
lowrightpt)

integer*4 *fid*

character*(*) *gridname*

integer*4 *xdimsize*

integer*4 *ydimsiz*e

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 *upleft* and *lowrgt* arrays to 0.

Define Region of Interest by Latitude/Longitude

GDdefboxregion

int32 GDdefboxregion(int32 *gridID*, float64 *cornerlon*[], float64 *cornerlat*[])

gridID IN: Grid id returned by GDcreate or GDattach

cornerlon IN: Longitude in decimal degrees of box corners

cornerlat 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 and 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 *GDextractregion* 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(GDdid, cornerlon, cornerlat);
```

FORTTRAN integer*4 function gddefboxreg(*gridid*, *cornerlon*, *cornerlat*)

integer*4 *gridid*

real*8 *cornerlon*

real*8 *cornerlat*

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

```
cornerlon(1) = 0.
cornerlat(2) = 90.
cornerlon(1) = 90.
cornerlat(2) = 0.

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

Set Grid Field Compression

GDdefcomp

intn GDdefcomp(int32 *gridID*, int32 *compcode*, intn *compparm*[])

gridID IN: Grid id returned by GDcreate or GDattach

compcode IN: HDF compression code

compparm IN: Compression parameters (if applicable)

Purpose Sets the field compression for all subsequent field definitions.

Return value Returns SUCCEED(0) if successful and 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 schemes currently supported are: run length encoding (HDFE_COMP_RLE = 1), skipping Huffman (HDFE_COMP_SKPHUFF = 3), deflate (gzip) (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 *GDwritefield* routine, however, the entire field must be written in a single call. If this is not possible, the user should consider tiling. See *GDdeftile* for further information. Any portion of a compressed field can then be accessed with the *GDreadfield* 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 *Pressure* using run length encoding, the *Opacity* field using deflate compression, the *Spectra* field with skipping Huffman compression, and use no compression for the *Temperature* field.

```
status = GDdefcomp(gridID, HDFE_COMP_RLE, NULL);

status = GDdeffield(gridID, "Pressure", "YDim,XDim",
DFNT_FLOAT32, HDFE_NOMERGE);

compparm[0] = 5;

status = GDdefcomp(gridID, HDFE_COMP_DEFLATE, compparm);
```

```

status = GDdeffield(gridID, "Opacity", "YDim,XDim",
DFNT_FLOAT32, HDFE_NOMERGE);

status = GDdefcomp(gridID, HDFE_COMP_SKPHUFF, NULL);

status = GDdeffield(gridID, "Spectra", "Bands,YDim,XDim",
DFNT_FLOAT32, HDFE_NOMERGE);

status = GDdefcomp(gridID, HDFE_COMP_NONE, NULL);

status = GDdeffield(gridID, "Temperature", "YDim,XDim",
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*4   gridid
integer     compcode
integer     compparm

```

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

```

parameter (HDFE_COMP_NONE=0)
parameter (HDFE_COMP_RLE=1)
parameter (HDFE_COMP_SKPHUFF=3)
parameter (HDFE_COMP_DEFLATE=4)

integer   compparm(5)

status = gddefcomp(gridid, HDFE_COMP_RLE, compparm)
status = gddeffld(gridid, "Pressure", "YDim,XDim",
DFNT_FLOAT32, HDFE_NOMERGE)

compparm(1) = 5

status = gddefcomp(gridid, HDFE_COMP_DEFLATE, compparm)
status = gdeffld(gridid, "Opacity", "YDim,XDim",
DFNT_FLOAT32, HDFE_NOMERGE)

status = gddefcomp(gridid, HDFE_COMP_SKPHUFF, compparm)
status = gddeffld(gridid, "Spectra", "Bands,YDim,XDim",
DFNT_FLOAT32, HDFE_NOMERGE)

status = gddefcomp(gridid, HDFE_COMP_NONE, compparm)

status = gddeffld(gridid, "Temperature", "YDim,XDim",
DFNT_FLOAT32, HDFE_AUTOMERGE)

```

Define a New Dimension Within a Grid

GDdefdim

intn GDdefdim(int32 *gridID*, char **dimname*, int32 *dim*)

gridID IN: Grid id returned by GDcreate or GDattach

dimname IN: Name of dimension to be defined

dim IN: The size of the dimension

Purpose Defines a new dimension within the grid.

Return value Returns SUCCEED(0) if successful and 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, *Band*, 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 zero or equivalently, *SD_UNLIMITED*:

```
status = GDdefdim(gridID, "Unlim", SD_UNLIMITED);
```

FORTRAN integer function gddefdim(*gridid*, *fieldname*, *dim*)

```
integer*4    gridid
```

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

```
integer*4    dim
```

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

```
parameter (SD_UNLIMITED=0)
```

```
status = gddefdim(gridid, "Band", 15)
```

```
status = gddefdim(gridid, "Unlim", SD_UNLIMITED)
```


Define a New Data Field Within a Grid

GDdefield

intn GDdefield(int32 *gridID*, char **fieldname*, char **dimlist*, int32 *numbertype*, int32 *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>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 grid.

Return value Returns SUCCEED(0) if successful and 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.

Example In this example, we define a grid field, *Temperature* with dimensions *XDim* and *YDim* (as established by the *GDcreate* routine) containing 4-byte floating point numbers and a field, *Spectra*, with dimensions *XDim*, *YDim*, and *Bands*:

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

```
status = GDdefield(gridID, "Spectra", "Bands,YDim,XDim",  
DFNT_FLOAT32, HDFE_NOMERGE);
```

FORTRAN integer function `gddeffld(gridid, fieldname, dimlist, numbertype, merge)`
 integer*4 `gridid`
 character*(*) `fieldname`
 character*(*) `dimlist`
 integer*4 `numbertype`
 integer*4 `merge`

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

```

parameter (DFNT_FLOAT32=5)
parameter (HDFE_NOMERGE=0)
parameter (HDFE_AUTOMERGE=1)

status = gddeffld(gridid, "Temperature", "XDim,YDim",
DFNT_FLOAT32, DFE_AUTOMERGE)

status = gddeffld(gridid, "Spectra", "XDim,YDim,Bands",
DFNT_FLOAT32, HDFE_NOMERGE)

```

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

Define the Origin of the Grid Data

GDdeforigin

intn GDdeforigin(int32 *gridID*, int32 *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 and 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(Default)(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*4    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

intn GDdefpixreg(int32 *gridID*, int32 *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 and 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);
```

FORTTRAN integer function gddefpixreg(*gridid*, *pixreg*)

```
integer*4    gridid
```

```
integer*4    pixreg
```

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

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

```
status = gddefpixreg(GDdid, HDFE_CORNER)
```

Define Grid Projection

GDdefproj

intn GDdefproj(int32 *gridID*, int32 *projcode*, int32 *zonecode*, int32 *spherecode*, 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

spherecode IN: GCTP spheroid code

projparm IN: GCTP projection parameter array

Purpose Defines projection of grid

Return Value Returns SUCCEED(0) if successful and 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), spherecode = 0

```
spherecode = 0;
```

```
zonecode = 40;
```

```
status = GDdefproj(gridID, GCTP_UTM, zonecode, spherecode, 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.

```
spherecode = 3;
```

```
for (i = 0; i < 13; i++) projparm[i] = 0;
```

```
/* Set Long below pole & true scale in DDDMMMSS.SSS form */
```

```
projparm[5] = 90000000.00;
```

```
status = GDdefproj(gridID, GCTP_PS, NULL, spherecode, 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*4    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

intn GDdeftile(int32 *gridID*, int32 *tilecode*, int32 *tilerank*, int32 *tiledims*[])

gridID IN: Grid id returned by GDcreate or GDattach

tilecode IN: Tile code: HDF_TILE, HDF_NOTILE (default)

tilerank IN: The number of tile dimensions

tiledims IN: Tile dimensions

Purpose Defines tiling dimensions for subsequent field definitions

Return Value Returns SUCCEED(0) if successful and 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 *GDdefcomp*. 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 *XDim* is 200 and *YDim* is 300.

```
tiledims[0] = 100;
tiledims[1] = 200;

status = GDdeftile(gridID, HDFE_TILE, 2, tiledims);

status = GDdeffield(gridID, "Pressure", "YDim,XDim",
DFNT_INT16, HDFE_NOMERGE);

status = GDdeffield(gridID, "Temperature", "YDim,XDim",
DFNT_FLOAT32, HDFE_NOMERGE);

tiledims[0] = 1;
tiledims[1] = 150;
tiledims[2] = 100;

status = GDdeftile(gridID, HDFE_TILE, 3, tiledims);
```

```

status = GDdefield(gridID, "Spectra", "Bands,YDim,XDim",
DFNT_FLOAT32, HDFE_NOMERGE);

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

status = GDdefield(gridID, "Communities", "YDim,XDim",
DFNT_INT32, HDFE_AUTOMERGE);

```

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

```

integer*4    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_TILE=1)

tiledims(1) = 200

tiledims(2) = 100

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

status = gddeffld(gridid, 'Pressure', 'XDim,YDim',
DFNT_INT16, HDFE_NOMERGE)

status = gddeffld(gridid, 'Temperature', 'XDim,YDim',
DFNT_FLOAT32, 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',
DFNT_FLOAT32, HDFE_NOMERGE)

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

status = gddeffld(gridid, 'Communities', 'XDim,YDim',
DFNT_INT32, HDFE_AUTOMERGE)

```


Define a Time Period of Interest

GDdeftimeperiod

int32 GDdeftimeperiod(int32 *gridID*, int32 *periodID*, float64 *starttime*, float64 *stoptime*)

gridID IN: Grid id returned by GDcreate or GDattach

periodID IN: Period (or region) id from previous subset call

starttime IN: Start time of period

stoptime IN: Stop time of period

Purpose Defines a time period for a grid.

Return value Returns the grid period ID if successful and FAIL (-1) otherwise.

Description This routine defines a time period for a grid. It returns a grid period ID which is used by the *GDextractperiod* routine to read all the entries of a data field within the time period.. The grid structure must have the *Time* field defined. This routine may be called after *GDdefboxregion* 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 *GDdefvrtregion* 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*4 function gddefmeper(*gridid*, *periodID*, *starttime*, *stoptime*)
integer*4 *gridid*
integer*4 *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 = gddefmeper(swathid, HDFE_NOPREVSUB, starttime,
stoptime)

periodid = gddefmeper(swathid, regionid, starttime, stoptime)
```

Define a Vertical Subset Region

GDdefvrtregion

int32 GDdefvrtregion(int32 *gridID*, int32 *regionID*, char **vertObj*, float64 *range*[])

gridID IN: Grid id returned by GDcreate or GDattach

regionID IN: Region (or period) id from previous subset call

vertObj IN: Dimension or field to subset

range IN: Minimum and maximum range for subset

Purpose Subsets on a **monotonic** field or contiguous elements of a dimension.

Return value Returns the grid region ID if successful and FAIL (-1) otherwise.

Description Whereas the *GDdefboxregion* routine subsets along the *XDim* and *YDim* 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 **monotonic** (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 *GDdefboxregion* 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 *GDdefvrtregion* (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 = GDdefvrtregion(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 = GDdefvrtregion(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 = GDdefvrtregion(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*4 function gddefvrtreg(*gridid*, *regionid*, *vertobj*, *range*)
integer*4 *gridid*
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 = gddefvrtreg(gridid, HDFE_NOPREVSUB, "Pressure",
range)
range(1) = 3      ! Note 1-based element numbers
range(2) = 6
regionid = gddefvrtreg(gridid, HDFE_NOPREVSUB, "DIM:Height",
range)
regionid = gddefvrtreg(gridid, subsetid, "Pressure", range)
```

Detach from Grid Structure

GDdetach

intn GDdetach(int32 *gridID*)

gridID IN: Grid id returned by GDcreate or GDattach

Purpose Detaches from grid interface.

Return value Returns SUCCEED(0) if successful and 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)
```

FORTTRAN integer function gddetach(gridid)

```
integer*4      gridid
```

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

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

Retrieve Size of Specified Dimension

GDdiminfo

int32 GDdiminfo(int32 *gridID*, char **dimname*)

gridID IN: Grid id returned by GDcreate or GDattach

dimname IN: Dimension name

Purpose Retrieve size of specified dimension.

Return value Size of dimension. If FAIL(-1), could signify 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

FORTTRAN integer*4 function gddiminfo(*gridid*,*dimname*)

```
integer*4    gridid
```

```
character*(*) dimname
```

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

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

Duplicate a Region or Period

GDdupregion

int32 GDdupregion(int32 *regionID*)

pointID IN: Region or period id returned by GDdefboxregion, GDdeftimeperiod, or GDdefvrtregion.

Purpose Duplicates a region.

Return value Returns new region or period ID.

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 = GDdefvrtregion(gridID, regionID, "Pressure",  
rangePres);  
  
regionID2 = GDdefvrtregion(gridID, regionID2, "Temperature",  
rangeTemp);
```

FORTRAN integer*4 gddupreg(*regionid*)

integer*4 *regionid*

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

```
regionid = gdddefboxreg(gridid, cornerlon, cornerlat)  
regionid2 = gddupreg(regionid)  
  
regionid = gdddefvrtreg(gridid, regionid, 'Pressure',  
rangePres)  
  
regionid2 = gdddefvrtreg(gridid, regionid2, 'Temperature',  
rangeTemp)
```


Read a Region of Interest from a Field

GDextractregion

intn GDextractregion(int32 *gridID*, int32 *regionID*, char **fieldname*, VOIDP *buffer*)

gridID IN: Grid id returned by GDcreate or GDattach
regionID IN: Region (period) id returned by GDdefboxregion (GDdeftimeperiod)
fieldname IN: Field to subset
buffer OUT: Data Buffer

Purpose Extracts (reads) from subsetted region.
Return value Returns SUCCEED (0) if successful and 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 *GDdefboxregion*. 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);
```

FORTTRAN integer*4 function gdextreg(*gridid*, *regionid*, *fieldname*, *datbuf*)

```
integer*4    gridid  
integer*4    regionid  
character*(*) fieldname  
<valid type> buffer(*)
```

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

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

Retrieve Information About Data Field in a Grid

GDfieldinfo

intn GDfieldinfo(int32 *gridID*, char **fieldname*, int32 *rank*, int32 *dims*[], int32 **numbertype*, char **dimlist*)

gridID IN: Grid id returned by GDcreate or GDattach

fieldname IN: Fieldname

rank OUT: Pointer to rank of the field

dims OUT: Array containing the dimension sizes of the field

numbertype OUT: Pointer to the numbertype of the field

dimlist OUT: Dimension list

Purpose Retrieve information about a specific geolocation or data field in the grid.

Return value Returns SUCCEED(0) if successful and FAIL(-1) if 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 *Spectra* data fields:

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

The return parameters will have the following values:

```
rank=3, numbertype=5, dims[3]={15,200,120} and  
dimlist="Bands,YDim,XDim"
```

FORTRAN integer function *gdflinfo*(*gridid*, *fieldname*, *rank*, *dims*, *numbertype*, *dimlist*)

integer*4 *gridid*

character*(*) *fieldname*

integer*32 *rank*

integer*4 *dims*(*)

integer*4 *numbertype*

character*(*) *dimlist*

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

```
status = gdflinfo(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

intn GDgetfillvalue(int32 *gridID*, char **fieldname*, VOIDP *fillvalue*)

gridID IN: Grid id returned by GDcreate or GDattach

fieldname IN: Fieldname

fillvalue OUT: Space allocated to store the fill value

Purpose Retrieves fill value for the specified field.

Return value Returns SUCCEED(0) if successful and 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);
```

FORTTRAN integer function

```
gdgetfill(gridid,fieldname,fillvalue)
```

```
integer*4 gridid
```

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

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

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

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

Get Row/Columns for Specified Longitude/Latitude Pairs

GDgetpixels

intn GDgetpixels(int32 *gridID*, int32 *nLonLat*, float64 *lonVal*[], float64 *latVal*[], int32 *pixRow*[], int32 *pixCol*[])

gridID IN: Grid id returned by GDcreate or GDattach

nLonLat IN: Number of longitude/latitude pairs

lonVal IN: Longitude values in degrees

latVal IN: Latitude values in degrees

lonVal OUT: Pixel Rows

pixCol OUT: Pixel Columns

Purpose Returns the pixel rows and columns for specified longitude/latitude pairs.

Return value Returns SUCCEED(0) if successful and 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 *GDdefboxregion*.

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.

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

The equivalent *FORTTRAN* 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.

Get Field Values for Specified Row/Columns

GDgetpixvalues

int32 GDgetpixvalues(int32 *gridID*, int32 *nPixels*, int32 *pixRow*[], int32 *pixCol*[],
char **fieldname*, VOIDP *buffer*)

gridID IN: Grid id returned by GDcreate or GDattach

nPixels IN: Number of pixels

lonVal IN: Pixel Rows

pixCol IN: Pixel Columns

fieldname IN: Field from which to extract data values

buffer OUT: Buffer for data values

Purpose Read field data values for specified pixels.

Return value Returns size of data buffer if successful and FAIL(-1) otherwise.

Description This routine reads data from a data field for the specified pixels. It is the pixel subsetting equivalent of *GDextractregion*. All entries along the non-geographic dimensions (ie, NOT *XDim* and *YDim*) are returned. If the buffer is set to NULL, no data is returned but the data buffer size can be determined from the function return value.

Example To read values from the *Spectra* field with dimensions, *Bands*, *YDim*, and *XDim*, make the following call:

```
float64      *datbuf;

bufsiz = GDgetpixvalues(gridID, 2, rowArr, colArr,
    "Spectra", NULL);

/* bufsiz will be equal to 2 * NBANDS * 8 where NBANDS is
the value for the Bands dimension */

datbuf = (float64 *) malloc(bufsiz);

bufsiz = GDgetpixvalues(gridID, 2, rowArr, colArr,
    "Spectra", datbuf);
```

FORTRAN integer*4 function gdgetpixval(*gridid*, *npixels*, *pixrow*, *pixcol*, *fieldname*,
buffer)
integer*4 *gridid*
integer*4 *nlonlat*
integer*4 *pixrow*
integer*4 *pixcol*
character*(*) *fieldname*
<valid type> *buffer*(*)

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

```
real*8      datbuf(2,NBANDS)  
  
bufsiz = gdgetpixval(gridid, 2, rowarr, colarr, "Spectra",  
datbuf)
```


Return Information About a Grid Structure

GDgridinfo

intn GDgridinfo(int32 *gridID*, int32 **xdimension*, int32 **ydimension*, float64 *upleft*[], float64 *lowright*[])

gridID IN: Grid id returned by GDcreate or GDattach

xdimension OUT: Number of columns in grid

ydimension OUT: Number of rows in grid

upleft OUT: Location, in meters, of upper left corner

lowright OUT: Location, in meters, of lower right corner

Purpose Returns position and size of grid

Return value Returns SUCCEED(0) if successful and 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, &xdimension, &ydimension, upleft, lowright);
```

FORTTRAN integer function gdgridinfo(*gridid*, *xdimension*, *ydimension*, *upleft*, *lowright*)

integer*4 *gridid*

integer*4 *xdimension*

integer*4 *ydimension*

real*8 *upleft*(*)

real*8 *lowright*(*)

The equivalent FORTRAN code for the example above is:

```
status = gdgridinfo(gridid, xdimension, ydimension, upleft, lowright);
```

Retrieve Information About Grid Attributes

GDinqattrs

int32 GDinqattrs(int32 *gridID*, char **attrlist*, int32 **strbufsize*)

gridID IN: Grid id returned by GDcreate or GDattach

attrlist OUT: Attribute list (entries separated by commas)

strbufsize OUT: String length of attribute list

Purpose Retrieve information about attributes defined in grid.

Return value Number of attributes found or (-1) if failure.

Description The attribute list is returned as a string with each attribute name separated by commas. If *attrlist* is set to NULL, then the routine will return just the string buffer size, *strbufsize*. 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".
```

FORTTRAN integer*4 function gdingattrs(*gridid*,*attrlist*,*strbufsize*)

integer*4 *gridid*

character*(*) *attrlist*

integer*4 *strbufsize*

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

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

Retrieve Information About Dimensions Defined in Grid

GDinqdims

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

gridID IN: Grid id returned by GDcreate or GDattach

dimname OUT: Dimension list (entries separated by commas)

dims OUT: Array containing size of each dimension

Purpose Retrieve information about dimensions defined in grid.

Return value Number of dimension entries found. If FAIL(-1), could signify an improper grid id.

Description The dimension list is returned as a string with each dimension name separated by commas. Output parameters set to *NULL* 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}

FORTTRAN integer*4 function gdingdims(*gridid*,*dimname*,*dims*)

integer*4 *gridid*

character*(*) *dimname*

integer*4 *dims*(*)

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

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

Retrieve Information About Data Fields Defined in Grid

GDinqfields

int32 GDinqfields(int32 *gridID*, char **fieldlist*, int32 *rank*[], int32 *numbertype*[])

gridID IN: Grid id returned by GDcreate or GDattach

fieldlist OUT: Listing of data fields (entries separated by commas)

rank OUT: Array containing the rank of each data field

numbertype OUT: Array containing the numbertype of each data field

Purpose Retrieve information about the data fields defined in grid.

Return value Number of data fields found. If FAIL(-1), could signify an improper grid id.

Description The field list is returned as a string with each data field separated by commas. The *rank* and *numbertype* arrays will have an entry for each field. Output parameters set to *NULL* 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]={5,5}
```

FORTRAN integer*4 function gdingflds(*gridid*, *fieldlist*, *rank*, *numbertype*)

integer*4 *gridid*

character*(*) *fieldlist*

integer*4 *rank*(*)

integer*4 *numbertype*(*)

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

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

The parameter, *fieldlist*, will have the value: "Spectra,Temperature"
with *rank*[2]={3,2}, *numbertype*[2]={5,5}

Retrieve Grid Structures Defined in HDF-EOS File

GDinqgrid

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

gridID IN: HDF-EOS filename

gridlist OUT: Grid list (entries separated by commas)

strbufsize 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 (-1) if failure.

Description The grid list is returned as a string with each grid name separated by commas. If *gridlist* is set to NULL, then the routine will return just the string buffer size, *strbufsize*. If *strbufsize* is also set to NULL, the routine returns just the number of grids. Note that *strbufsize* does not count the null string terminator.

Example In this example, we retrieve information about the grids defined in an HDF-EOS file, *HDFEOS.hdf*. We assume that there are two grids stored, *GridOne* and *Grid_2*:

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

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

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

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

“*GridOne,Grid_2*”.

FORTTRAN integer*4 function gdingqgrid(*filename,gridlist,strbufsize*)

character*(*) *filename*

character*(*) *gridlist*

integer*4 *strbufsize*

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

```
ngrid = gdingqgrid('HDFEOS.hdf', gridlist, strbufsize)
```

Perform Bilinear Interpolation on Grid Field

GDinterpolate

int32 GDinterpolate(int32 *gridID*, int32 *nInterp*, float64 *lonVal*[], float64 *latVal*[], char **fieldname*, float64 *interpVal*[])

gridID IN: Grid id returned by GDcreate or GDattach

nInterp IN: Number of interpolation points

lonVal IN: Longitude of interpolation points

latVal IN: Latitude of interpolation points

fieldname OUT: Field from which to interpolate data values

interpVal OUT: Buffer for interpolated data values

Purpose Performs bilinear interpolation on a grid field.

Return value Returns size in bytes of interpolated data values if successful and FAIL(-1) otherwise.

Description This routine performs bilinear interpolation on a grid field. It assumes that the pixel data values are uniformly spaced which is strictly true only for an infinitesimally small region of the globe but is a good approximation for a sufficiently small region. The default position of the pixel value is pixel center, however if the pixel registration has been set to HDFE_CORNER (with the *GDdefpixreg* routine) then the value is located at one of the four corners (HDFE_GD_UL, _UR, _LL, _LR) specified by the *GDdeforigin* routine. All entries along the non-geographic dimensions (ie, NOT *XDim* and *YDim*) are interpolated and all interpolated values are returned as FLOAT64. The data buffer size can be determined by setting the *interpVal* parameter to NULL. The reference for the interpolation algorithm is *Numerical Recipes in C* (2nd ed). (Note for the current version of this routine, the number type of the field to be interpolated is restricted to INT16, INT32, FLOAT32, FLOAT64.)

Example To interpolate the *Spectra* field at two geographic data points:

```
lonVal[0] = 134.2;
```

```
latVal[0] = -20.8;
```

```
lonVal[1] = 15.8;
```

```
latVal[1] = 84.6;
```

```

float64      *interVal;

bufsiz = GDinterpolate(gridID, 2, lonVal, latVal, "Spectra",
NULL);

/* bufsiz will be equal to 2 * NBANDS * 8 where NBANDS is
the value for the Bands dimension */

interpVal = (float64 *) malloc(bufsiz);

bufsiz = GDinterpolate(gridID, 2, lonVal, latVal, "Spectra",
interpVal);

```

FORTRAN integer*4 function gdinterpolate(*gridid, ninterp, lonval, latval, fieldname, interpval*)

```

integer*4    gridid
integer*4    ninterp
real*8       lonval
real*8       latval
character*(*) fieldname
real*8       interpval

```

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

```

real*8       interpval(NBANDS, 2)

bufsiz = gdinterpolate(gridid, 2, lonval, latval, "Spectra",
interpval)

```

Return Number of Specified Objects in a Grid

GDnentries

int32 GDnentries(int32 *gridID*, int32 *entrycode*, int32 **strbufsize*)

gridID IN: Grid id returned by GDcreate or GDattach

entrycode IN: Entrycode

strbufsize OUT: String buffer size

Purpose Returns number of entries and descriptive string buffer size for a specified entity.

Return value Number of entries. If FAIL(-1), could signify 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 gdnentries(*gridid*,*enyrtrcode*, *bufsize*)

integer*4 *gridid*

integer*4 *entrycode*

integer*4 *bufsize*

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

```
ndims = gdnentries(gridid, 4, bufisz)
```


Open HDF-EOS File

GDopen

int32 GDopen(char *filename, intn access)

filename IN: Complete path and filename for the file to be opened
access IN: DFACC_READ, DFACC_RDWR or DFACC_CREATE
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 and FAIL(-1) otherwise.
Description This routine creates a new file or opens an existing one, depending on the access parameter.

Access codes:

DFACC_READ Open for read only. If file does not exist, error
DFACC_RDWR Open for read/write. If file does not exist, create it
DFACC_CREATE If file exist, delete it, then open a new file for read/write

Example In this example, we create a new grid file named, *GridFile.hdf*. It returns the file handle, *fid*.

```
fid = GDopen("GridFile.hdf", DFACC_CREATE);
```

FORTRAN integer*4 function gdopen(filename, access)

character*(*) filename

integer access

The access codes should be defined as parameters:

parameter (DFACC_READ=1)

parameter (DFACC_RDWR=3)

parameter (DFACC_CREATE=4)

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

```
fid = gdopen("GridFile.hdf", DFACC_CREATE)
```

Note to users of the SDP Toolkit: Please refer to the *SDP Toolkit 5 User Guide for the ECS Project* (333-CD-003-002), Section 6.2.1.2 for information on how to obtain a file name (referred to as a physical file handle) from within a PGE. See also Section 9 of this document for code examples.

Return Grid Origin Information

GDorigininfo

intn GDorigininfo(int32 *gridID*, int32 **origincode*)

gridID IN: Grid id returned by GDcreate or GDattach

origincode IN: Origin code

Purpose Retrieve origin code.

Return value Origin code.

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*4 *gridid*

integer*4 *origincode*

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

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

Return Pixel Registration Information

GDpixreginfo

intn GDpixreginfo(int32 *gridID*, int32 **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.

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

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

```
integer*4    gridid
```

```
integer*4    pixregcode
```

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

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

Retrieve Grid Projection Information

GDprojinfo

intn GDprojinfo(int32 *gridID*, int32 **projcode*, int32 **zonecode*, int32 **spherecode*, float64 *projparm*[])

gridID IN: Grid id returned by GDcreate or GDattach

projcode OUT: GCTP projection code

zonecode OUT: GCTP zone code used by UTM projection

spherecode OUT: GCTP spheroid code

projparm OUT: GCTP projection parameter array

Purpose Retrieves projection information of grid

Return Value Returns SUCCEED(0) if successful and 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,
&spherecode, projparm);
```

FORTRAN integer function gdprojinfo(*gridid*, *projcode*, *zonecode*, *spherecode*,
projparm)

integer*4 *gridid*

integer*4 *projcode*

integer*4 *zonecode*

integer*4 *spherecode*

real*8 *projparm*(*)

The equivalent FORTRAN code for the example above is:

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

Read Grid Attribute

GDreadattr

intn GDreadattr(int32 *gridID*, char **attrname*, VOIDP *datbuf*)

gridID IN: Grid id returned by GDcreate or GDattach

attrname IN: Attribute name

datbuf OUT: Buffer allocated to hold attribute values

Purpose Reads attribute from a grid.

Return value Returns SUCCEED(0) if successful and 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 gdrdatr(*gridid*, *attrname*, *datbuf*)

```
integer*4    gridid
```

```
character*(*) attrname
```

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

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

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

Read Data From a Grid Field

GDreadfield

intn GDreadfield(int32 *gridID*, char **fieldname*, int32 *start*[], int32 *stride*[], int32 *edge*[], VOIDP *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 and FAIL(-1) otherwise. Typical reasons for failure are improper grid id of 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 values for <i>start</i> and <i>stride</i> are 0 and 1 respectively and are used if these parameters are set to <i>NULL</i> . The default values for <i>edge</i> are $(dim - start) / stride$ where <i>dim</i> refers to the size of the dimension.
Example	In this example, we read data from the 10th row (0-based) of the <i>Temperature</i> field.

```
float32 row[120];

int32 start[2]={10,1}, edge[2]={1,120};

status = GDreadfield(gridID, "Temperature", start, NULL,
edge, row);
```

FORTRAN integer function
gdrdfld(gridid,fieldname,start,stride,edge,buffer)
 integer*4 *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);
  
```

Read from Tile within Field

GDreadtile

intn GDreadtile(int32 *gridID*, char **fieldname*, int32 *tilecoords*[], VOIDP *buffer*)

gridID IN: Grid id returned by GDcreate or GDattach

fieldname IN: Fieldname

tilecoords IN: Array of tile coordinates

buffer OUT: Data to be written to tile

Purpose Reads from tile within field.

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

Description This routine reads a single tile of data from a field. If the data is to be read tile by tile, this routine is more efficient than *GDreadfield*. In all other cases, the later routine should be used. *GDreadtile* does not work on non-tiled fields. Note that the coordinates in terms of tiles, not data elements.

Example In this example. we read one tile from the *Temperature* field (see *GDdeftile* example) located at the second column of the first row of tiles. *Buffer* should contain space for $200 * 100 * 4 = 80000$ bytes.

```
tilecoords[0] = 0;
tilecoords[1] = 1;
status = GDreadtile(gridid, "Temperature", tilecoords,
buffer);
```

FORTRAN integer function gdrdtle(*gridid*, *fieldname*, *tilecoords*, *buffer*)

integer*4 *gridid*

character*(*) *fieldname*

integer*4 *tilecoords*(*)

<valid type> *buffer*(*)

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

```
tilecoords(1) = 1;
tilecoords(2) = 0;
status = gdrdtle(gridid, "Temperature", tilecoords, buffer)
```

Note that *tilecoords* for *FORTRAN* are reversed from the C language example but the values are still 0-based.

Return Information About a Region

GDregioninfo

intn GDregioninfo(int32 *gridID*, int32 *regionID*, char * *fieldname*, int32 **ntype*, int32 **rank*, int32 *dims*[], int32 **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: Number type 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 and 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(GDdid, regionID, "Temperature", &ntype,
                    &rank, dims, &size, upleft, lowright);
```

FORTRAN integer function *gdreginfo*(*gridid*, *regionid*, *fieldname*, *ntype*, *rank*, *dims*,
size, *upleftpt*, *lowrightpt*)

integer*4 *gridid*

integer*4 *gridid*

character*(*) *fieldname*

integer*4 *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

intn GDsetfillvalue(int32 *gridID*, char **fieldname*, VOIDP *fillvalue*)

gridID IN: Grid id returned by GDcreate or GDattach

fieldname IN: Fieldname

fillvalue IN: Pointer to the fill value to be used

Purpose Sets fill value for the specified field.

Return value Returns SUCCEED(0) if successful and 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);
```

FORTTRAN integer function

```
gdsetfill(gridid,fieldname,fillvalue)
```

```
integer*4 gridid
```

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

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

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

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

Set Tile Cache Parameters

GDsettilecache

intn GDsettilecache(int32 *gridID*, char **fieldname*, int32 *maxcache*, int32 *cachecode*)

gridID IN: Grid id returned by GDcreate or GDattach
fieldname IN: Fieldname
maxcache IN: Maximum number of tiles to cache in memory
cachecode IN: Currently must be set to 0

Purpose Sets tile cache parameters

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

Description This routine sets the maximum cache for tiling. If the cache is set for a fewer number of tiles than needed for a particular subset of the field, there can be serious efficiency problems. Therefore it is recommended that this routine not be used unless one is aware for each field, the expected size of a particular subset and its position relative to the tiles. The *maxcache* value should be set to the number of tiles which fit along the fastest varying dimension.

Example In this example, we set *maxcache* to 10 tiles. The particular subsetting envisioned for the *Spectra* field (defined in the *GDdefile* example) would never cross more than 10 tiles along the field's fastest varying dimension, ie, *XDim*..

```
status = GDsettilecache(gridID, "Spectra", 10, 0);
```

FORTRAN integer function gdsettleche(*gridid*, *fieldname*, *maxcache*, *cachecode*)

```
integer*4    gridid  
character*(*) fieldname  
integer*4    maxcache  
integer*4    cachecode
```

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

```
status = gdsettleche(gridid, 'Spectra', 10, 0)
```

Retreive Tiling Information for Field

GDtileinfo

intn GDtileinfo(int32 *gridID*, char **fieldname*, int32 **tilecode*, int32 **tilerank*,
int32 *tiledims*[*I*])

gridID IN: Grid id returned by GDcreate or GDattach

fieldname IN: Fieldname

tilecode OUT: Tile code: HDF_TILE, HDF_NOTILE

tilerank OUT: The number of tile dimensions

tiledims OUT: Tile dimensions

Purpose Retrieves tiling information about a field.

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

Description This routine returns the tiling code, tiling rank, and tiling dimensions for a given field.

Example To retrieve the tiling information about the *Pressure* field defined in the *GDdefile* section:

```
status = GDcompinfo(gridID, "Pressure", &tilecode,  
&tilerank, tiledims);
```

The *tilecode* parameter will be set to 1, the *tilerank* to 2, and *tiledims* to {100,200}.

FORTTRAN integer function gdtileinfo(*gridid*,*fieldname* *tilecode*,*tilerank*,*tiledims*)

integer*4 *gridid*

character*(*) *fieldname*

integer*4 *tilecode*

integer*4 *tilerank*

integer*4 *tiledims*(*)

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

```
status = gdtileinfo(gridid, 'Pressure', tilecode, tilerank,  
tiledims)
```

The *tilecode* parameter will be set to 1, the *tilerank* to 2, and *tiledims* to {200,100}.

Write/Update Grid Attribute

GDwriteattr

intn GDwriteattr(int32 *gridID*, char **attrname*, int32 *ntype*, int32 *count*, VOIDP *datbuf*)

gridID IN: Grid id returned by GDcreate or GDattach

attrname IN: Attribute name

ntype IN: Number type of attribute

count IN: Number of values to store in attribute

datbuf IN: Attribute values

Purpose Writes/Updates attribute in a grid.

Return value Returns SUCCEED(0) if successful and 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", DFNT_FLOAT32, 1,  
                    &f32);
```

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

```
f32 = 3.14159;  
status = GDwriteattr(gridid, "ScalarFloat", DFNT_FLOAT32, 1,  
                    &f32);
```

FORTRAN integer function *gdwrattr*(*gridid*, *attrname*,
ntype, *count*, *datbuf*)

integer*4 *gridid*

character*(*) *attrname*

integer*4 *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 = gdwrattr(gridid, "ScalarFloat", DFNT_FLOAT32, 1,
                 f32)
```

Write Data to a Grid Field

GDwritefield

intn GDwritefield(int32 *gridID*, char **fieldname*, int32 *start*[], int32 *stride*[], int32 *edge*[], VOIDP *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 and 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 values for <i>start</i> and <i>stride</i> are 0 and 1 respectively and are used if these parameters are set to <i>NULL</i> . The default values for <i>edge</i> are $(dim - start) / stride$ where <i>dim</i> refers to the size of the dimension. 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>GDdefile</i> for further information.

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

```
float32 temperature [200][120];
/* Define elements of temperature array */
status = GDwritefield(gridID, "Temperature", NULL, NULL,
    NULL, temperature);
```

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

```
float32 newrow[2000];
int32 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*4 *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)
```

Write Field Metadata for an Existing Field Not Defined With the Grid API

GDwritefieldmeta

intn GDwritefieldmeta(int32 *gridID*, char **fieldname*, char **dimlist*, int32 *numbertype*)

gridID IN: Grid id returned by GDcreate or GDattach

fieldname IN: Name of field that metadata information is to be written

dimlist IN: Dimension list of field

numbertype IN: Number type of data in field

Purpose Writes field metadata for an existing grid field not defined with the Grid API

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

Description This routine writes the field metadata for a grid field not defined by the Grid API

Example

```
status = GDwritefieldmeta(gridID, "ExternField",  
"Ydim,Xdim", DFNT_FLOAT32);
```

FORTRAN integer function gdwrmeta(gridid, fieldname, dimlist, numbertype)

integer*4 *gridid*

character*(*) *fieldname*

character*(*) *dimlist*

integer*4 *numbertype*

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

```
status = gdwrmeta(gridid, "ExternField", "Xdim,Ydim", DFNT_FLOAT32)
```

Write to Tile within Field

GDwritetile

intn GDwritetile(int32 *gridID*, char **fieldname*, int32 *tilecoords*[], VOIDP *data*)

gridID IN: Grid id returned by GDcreate or GDattach

fieldname IN: Fieldname

tilecoords IN: Array of tile coordinates

data IN: Data to be written to tile

Purpose Writes to tile within field.

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

Description This routine writes a single tile of data to a field. If the data to be written to a field can be arranged tile by tile, this routine is more efficient than *GDwritefield*. In all other cases, the later routine should be used. *GDwritetile* does not work on non-tiled fields. Note that the coordinates in terms of tiles, not data elements.

Example In this example. we write one tile to the *Temperature* field (see *GDdeftile* example) at the second column of the first row of tiles. Note that there are $200 * 100 * 4 = 80000$ bytes in *data*:

```
tilecoords[0] = 0;
tilecoords[1] = 1;
status = GDwritetile(gridid, "Temperature", tilecoords,
data);
```

FORTRAN integer function gdwrtle(*gridid*, *fieldname*, *tilecoords*, *data*)

integer*4 *gridid*

character*(*) *fieldname*

integer*4 *tilecoords*(*)

<valid type> *data*(*)

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

```
tilecoords(1) = 1;
tilecoords(2) = 0;
status = gdwrtle(gridid, "Temperature", tilecoords, data)
```

Note that *tilecoords* for *FORTRAN* are reversed from the C language example but the values are still 0-based.

2.1.4 HDF-EOS Utility Routines

This section contains an alphabetical listing of the HDF-EOS utility routines.

Convert Among Angular Units

EHconvAng

float64 EHconvAng(float64 *inAngle*, intn *code*)

inAngle IN: Input angle

code IN: Conversion code

Purpose Convert among various angular units.

Return value Returns angle in desired units.

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: DDDMMMSS.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

intn EHgetversion(int32 *fid*, char **version*)

fid IN: File id returned by *SWopen*, *GDopen*, or *PTopen*.

version OUT: HDF-EOS version string

Purpose Get HDF-EOS version string.

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

Description This routine returns the HDF-EOS version string of an HDF-EOS file. This designates the version of HDF-EOS that was used to create the file. This string is of the form: "HDFEOS_V*maj.min*" where *maj* is the major version and *min* is the minor version.

Example To get the HDF-EOS version (assumed to be 2.0) used to create the HDF-EOS file: "SwathFile.hdf":

```
char version[16];  
fid = SWopen("SwathFile.hdf", DFACC_READ);  
status = EHgetversion(fid, version);  
"version" will contain the string: "HDFEOS_V2.0".
```

FORTRAN integer function ehgetver(*fid*,*version*)

integer*4 *fid*

character*(*) *version*

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

```
character*16 version  
fid = swopen("SwathFile.hdf",1)  
status = ehgetver(fid, version)
```

Get HDF File ids

EHidinfo

intn EHidinfo(int32 *fid*, int32 **HDFfid*, int32 **sdInterfaceID*)

fid IN: File id returned by *SWopen*, *GDopen*, or *PTopen*.

HDFfid OUT: HDF file ID (returned by *Hopen*)

sdInterfaceID OUT: SD interface ID (returned by *SDstart*)

Purpose Get HDF file IDs.

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

Description This routine returns the HDF file ids corresponding to the HDF-EOS file id returned by *SWopen*, *GDopen*, or *PTopen*. These ids can then be used to create or access native HDF structure such as SDS arrays, Vdatas, or HDF attributes within an HDF-EOS file.

Example To create a vdata within an existing HDF-EOS file:

```
char version[16];  
  
fid = SWopen("SwathFile.hdf", DFACC_RDWR);  
status = EHgetid(fid, &HDFfid, &sdInterfaceID);  
vdata_id = VSattach(HDFfid, -1, "w");  
  
[Define vdata fields]  
  
VSdetach(vdata_id);  
  
SWclose(fid);
```

Note that the file is opened and closed using the HDF-EOS open and close routines.

To access the SDS id of an HDF-EOS (unmerged) grid field:

```
fid = SWopen("GridFile.hdf", DFACC_RDWR);  
status = EHgetid(fid, &HDFfid, &sdInterfaceID);  
  
idx = SDnametoindex(sdInterfaceID, "GridField");  
sdsID = SDselect(sdInterfaceID, idx);
```

The user can now apply the HDF SD interface directly to the field.

```
FORTRAN  integer function ehidinfo(fid,hdffid,sdid)  
integer*4  fid  
integer*4  hdffid  
integer*4  sdid
```

To retrieve the HDF file id and SD interface id:

```
fid = swopen("SwathFile.hdf",1)  
status = ehidinfo(fid, hdffid, sdid)
```


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	maintaince 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	relation 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-Processor
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