

175-EMD-002, Revision 05

# HDF-EOS Interface Based on HDF5, Volume 2: Function Reference Guide

Technical Paper

February 2008

Prepared Under Contract NAS5-03098

**RESPONSIBLE ENGINEER**

---

Abe Taaheri Date  
EOSDIS Maintenance and Development Project

**SUBMITTED BY**

---

Art Cohen, Custom Code Maintenance Date  
EOSDIS Maintenance and Development Project

Raytheon Company  
Riverdale, Maryland

This page intentionally left blank.

# Preface

---

This document is a Users Guide for HDF-EOS (Hierarchical Data Format - Earth Observing System) library tools. The version described in this document is HDF-EOS Version 5.1.11. The software is based on HDF5, a new version of HDF provided by NCSA. HDF5 is a complete rewrite of the earlier HDF4 version, containing a different data model and user interface. HDF-EOS V5.1.11 incorporates HDF5, and keeps the familiar HDF4-based interface. There are a few exceptions and these exceptions are described in this document. Note that the major functional difference is that Version 5.1.11 of the HDF-EOS library is a thread-safe.

HDF is the scientific data format standard selected by NASA as the baseline standard for EOS. This Users Guide accompanies Version 5.1.11 software, which is available to the user community on the EDHS1 server. This library is aimed at EOS data producers and consumers, who will develop their data into increasingly higher order products. These products range from calibrated Level 1 to Level 4 model data. The primary use of the HDF-EOS library will be to create structures for associating geolocation data with their associated science data. This association is specified by producers through use of the supplied library. Most EOS data products which have been identified, fall into categories of Point, Grid, Swath or Zonal Average structures, the latter two of which are implemented in the current version of the library. Services based on geolocation information will be built on HDF-EOS structures. Producers of products not covered by these structures, e.g. non-geolocated data, can use the standard HDF libraries.

In the ECS (EOS Core System) production system, the HDF-EOS library will be used in conjunction with SDP (Science Data Processing) Toolkit software. The primary tools used in conjunction with HDF-EOS library will be those for metadata handling, process control and status message handling. Metadata tools will be used to write ECS inventory and granule specific metadata into HDF-EOS files, while the process control tools will be used to access physical file handles used by the HDF tools. (SDP Toolkit Users Guide for the ECS Project, February 2008, 333-EMD-001 Revision 05).

HDF-EOS5 is an extension of NCSA (National Center for Supercomputing Applications) HDF5 and uses HDF5 library calls as an underlying basis. Version 5-1.6.7 of HDF5 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, Swath and Zonal Average structures. This document includes overviews of the interfaces, and code examples. HE5View and HDFView, the HDF-EOS viewing tools, have been revised to accommodate the current version of the library.

Note that HDF-EOS V2.X, a separate library based on HDF4, is also available. Both versions of HDF-EOS will be supported by ECS.

Technical Points of Contact within EOS are:

Abe Taaheri, Abe\_Taaheri@raytheon.com

An email address has been provided for user help:

Landover\_PGSTLKIT@raytheon.com

Any questions should be addressed to:

Data Management Office

The EMD Project Office

Raytheon Company

5700 Rivertech Court

Riverdale, MD 20737

# Abstract

---

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

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

This page intentionally left blank

# Contents

---

## Preface

## Abstract

## 1. Introduction

1.1	Purpose.....	1-1
1.2	Organization.....	1-1
1.3	Point Data .....	1-1
1.3.1	The Point Data Interface .....	1-1
1.3.2	List of Point API Routines.....	1-2
1.4	Swath Data.....	1-3
1.4.1	The Swath Data Interface .....	1-3
1.4.2	List of Swath API Routines .....	1-3
1.5	Grid Data.....	1-6
1.5.1	The Grid Data Interface .....	1-6
1.5.2	List of Grid API Routines.....	1-7
1.6	GCTP Usage .....	1-9
1.6.1	GCTP Projection Codes.....	1-9
1.6.2	UTM Zone Codes .....	1-10
1.6.3	GCTP Spheroid Codes.....	1-11
1.6.4	GCTP Projection Parameters .....	1-12
1.7	Zonal Average Data .....	1-18
1.7.1	The Zonal Average Data Interface.....	1-18
1.7.2	List of Zonal Average API Routines .....	1-18

## 2. Function Reference

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

2.1.2	Swath Interface Functions .....	2-45
2.1.3	Grid Interface Functions .....	2-166
2.1.4	HDF-EOS Utility Routines.....	2-253
2.1.5	Zonal Average Interface Functions.....	2-264

## **List of Tables**

1-1.	Summary of the Point Interface .....	1-2
1-2.	Summary of the Swath Interface.....	1-4
1-3.	Summary of the Grid Interface .....	1-7
1-4.	Projection Transformation Package Projection Parameters.....	1-12
1-5.	Summary of the Zonal Average Interface.....	1-19

## **Appendix A. Numbertype Codes**

### **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 Point (PT) interface consists of routines for storing, retrieving, and manipulating data in point data sets.

### 1.3.1 The Point Data Interface

All C routine names in the Point data interface have the prefix “HE5\_PT” and the equivalent FORTRAN routine names are prefixed by “he5\_pt.” The Point routines are classified into the following categories:

- **Access routines** initialize and terminate access to the Point 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 Point API Routines

The Point function calls are listed in Table 1-1 and are described in detail in the Software Reference Guide that accompanies this document. The page number column in the following table refers to the Software Reference Guide.

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

Category	Routine Name		Description	Pg. Nos.
	C	FORTRAN		
Access	HE5_PTopen	he5_ptopen	Creates a new file or opens an existing one	2-32
	HE5_PTcreate	he5_ptcreate	Creates a new point data set and returns a handle	2-06
	HE5_PTattach	he5_ptattach	Attaches to an existing point data set	2-02
	HE5_PTdetach	he5_ptdetach	Releases a point data set and frees memory	2-14
	HE5_PTclose	he5_ptclose	Closes the HDF-EOS file and deactivates the point interface	2-05
Definition	HE5_PTdeflevel	he5_ptdeflevel	Defines a level within the point data set	2-07
	HE5_PTdeflinkage	he5_ptdeflinkage	Defines link field to use between two levels	2-13
Basic I/O	HE5_PTwritelevel	he5_ptwritelevel	Writes (appends) full records to a level	2-42
	HE5_PTreadlevel	he5_ptreadlevel	Reads data from the specified fields and records of a level	2-36
	HE5_PTupdatelevel	he5_ptupdatelevel	Updates the specified fields and records of a level	2-37
	HE5_PTwriteattr	he5_ptwriteattr	Creates or updates an attribute of the point data set	2-38
	HE5_PTwritegrpatr	he5_ptwritegrpatr	Writes/updates group attribute in a point	2-40
	HE5_PTritelocatr	he5_ptritelocatr	Write/updates local attribute in a point	2-43
	HE5_PTreadattr	he5_ptreadattr	Reads existing attribute of point data set	2-33
	HE5_PTreadgrpatr	he5_ptreadgrpatr	Reads group attribute from a point	2-34
	HE5_PTreadlocatr	he5_ptreadlocatr	Reads local attribute from a point	2-35
Inquiry	HE5_PTNlevels	he5_ptnlevels	Returns the number of levels in a point data set	2-30
	HE5_PTNrecs	he5_ptnrecs	Returns the number of records in a level	2-31
	HE5_PTNfields	he5_ptnfields	Returns number of fields defined in a level	2-29
	HE5_PTlevelinfo	he5_ptlevelinfo	Returns information about a given level	2-27
	HE5_PTlevelindx	he5_ptlevelindx	Returns index number for a named level	2-26
	HE5_PTbcklinkinfo	he5_ptbcklinkinfo	Returns link field to previous level	2-04
	HE5_PTfwdlinkinfo	he5_ptfwdlinkinfo	Returns link field to following level	2-15
	HE5_PTgetlevelname	he5_ptgetlevelname	Returns level name given level number	2-16
	HE5_PTgetrecnums	None	Retrieves number of records in one level corresponding to a group of records in a different level	2-17
HE5_PTattrinfo	he5_ptattrinfo	Returns information about point attributes	2-03	

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

Category	Routine Name		Description	Pg. Nos.
	C	FORTRAN		
Inquiry	HE5_PTgrpattrinfo	he5_ptgrpattrinfo	Returns information about point group attributes	2-18
	HE5_PTlocattrinfo	he5_ptlocattrinfo	Returns information about point local attributes	2-28
	HE5_PTinqattr	he5_ptinqattr	Retrieves number and names of point attributes	2-19
	HE5_PTinqgrpattr	he5_ptinqgrpattr	Retrieves number and names of group attributes	2-22
	HE5_PTinqlocattr	he5_ptinqlocattr	Retrieves number and names of local attributes defined	2-23
	HE5_PTinqpoint	he5_ptinqpoint	Retrieves number and names of points in file	2-25
	HE5_PTinqdatatype	he5_ptinqdatatype	Returns data type information about specified level in point	2-20

## 1.4 Swath Data

The Swath (SW) 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 "HE5\_SW" and the equivalent FORTRAN routine names are prefixed by "he5\_sw." The Swath routines are classified into the following categories:

- **Access routines** initialize and terminate access to the Swath 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 Swath API Routines

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

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

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Access	HE5_SWopen	He5_swopen	Opens or creates HDF file in order to create, read, or write a swath	2-117
	HE5_SWcreate	He5_swcreate	Creates a swath within the file	2-53
	HE5_SWattach	He5_swattach	Attaches to an existing swath within file	2-47
	HE5_SWdetach	He5_swdetach	Detaches from swath interface	2-76
	HE5_SWclose	He5_swclose	Closes file	2-51
Definition	HE5_SWdefdim	He5_swdefdim	Defines a new dimension within the swath	2-64
	HE5_SWdefdimmap	He5_swdefmap	Defines the mapping between the geolocation and data dimensions	2-66
	HE5_SWdefidxmap	He5_swdefimap	Defines a non-regular mapping between the geolocation and data dimension	2-70
	HE5_SWdefgeofield	He5_swdefgfld	Defines a new geolocation field within the swath	2-68
	HE5_SWdefdatafield	He5_swdefdfld	Defines a new data field within the swath	2-62
	HE5_SWdefcomp	He5_swdefcomp	Defines a field compression scheme	2-59
	HE5_SWdefchunk	He5_swdefchunk	Define chunking parameters	2-56
	HE5_SWdefcomchunk	He5_swdefcomch	Defines compression with automatic chunking	2-57
	HE5_SWsetalias	He5_swsetalias	Defines alias for data field	2-131
	HE5_SWdropalias	He5_swdrpalias	Removes alias from the list of field aliases	2-78
	HE5_SWfldrename	He5_swfldnm	Changes the field name	2-85
Basic I/O	HE5_SWwritefield	He5_swwrfld	Writes data to a swath field	2-141
	HE5_SWwritegeometa	He5_swwrgmeta	Writes field metadata for an existing swath geolocation field	2-146
	HE5_SWwritedatameta	He5_swwrmeta	Writes field metadata for an existing swath data field	2-140
	HE5_SWreadfield	He5_swrdfld	Reads data from a swath field.	2-122
	HE5_SWwriteattr	He5_swwrattr	Writes/updates attribute in a swath	2-138
	HE5_SWreadattr	He5_swrdatr	Reads attribute from a swath	2-120
	HE5_SWwritegeogrpatr	He5_swwrgeogattr	Writes/updates group Geolocation Fields attribute in a swath	2-144
	HE5_SWwritegrpattr	He5_swwrgrattr	Writes/updates group Data Fields attribute in a swath	2-148
	HE5_SWwritelocattr	He5_swwrlattr	Write/updates local attribute in a swath	2-150
	HE5_SWreadgeogrpatr	He5_swrdgeogattr	Reads attribute in group Geolocation Fields from a swath	2-124
	HE5_SWreadgrpattr	He5_swrgrattr	Reads attribute in Data Fields from a swath	2-125
	HE5_SWreadlocattr	He5_swrldattr	Reads a local attribute from a swath	2-126
	HE5_SWsetfillvalue	He5_swsetfill	Sets fill value for the specified field	2-133
	HE5_SWgetfillvalue	He5_swgetfill	Retrieves fill value for the specified field	2-90
Inquiry	HE5_SWaliasinfo	He5_swaliasinfo	Retrieves information about field aliases	2-46
	HE5_SWgetaliaslist	He5_swgetaliaslist	Retrieves list and number of aliases in a geo or data group	2-88
	HE5_SWinqdims	He5_swinqdims	Retrieves information about dimensions defined in swath	2-101
	HE5_SWinqmaps	He5_swinqmaps	Retrieves information about the geolocation relations defined	2-111
	HE5_SWinqidxmaps	He5_swinqimaps	Retrieves information about the indexed geolocation/data mappings defined	2-108
	HE5_SWinqgeofields	He5_swinqgflds	Retrieves information about the geolocation fields defined	2-102

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

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Inquiry	HE5_SWinqdatafields	he5_swinqdflds	Retrieves information about the data fields defined	2-96
	HE5_SWinqattrs	he5_swinqattrs	Retrieves number and names of attributes defined	2-95
	HE5_SWinqdatatype	he5_swidtype	Returns data type information about specified fields in swath	2-97
	HE5_SWinqfldalias	he5_swinqfldalias	Returns information about data fields & aliases defined in swath	2-99
	HE5_SWinqgfldalias	he5_swinqgfldalias	Returns information about geolocation fields & aliases defined in swath	2-105
	HE5_SWinqgeogrpatrs	he5_swinqgeogattr	Retrieve information about group Geolocation Fields attributes defined in swath	2-103
	HE5_SWinqgrpattr	he5_swinqgattr	Retrieve information about group Data Fields attributes defined in swath	2-107
	HE5_SWinqlocattr	he5_swinqlattr	Retrieve information about local attributes defined in swath	2-109
	HE5_SWlocattrinfo	he5_swlocattrinfo	Returns information about a local attribute(s)	2-113
	HE5_SWnentries	he5_swnentries	Returns number of entries and descriptive string buffer size for a specified entity	2-116
	HE5_SWdiminfo	he5_swdiminfo	Retrieve size of specified dimension	2-77
	HE5_SWchunkinfo	he5_swchunkinfo	Retrieve chunking information	2-49
	HE5_SWmapinfo	he5_swmapinfo	Retrieve offset and increment of specified geolocation mapping	2-114
	HE5_SWidxmapinfo	he5_swidxmapinfo	Retrieve offset and increment of specified geolocation mapping	2-92
	HE5_SWattrinfo	he5_swattrinfo	Returns information about swath attribute	2-48
	HE5_SWgeogrpatrinfo	he5_swgeogattrinfo	Returns information about group Geolocation Fields attribute	2-86
	HE5_SWgrpattrinfo	he5_swgattrinfo	Returns information about group Data Fields attribute	2-91
	HE5_SWfieldinfo	he5_swfldinfo	Retrieve information about a specific geolocation or data field	2-83
	HE5_SWcompinfo	he5_swcompinfo	Retrieve compression information about a field	2-52
	HE5_SWinqswath	he5_swinqswath	Retrieves number and names of swaths in file	2-112
	HE5_SWregionindex	he5_swregidx	Returns information about the swath region ID	2-127
	HE5_SWupdateidxmap	he5_swupimap	Update map index for a specified region	2-136
	HE5_SWgeomapinfo	he5_swgmapinfo	Retrieve type of dimension mapping for a dimension	2-87
	HE5_SWdefboxregion	he5_swdefboxreg	Define region of interest by latitude/longitude	2-54
	HE5_SWregioninfo	he5_swreginfo	Returns information about defined region	2-129
	HE5_SWextractregion	he5_swextreg	Read a region of interest from a field	2-82

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

Category	Routine Name		Description	Page Nos.
	C	FORTTRAN		
Subset	HE5_SWdeftimeperiod	he5_swdeftimeper	Define a time period of interest	2-71
	HE5_SWperiodinfo	he5_swperinfo	Returns information about a defined time period	2-118
	HE5_SWextractperiod	he5_swextper	Extract a defined time period	2-80
	HE5_SWdefvrtregion	he5_swdefvrtreg	Define a region of interest by vertical field	2-73
	HE5_SWindexinfo	he5_swindexinfo	Returns the indices about a subsetted region	2-93
	HE5_SWdupregion	he5_swdupreg	Duplicate a region or time period	2-79
Profile	HE5_PRdefine	he5_prdefine	Defines profile data structure	2-152
	HE5_PRread	he5_prread	Reads profile data	2-158
	HE5_PRwrite	he5_prwrite	Writes profile data	2-162
	HE5_PRinquire	he5_prinquire	Retrieves information about profiles	2-157
	HE5_PRinfo	he5_prinfo	Return information about profile	2-154
	HE5_PRwritegrpattr	he5_prwrgattr	Writes/updates group Profile Fields attribute in a swath	2-164
	HE5_PRreadgrpattr	he5_prrdgattr	Reads attribute in group ProfileFields from a swath	2-160
	HE5_PRinqgrpattr	he5_prinqgattr	Retrieve information about group Profile Fields attributes defined in swath	2-156
	HE5_PRgrpattrinfo	he5_prgattrinfo	Returns information about a group Profile Fields attribute	2-153
	HE5_PRreclaimspace	Not available	Reclaims memory used by data buffer in HE5_PRread()call	2-161
External Files	HE5_SWmountexternal	Not available	Mount external data file	2-115
	HE5_SWreadexternal	Not available	Read external data set	2-121
	HE5_SWunmount	Not available	Dismount external data file	2-135
External Data Sets	HE5_SWsetextdata	he5_swsetxdat	Set external data set	2-132
	HE5_SWgetextdata	he5_swgetxdat	Get external data set	2-89

## 1.5 Grid Data

The Grid (GD) 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 "HE5\_GD" and the equivalent FORTRAN routine names are prefixed by "he5\_gd." The GD routines are classified into the following categories:

- **Access routines** initialize and terminate access to the Grid 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 Grid 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 Grid Interface (1 of 3)**

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Access	HE5_GDopen	he5_gdopen	Creates a new file or opens an existing one	2-226
	HE5_GDcreate	he5_gdcreate	Creates a new grid in the file	2-174
	HE5_GDattach	he5_gdattach	Attaches to a grid	2-168
	HE5_GDdetach	he5_gddetach	Detaches from grid interface	2-196
	HE5_GDclose	he5_gdclose	Closes file	2-172
Definition	HE5_GDdeforigin	he5_gddeforigin	Defines origin of grid pixel	2-185
	HE5_GDdefdim	he5_gddefdim	Defines dimensions for a grid	2-182
	HE5_GDdefproj	he5_gddefproj	Defines projection of grid	2-187
	HE5_GDdefpixreg	he5_gddefpixreg	Defines pixel registration within grid cell	2-186
	HE5_GDdeffield	he5_gddeffld	Defines data fields to be stored in a grid	2-183
	HE5_GDdefcomp	he5_gddefcomp	Defines a field compression scheme	2-178
	HE5_GDblkSOMoffset	None	This is a special function for SOM MISR data. Write block SOM offset values.	2-170
	HE5_GDdefcomtile	he5_gddefcomtile	Defines compression with automatic tiling	2-181
	HE5_GDsetalias	he5_gdsetalias	Defines alias for data field	2-238
HE5_GDdropalias	he5_gddropalias	Removes alias from a list of field alias	2-198	
Basic I/O	HE5_GDwritefieldmeta	he5_gdwrmeta	Writes metadata for field already existing in file	2-248
	HE5_GDwritefield	he5_gdwrfld	Writes data to a grid field.	2-245
	HE5_GDreadfield	he5_gdrdfld	Reads data from a grid field	2-232
	HE5_GDwriteattr	he5_gdwrattr	Writes/updates attribute in a grid.	2-243
	HE5_GDwritelocattr	he5_gdwrlattr	Writes/updates local attribute in a grid	2-251
	HE5_GDwritegrpattr	he5_gdwrgattr	Writes/updates group attribute in a grid	2-249
	HE5_GDreadattr	he5_gdrdatr	Reads attribute from a grid	2-231
	HE5_GDreadgrpattr	he5_gdrdgattr	Reads group attribute from a grid	2-234
	HE5_GDreadlocattr	he5_gdrdlattr	Reads local attribute from a grid	2-235
	HE5_GDsetfillvalue	he5_gdsetfill	sets fill value for the specified field	2-240
HE5_GDgetfillvalue	he5_gdgetfill	Retrieves fill value for the specified field	2-205	
	HE5_GDgetaliaslist	He5_gdgetaliaslist	Retrieves list and number of aliases in a data group	2-203
	HE5_GDinqdims	he5_gdinqdims	Retrieves information about dimensions defined in grid	2-215

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

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Inquiry	HE5_GDinqfields	he5_gdinqflds	Retrieves information about the data fields defined in grid	2-216
	HE5_GDinqattrs	he5_gdinqattrs	Retrieves number and names of attributes defined	2-212
	HE5_GDinqdatatype	he5_gdinqdatatype	Returns data type information about specified fields in grid	2-213
	HE5_GDinqgrpattrs	he5_gdinqgrpattrs	Retrieve information about group attributes defined in grid	2-219
	HE5_GDinqlocattrs	he5_gdinqlattrs	Retrieve information about local attributes defined for a field	2-220
	HE5_GDinqfldalias	he5_gdinqfldalias	Returns information about data fields & aliases defined in grid	2-217
	HE5_GDnentries	he5_gdnentries	Returns number of entries and descriptive string buffer size for a specified entity	2-225
	HE5_GDaliasinfo	he5_gdaliasinfo	Retrieves information about aliases	2-167
	HE5_GDgridinfo	he5_gdgridinfo	Returns dimensions of grid and X-Y coordinates of corners	2-210
	HE5_GDprojinfo	he5_gdprojinfo	Returns all GCTP projection information	2-230
	HE5_GDdiminfo	he5_gddiminfo	Retrieves size of specified dimension.	2-197
	HE5_GDcompinfo	he5_gdcompinfo	Retrieve compression information about a field	2-173
	HE5_GDfieldinfo	he5_gdfieldinfo	Retrieves information about a specific field in the grid	2-201
	HE5_GDinqgrid	he5_gdinqgrid	Retrieves number and names of grids in file	2-218
	HE5_GDattrinfo	he5_gdattrinfo	Returns information about grid attributes	2-169
	HE5_GDgrpattrinfo	he5_gdgrpattrinfo	Returns information about a grid group attribute	2-211
	HE5_GDlocattrinfo	he5_gdlocattrinfo	Returns information about a Data Field's local attribute(s)	2-224
	HE5_GDorigininfo	he5_gdorigininfo	Return information about grid pixel origin	2-228
	HE5_GDpixreginfo	he5_gdpxreginfo	Return pixel registration information for given grid	2-229
	HE5_GDdefboxregion	he5_gddefboxreg	Define region of interest by latitude/longitude	2-177
	HE5_GDregioninfo	he5_gdregioninfo	Returns information about a defined region	2-236
	HE5_GDextractregion	he5_gdextractreg	Read a region of interest from a field	2-200



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

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Subset	HE5_GDdeftimeperiod	he5_gddefmeper	Define a time period of interest	2-192
	HE5_GDdefvrtregion	he5_gddefvrtreg	Define a region of interest by vertical field	2-194
	HE5_GDgetpixels	he5_gdgetpix	get row/columns for lon/lat pairs	2-206
	HE5_GDgetpixvalues	he5_gdgetpixval	get field values for specified pixels	2-208
	HE5_GDinterpolate	he5_gdinterpolate	Perform bilinear interpolation on a grid field	2-222
	HE5_GDdupregion	he5_gddupreg	Duplicate a region or time period	2-199
Tiling	HE5_GDdeftile	he5_gddeftle	Define a tiling scheme	2-189
	HE5_GDtileinfo	he5_gdtileinfo	Retrieve tiling information	2-241
Utility	HE5_GDij2ll	he5_gdij2ll	convert (i,j) coordinates to (lon,lat) for a grid	2-325
	HE5_GDl2ij	he5_gdl2ij	convert (lon,lat) coordinates to (i,j) for a grid	2-328
	HE5_GDRs2ll	he5_gdrs2ll	Convert (r,s) coordinates to (lon,lat) for EASE grid	2-331
External Data Sets	HE5_GDsetextdata	he5_gdsetxdat	Set external data set	2-239
	HE5_GDgetextdata	he5_gdgetxdat	Get external data set	2-204

## 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 describes 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 4 below:

HE5_GCTP_ALBERS		Albers Conical Equal-Area Projection
HE5_GCTP_MERCAT		Mercator Projection
HE5_GCTP_SPCS		State Plane Coordinate System
HE5_GCTP_GEO	(0)	Geographic
HE5_GCTP_UTM	(1)	Universal Transverse Mercator
HE5_GCTP_LAMCC	(4)	Lambert Conformal Conic
HE5_GCTP_PS	(6)	Polar Stereographic
HE5_GCTP_POLYC	(7)	Polyconic
HE5_GCTP_TM	(9)	Transverse Mercator
HE5_GCTP_LAMAZ	(11)	Lambert Azimuthal Equal Area
HE5_GCTP_HOM	(20)	Hotine Oblique Mercator
HE5_GCTP_SOM	(22)	Space Oblique Mercator
HE5_GCTP_GOOD	(24)	Interrupted Goode Homolosine
HE5_GCTP_ISINUS	(99/31)	Integerized Sinusoidal Projection*

GCTP_CEA	(97)	Cylindrical Equal-Area (for EASE grid with Corners in meters)**
GCTP_BCEA	(98)	Cylindrical Equal-Area (for EASE grid with grid corners in packed degrees, DMS)**

\* The Integerized 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.

\*\* The Cylindrical Equal-Area Projection was not part of the original GCTP package. It has been added by ECS. See Notes for section 6.5.4.

In the new GCTP package the Integerized Sinusoidal Projection is included as the 31<sup>st</sup> projection. The Code 31 was added to HDF-EOS for users who wish to use 31 instead of 99 for Integerized Sinusoidal Projection.

Note that other projections supported by GCTP will be adapted for HDF-EOS Version 5 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, February 2008, (333-EMD-001 Revision 05).

## 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)
Australian National	(14)
Krassovsky	(15)
Hough	(16)
Mercury 1960	(17)
Modified Mercury 1968	(18)
Sphere of Radius 6370997m	(19)
Sphere of Radius 6371228m	(20)
Sphere of Radius 6371007.181m	(21)

## 1.6.4 GCTP Projection Parameters

**Table 1-4. 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							
97 CEA Utilizd by EASE Grid (see Notes)	Smajor	SMinor			CentMer	TrueScale	FE	FN
98 BCEA Utilized by EASE Grid (see Notes)	Smajor	SMinor			CentMer	TrueScale	FE	FN
31 & 99 Integerized Sinusoidal	Sphere				CentMer		FE	FN

**Table 1-4. 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	HDF-EOS Variable	Zero
22 Space Oblique Merc B				HDF-EOS Variable	One
24 Interrupted Goode					
97 CEA Utilized by EASE Grid (see Notes)					
98 BCEA Utilized by EASE Grid (see Notes)					
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. It is recommended that explicit value, rather than zero, is used for Smajor.

Sminor	Eccentricity squared of the ellipsoid if less than one, if zero, a spherical form is assumed, or if greater than one, the semi-minor axis of ellipsoid. It should be noted that a negative sphere code should be used in order to have user specified Smajor and Sminor be accepted by GCTP, otherwise default ellipsoid Smajor and Sminor will be used.
Sphere	Radius of reference sphere. If zero, 6370997 meters is used. It is recommended that explicit value, rather than zero, is used for Sphere.
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:**

- HDF-EOS variable is used by the library function HE5\_GDbklsomoffset.
- 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 (DDMMSS.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

The following notes apply for **BCEA and CEA projections**, and **EASE grid**:

Behrmann Cylindrical Equal-Area (BECA) projection was used for 25 km global EASE grid. For this projection the Earth radius is set to 6371228.0m and latitude of true scale is 30 degrees. For 25 km global EASE grid the following apply:

Grid Dimensions:

Width 1383

Height 586

Map Origin:

Column (r0) 691.0

Row (s0) 292.5

Latitude 0.0

Longitude 0.0

Grid Extent:

Minimum Latitude 86.72S

Maximum Latitude 86.72N

Minimum Longitude 180.00W

Maximum Longitude 180.00E

Actual grid cell size 25.067525km

Grid coordinates (r,s) start in the upper left corner at cell (0.0), with r increasing to the right and s increasing downward.

Although the projection code and name (tag) kept the same, BCEA projection was generalized to accept Latitude of True Scales other than 30 degrees, Central Meridian other than zero, and ellipsoid earth model besides the spherical one with user supplied radius. This generalization along with the removal of hard coded grid parameters will allow users not only subsetting, but also creating other grids besides the 25km global EASE grid and having freedom to use different appropriate projection parameters. One can create the above mentioned 25km global EASE grid using:

Grid Dimensions:

Width 1383

Height 586

Grid Extent:

UpLeft Latitude 86.72

LowRight Latitude -86.72

UpLeft Longitude -180.00



LowRight Longitude 180.00

Projection Parameters:

- 1)  $6371.2280/25.067525 = 254.16263$
- 2)  $6371.2280/25.067525 = 254.16263$
- 5) 0.0
- 6) 30000000.0
- 7) 691.0
- 8) -292.5

Also one may create **12.5 km global EASE grid** using:

Grid Dimensions:

Width 2766

Height 1171

Grid Extent:

UpLeft Latitude 85.95

LowRight Latitude -85.95

UpLeft :Longitude -179.93

LowRight Longitude 180.07

Projection Parameters:

- 1)  $6371.2280 / (25.067525/2) = 508.325253$
- 2)  $6371.2280 / (25.067525/2) = 508.325253$
- 5) 0.0
- 6) 30000000.0
- 7) 1382.0
- 8) -585.0

Any other grids (normalized pixel or not) with generalized BCEA projection can be created using appropriate grid corners, dimension sizes, and projection parameters. Please note that like other projections Semi-major and Semi-minor axes will default to Clarke 1866 values (in meters) if they are set to zero.

A new projection CEA (97) was added to GCTP. This projection is the same as the generalized BCEA, except that the EASE grid produced will have its corners in meters rather than packed degrees, which is the case with EASE grid produced by BCEA.

## 1.7 Zonal Average Data

The Zonal Average (ZA) interface consists of routines for storing, retrieving, and manipulating data in zonal average data sets. The zonal average data is not associated with specific geolocation information. See the Users' Guide, Volume 1 that accompanies this document for more information.

### 1.7.1 The Zonal Average Data Interface

All C routine names in the zonal average data interface have the prefix "HE5\_ZA" and the equivalent FORTRAN routine names are prefixed by "he5\_za". The zonal average routines are classified into the following categories:

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

### 1.7.2 List of Zonal Average API Routines

The Zonal Average 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 Zonal Average Interface (1 of 2)**

Category	Routine Name		Description	Page
	C	FORTRAN		Nos.
Access	HE5_Zaopen	he5_zaoopen	Opens or creates HDF file in order to create, read, or write a zonal average	2-305
	HE5_Zacreate	he5_zacreate	Creates a zonal average within the file	2-272
	HE5_Zaattach	he5_zaattach	Attaches to an existing zonal average within the file	2-266
	HE5_Zadetach	he5_zadetach	Detaches from zonal average interface	2-281
	HE5_Zaclose	he5_zaclose	Closes file	2-270
Definition	HE5_Zadefdim	he5_zadefdim	Defines a new dimension within the zonal average	2-278
	HE5_Zadefine	he5_zadefine	Defines a new data field within the zonal average	2-280
	HE5_Zadefcomp	he5_zadefcomp	Defines a field compression scheme	2-276
	HE5_Zadefchunk	he5_zadefchunk	Defines chunking parameters	2-273
	HE5_Zadefcomchunk	he5_zadefcomch	Defines compression with automatic chunking	2-274
	HE5_Zasetalias	he5_zasetalias	Defines alias for data field	2-312
	HE5_Zadropalias	he5_zadropalias	Removes alias from the list of field aliases	2-283
HE5_Zafldrename	he5_zafldnm	Changes the field name	2-284	
Basic I/O	HE5_Zawrite	he5_zawrite	Writes data to a zonal average field	2-316
	HE5_Zaread	he5_zaread	Reads data from a zonal average field.	2-306
	HE5_Zawriteattr	he5_zawratr	Writes/updates attribute in a zonal average	2-318
	HE5_Zareadattr	he5_zardattr	Reads attribute from a zonal average	2-308
	HE5_Zawritegrpattr	he5_zawrgattr	Writes/updates group attribute in a zonal average	2-321
	HE5_Zawritelocatr	he5_zawrlattr	Writes/updates group attribute in a zonal average	2-323
	HE5_Zawritedatameta	he5_zawrdmeta	Writes field metadata for an existing zonal average data field	2-320
	HE5_Zareadgrpattr	he5_zardgattr	Reads attribute from a zonal average	2-310
	HE5_Zareadlocattr	he5_zardlattr	Reads attribute from a zonal average	2-311
	HE5_Zasetfillvalue	he5_zasetfill	Sets fill value for the specified field	2-314
	HE5_Zagetfillvalue	he5_zagetfill	Retrieves fill value for the specified field	2-287
	Inquiry	HE5_ZAgetaliaslist	He5_zagetaliaslist	Retrieves list and number of aliases in a data group
HE5_Zaaliasinfo		he5_zaliasinfo	Retrieves information about field aliases	2-265
HE5_Zainqdims		he5_zainqdims	Retrieves information about dimensions defined in zonal average	2-294
HE5_Zainquire		he5_zainquire	Retrieves information about the data fields defined	2-300
HE5_Zainqattrs		he5_zainqattrs	Retrieves number and names of attributes defined	2-291
HE5_Zainqdatatype		he5_zaidtype	Returns data type information about specified fields in a zonal average	2-292
HE5_Zainqfldalias		he5_zainqfldalias	Returns information about data fields & aliases defined in a zonal average	2-295
HE5_Zachunkinfo		he5_zachunkinfo	Retrieves chunking information	2-268
HE5_Zainqgrpattrs		he5_zainqgattrs	Retrieves information about group attributes defined in zonal average	2-297
HE5_Zainqlocattrs		he5_zainqlattrs	Retrieves information about local attributes defined in zonal average	2-298
HE5_Zalocatrinfo		he5_zalocatrinfo	Returns information about a data field's local attribute(s)	2-302

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

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Inquiry	HE5_Zanentries	he5_zanentries	Returns number of entries and descriptive string buffer size for a specified entity	2-304
	HE5_Zadiminfo	he5_zadiminfo	Retrieves size of specified dimension	2-282
	HE5_Zaattrinfo	he5_zaattrinfo	Returns information about zonal average attributes	2-267
	HE5_Zagrpattrinfo	he5_zagattrinfo	Returns information about a zonal average group attribute	2-288
	HE5_Zainfo	he5_zainfo	Retrieves information about a specific data field	2-289
	HE5_Zacompinfo	he5_zacompinfo	Retrieves compression information about a field	2-271
	HE5_Zainqza	he5_zainqza	Retrieves number and names of zas in file	2-301
External Files	HE5_Zamountexternal	Not available	Mounts external data file	2-303
	HE5_Zareadexternal	Not available	Reads external data set	2-309
	HE5_Zaunmount	Not available	Dismounts external data file	2-315
External Data Sets	HE5_Zasetextdata	he5_zasetxdat	Sets external data set	2-313
	HE5_Zagetextdata	he5_zagetxdat	Gets external data set	2-286

## 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

---

## HE5\_PTattach

hid\_t HE5\_PTattach(hid\_t *fid*, const char \**pointname*)

*fid* IN: Point file ID returned by HE5\_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 or 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-EOS file, *Point.he5*, referred to by the handle, *fid*:

```
pointID = HE5_PTattach(fid, "ExamplePoint");
```

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

FORTRAN integer function he5\_ptattach(*fid*,*pointname*)

integer *fid*

character\*(\*) *pointname*

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

```
pointid = he5_ptattach(fid, "ExamplePoint")
```

## Return Information About a Point Attribute

---

### HE5\_PTattrinfo

herr\_t HE5\_PTattrinfo(hid\_t *pointID*, const char \**attrname*, hid\_t \* *numbertype*, hsize\_t \**count*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*attrname* IN: Attribute name

*numbertype* OUT: Number type of attribute. See Appendix A for interpretation of number types.

*count* OUT: Number of elements in attribute

Purpose Returns information about an object attribute in a specific point. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Returns SUCCEED (0) if successful or 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 = HE5_PTattrinfo(pointID, "ScalarFloat",&nt,&count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

FORTRAN integer function he5\_ptattrinfo(*pointid*,*attrname*,*ntype*,*count*)

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

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

```
pointid = he5_ptattrinfo(pointid, "ScalarFloat",ntype,count)
```

# Return Linkage Field to Previous Level

---

## HE5\_PTbcklinkinfo

herr\_t HE5\_PTbcklinkinfo(hid\_t *pointID*, int *level*, char \**linkfield*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_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 or 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 *HE5\_PTdeflevel* routine.)

```
status = HE5_PTbcklinkinfo(pointID2, 1, linkfield);
```

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

FORTRAN integer function he5\_ptbcklinkinfo(*pointid*,*level*,*linkfield*)

integer *pointid*,*status*

character \*(\*) *linkfield*

integer *level*

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

```
level = 1
```

```
status = he5_ptbcklinkinfo(pointid, level, linkfield)
```



# Close an HDF-EOS File

---

## HE5\_PTclose

herr\_t HE5\_PTclose(hid\_t *fid*)

*fid* IN: Point file ID returned by HE5\_PTopen

Purpose Closes a file opened by HE5\_Ptopen() .

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

Description This routine closes the HDF-EOS Point file.

Example

```
status = HE5_PTclose(fid);
```

FORTRAN integer function he5\_ptclose(*fid*)

integer *fid*

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

```
status = he5_ptclose(fid)
```

# Create a New Point Structure

---

## HE5\_PTcreate

hid\_t HE5\_PTcreate(hid\_t *fid*, const char \**pointname*)

*fid* IN: Point file ID returned by HE5\_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 or FAIL (-1) otherwise.

Description The point is created as a Compound dataset within the HDF-EOS file with the name *pointname* .

Example In this example, we create a new point structure, *ExamplePoint*, in the previously created file, *Point.he5*.

```
pointID = HE5_PTcreate(fid, "ExamplePoint");
```

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

FORTTRAN integer function he5\_ptcreate(*fid*,*pointname*)

integer *pointid*, *fid*

character \*(\*) *pointame*

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

```
pointid = he5_ptcreate(fid, "ExamplePoint")
```

# Define a New Level Within a Point

---

## HE5\_PTdeflevel

herr\_t HE5\_PTdeflevel(hid\_t *pointID*, const char \**levelname*, HE5\_CmpDTSinfo \**levelinfo*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach  
*levelname* IN: Name of level to be defined  
*levelinfo* IN: C-data structure containing all necessary information about level to be defined

**Note: Merging is not supported in this release of the library. There are three illegal characters for field names: “/”, “;”, “,”**

**Purpose** Defines a new level within the point.  
**Return value** Returns SUCCEED (0) if successful or 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.

```
typedef struct {
    int    id;
    int    time;
    float  lon;
    float  lat;
    float  temp[4];
    char   mode[4];
} InputData1;
```

```

HE5_CmpDTSinfo  dtsinfo;

dtsinfo.nfields = 6;

dtsinfo.rank[0] = 1;

dtsinfo.rank[1] = 1;

dtsinfo.rank[2] = 1;

dtsinfo.rank[3] = 1;

dtsinfo.rank[4] = 1;

dtsinfo.rank[5] = 1;

for (i = 0; i < 6; i++)

    dtsinfo.fieldname[i] = (char
*)malloc(64,sizeof(char));

strcpy(dtsinfo.fieldname[0], "ID");
strcpy(dtsinfo.fieldname[1], "Time");
strcpy(dtsinfo.fieldname[2], "Longitude");
strcpy(dtsinfo.fieldname[3], "Latitude");
strcpy(dtsinfo.fieldname[4], "Temperature");
strcpy(dtsinfo.fieldname[5], "Mode");

dtsinfo.offset[0] = HOFFSET(InputData1, id);
dtsinfo.offset[1] = HOFFSET(InputData1, time);
dtsinfo.offset[2] = HOFFSET(InputData1, lon);
dtsinfo.offset[3] = HOFFSET(InputData1, lat);
dtsinfo.offset[4] = HOFFSET(InputData1, temp);
dtsinfo.offset[5] = HOFFSET(InputData1, mode);

dtsinfo.dtype[0] = H5T_NATIVE_INT;
dtsinfo.dtype[1] = H5T_NATIVE_INT;
dtsinfo.dtype[2] = H5T_NATIVE_FLOAT;
dtsinfo.dtype[3] = H5T_NATIVE_FLOAT;
dtsinfo.dtype[4] = H5T_NATIVE_FLOAT;

```

```

dtsinfo.dtype[5] = H5T_NATIVE_CHAR;

dtsinfo.dims[0][0] = 1;
dtsinfo.dims[1][0] = 1;
dtsinfo.dims[2][0] = 1;
dtsinfo.dims[3][0] = 1;
dtsinfo.dims[4][0] = 4;
dtsinfo.dims[5][0] = 4;

dtsinfo.datasize = sizeof(InputData1);
status = HE5_PTdeflevel(pointID1, "Sensor", &dtsinfo);
for (i = 0; i < 6; i++)
    free(dtsinfo.fieldname[i]);

```

### 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 *HE5\_PTdeflinkage* routine described later.) The entries within this 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

```

HE5_CmpDTSinfo  lev0_info;

typedef struct {
    char    label[8];
    double  lon;
    double  lat;
    float   deploydate;
    char    id;

```

```

} Lev0_Data;

lev0_info.nfields = 5;
lev0_info.rank[0] = 1;
lev0_info.rank[1] = 1;
lev0_info.rank[2] = 1;
lev0_info.rank[3] = 1;
lev0_info.rank[4] = 1;
for (i = 0; i < 5; i++)
    lev0_info.fieldname[i] = (char
*)calloc(64,sizeof(char));

strcpy(lev0_info.fieldname[0], "Label");
strcpy(lev0_info.fieldname[1], "Longitude");
strcpy(lev0_info.fieldname[2], "Latitude");
strcpy(lev0_info.fieldname[3], "DeployDate");
strcpy(lev0_info.fieldname[4], "ID");

lev0_info.offset[0] = HOFFSET(Lev0_Data, label);
lev0_info.offset[1] = HOFFSET(Lev0_Data, lon);
lev0_info.offset[2] = HOFFSET(Lev0_Data, lat);
lev0_info.offset[3] = HOFFSET(Lev0_Data, deploydate);
lev0_info.offset[4] = HOFFSET(Lev0_Data, id);

lev0_info.dtype[0] = H5T_NATIVE_CHAR;
lev0_info.dtype[1] = H5T_NATIVE_DOUBLE;
lev0_info.dtype[2] = H5T_NATIVE_DOUBLE;
lev0_info.dtype[3] = H5T_NATIVE_FLOAT;
lev0_info.dtype[4] = H5T_NATIVE_CHAR;

lev0_info.dims[0][0] = 8;
lev0_info.dims[1][0] = 1;

```

```

lev0_info.dims[2][0] = 1;
lev0_info.dims[3][0] = 1;
lev0_info.dims[4][0] = 1;

lev0_info.datasize = sizeof(Lev0_Data);

status = HE5_PTdeflevel(pointID2, "Desc-Loc", &lev0_info);
for (i = 0; i < 5; i++)
    free (lev0_info.fieldname[i]);

```

### Level 1

```

HE5_CmpDTSinfo  lev1_info;
typedef struct {
    double  time;
    float   rain;
    float   temp;
    char    id;
} Lev1_Data;
lev1_info.nfields = 4;
lev1_info.rank[0] = 1;
lev1_info.rank[1] = 1;
lev1_info.rank[2] = 1;
lev1_info.rank[3] = 1;
for (i = 0; i < 4; i++)
    lev1_info.fieldname = (char *)calloc(64,sizeof(char));

strcpy(lev1_info.fieldname[0], "Time");
strcpy(lev1_info.fieldname[1], "Rainfall");
strcpy(lev1_info.fieldname[2], "Temperature");
strcpy((lev1_info.fieldname[3], "ID");

lev1_info.offset[0] = HOFFSET(Lev1_Data, time);

```

```

levl_info.offset[1] = HOFFSET(Levl_Data, rain);
levl_info.offset[2] = HOFFSET(Levl_Data, temp);
levl_info.offset[3] = HOFFSET(Levl_Data, id);

levl_info.dtype[0] = H5T_NATIVE_DOUBLE;
levl_info.dtype[1] = H5T_NATIVE_FLOAT;
levl_info.dtype[2] = H5T_NATIVE_FLOAT;
levl_info.dtype[3] = H5T_NATIVE_CHAR;

levl_info.dims[0][0] = 1;
levl_info.dims[1][0] = 1;
levl_info.dims[2][0] = 1;
levl_info.dims[3][0] = 1;

levl_info.datasize = sizeof(Levl_Data);

status = HE5_PTdeflevel(pointID2, "Observations",
&levl_info);

for (i = 0; i < 4; i++)
    free(levl_info.fieldname[i]);

```

**FORTTRAN** See Example 2 from Section 7.1.1.2 of Volume 1 (Overview and Examples)



## Define Linkage Field Between Two Levels

---

### HE5\_PTdeflinkage

herr\_t HE5\_PTdeflinkage(hid\_t *pointID*, char \**parent*, char \**child*, char \**linkfield*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*parent* IN: Name of parent level

*child* IN: Name of child level

*linkfield* IN: Name of (common) link field

Purpose Defines a link field between two (adjacent) levels.

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

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

**Note** The defining of a linkage is necessary if more than one level is defined.

Example In this example we define the *ID* field as the link between the two levels defined previously in the *HE5\_PTdeflevel* routine.

```
status = HE5_PTdeflinkage(pointID2, "Desc-Loc",  
"Observations", "ID");
```

FORTRAN integer function  
he5\_ptdeflinkage(*pointid*,*levelname1*,*levelname2*,*linkname*)

integer *pointid*,*status*

character \*(\*) *linkname*,*levelname1*,*levelname2*

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

```
status = he5_ptdeflinkage(pointid, "Desc-Loc",  
"Observations", "ID")
```

# Detach from Point Structure

---

## HE5\_PTdetach

herr\_t HE5\_PTdetach(hid\_t *pointID*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

Purpose Detaches from point data set.

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

Description This routine should be run before exiting from the point file for every point opened by *HE5\_PTcreate* or *HE5\_PTattach*.

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

```
status = HE5_PTdetach(pointID);
```

FORTRAN integer function he5\_ptdetach(*pointid*)

integer *pointid,status*

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

```
status = he5_ptdetach(pointid)
```

## Return Linkage Field to Following Level

---

### HE5\_PTfwdlinkinfo

herr\_t HE5\_PTfwdlinkinfo(hid\_t *pointID*, int *level*, char \**linkfield*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*level* IN: Point level (0-based)

*linkfield* OUT: Link field

Purpose Returns the link field to the following level.

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

Description This routine returns the link field to the following level\*.

Example In this example, we return the link field connecting the *Desc-Loc* level to the following *Observations* level. (These levels are defined in the *HE5\_PTdeflevel* routine.)

```
status = HE5_PTfwdlinkinfo(pointID2, 1, linkfield);
```

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

FORTTRAN integer function he5\_ptfwdlinkinfo(*pointid*,*level*,*linkfield*)

integer *pointid*,*status*

character \*(\*) *linkfield*

integer *level*

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

```
level = 1
```

```
status = he5_ptfwdlinkinfo(pointid, level, linkfield)
```

\* **Note:** Forward linkage field records will be (-1,-1) if the records in the child level are not ordered monotonically.

# Return Level Name

---

## HE5\_PTgetlevelname

herr\_t HE5\_PTgetlevelname(hid\_t *pointID*, int *level*, char \**levelname*, long \**strbufsize*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*level* IN: Point level (0-based)

*levelname* 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 or 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 level name, 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 0<sup>th</sup> level of the second point defined in the *HE5\_PTdeflevel* section:

```
status = HE5_PTgetlevelname(pointID2, 0, levelname,
&strbufsize);
```

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

FORTTRAN integer function he5\_ptgetlevelname(*pointid*,*level*,*levelname*,*strbufsz*)

integer *pointid*,*status*,*level*

character \*(\*) *levelname*

integer\*4 *strbufsz*

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

```
level = 0
```

```
status = he5_ptgetlevelname(pointid, level, levelname,
strbufsz)
```

## Return Record Numbers Related to Level

---

### HE5\_PTgetrecnums

herr\_t HE5\_PTgetrecnums(hid\_t *pointID*, int *inlevel*, int *outlevel*, hsize\_t *inNrec*,  
hssize\_t *inRecs*[], hsize\_t \**outNrec*, hssize\_t *outRecs*[])

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_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 or 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 *HE5\_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 = HE5_PTgetrecnums(pointID2, inLevel, outLevel, nrec,  
recs, &outNrec, outRecs);
```

FORTTRAN Not available with this release.

# Return Information About Group Attribute

---

## HE5\_PTgrpattrinfo

herr\_t HE5\_PTgrpattrinfo(hid\_t *pointID*, const char \**attrname*, hid\_t \**numbertype*, hsize\_t \**count*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*attrname* IN: Attribute name

*numbertype* OUT: Number type of attribute. See Appendix A for interpretation of number types.

*count* OUT: Number of elements in attribute

Purpose Returns information about group attribute associated with the point “Data” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description This routine returns number type and number of elements (count) of an attribute associated with the point “Data” group.

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

```
status = HE5_PTgrpattrinfo(pointID,  
"GroupFloat", &nt, &count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

FORTRAN integer function he5\_ptgrpattrinfo(*pointid*,*attrname*,*ntype*,*count*)

integer *pointid*,*status*

integer\*4 *count*

integer *ntype*

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

```
status = he5_ptgrpattrinfo(pointid, "GroupFloat", ntype,  
count)
```

## Retrieve Information About Point Attributes

---

### HE5\_PTinqattrs

long HE5\_PTinqattrs(hid\_t *pointID*, char \**attrlist*, long \**strbufsize*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach  
*attrlist* OUT: Attribute list (entries separated by commas)  
*strbufsize* OUT: String length of attribute list

Purpose Retrieve information about object attributes defined in a specific point. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each attribute name separated by commas. If *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. In the first call, set the parameter *attrlist* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_PTinqattrs(pointID, NULL, strbufsize);
```

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

```
attrlist = (char *)malloc((strbufsize+1) * sizeof(char));  
nattr = HE5_PTinqattrs(pointID, attrlist, strbufsize);
```

The variable, *attrlist*, will be set to: "*attrOne,attr\_2*".

FORTRAN integer\*4      function he5\_ptinqattrs(*pointid*,*attrlist*,*strbufsz*)  
integer          *pointid*  
character \*(\*) *attrlist*  
integer\*4        *nattr*, *strbufsz*

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

```
nattr = he5_ptinqattrs(pointid, attrlist, strbufsz)
```

# Return Data Type Information for a Level

---

## HE5\_PTinqdatatype

herr\_t HE5\_PTinqdatatype(hid\_t *pointID*, const char \**levelname*, const char \**attrname*, int *fieldgroup*, hid\_t \**datatype*, H5T\_class\_t \**classid*, H5T\_order\_t \**order*, size\_t \**size*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*levelname* IN: Level name

*attrname* IN: Attribute name

*fieldgroup* IN: Field group flag: HE5\_HDFE\_DATAGROUP - 1  
HE5\_HDFE\_ATTRGROUP - 2  
HE5\_HDFE\_GRPATTRGROUP - 3  
HE5\_HDFE\_LOCATTRGROUP - 4

*datatype* OUT: Data type ID

*classID* OUT: Data type class ID

*order* OUT: Data type byte order

*size* OUT: Data type size (in bytes)

Purpose Returns data type information about specified level in point.

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

Description This routine returns information about level data in a point.

Example In this example we return the data type information for the *Observations* level in the point defined in the *HE5\_PTdeflevel* routine.

```
status = HE5_PTinqdatatype(pointID, "Observations", NULL,  
fieldgroup, &datatype, &classid, &order, &size);
```

FORTRAN integer function  
he5\_ptinqdatatype(*pointid*,*levelname*,*attrname*,*fldgrp*,*dtype*,*classid*,*order*,  
*size*)

integer *pointid*,*status*

integer *dtype*,*classid*,*order*



integer\*4     *size*

character \*(\*) *levelname*

integer        *HE5\_HDFE\_DATAGROUP*

parameter     (*HE5\_HDFE\_DATAGROUP=1*)

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

```
status = he5_ptinqdatatype(pointid1, "Observations", " ",  
HE5_HDFE_DATAGROUP, dtype, classid, order, size)
```

# Retrieve Information About Group Attributes

---

## HE5\_PTinqgrpattrs

long HE5\_PTinqgrpattrs(hid\_t *pointID*, char \**attrlist*, long \**strbufsize*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*attrlist* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

**Purpose** Retrieve information about group attributes defined in point “Data” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

**Return value** Number of attributes found if successful or FAIL (-1) otherwise.

**Description** The attribute list is returned as a string with each attribute name separated by commas. If *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 the “Data” group of point structure. In the first call, set the parameter *attrlist* to NULL. We assume that there are two attributes stored, *GrpAttrOne* and *GrpAttr\_2*:

```
nattr = HE5_PTinqgrpattrs(pointID, NULL, strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 20.

```
attrlist = (char *)malloc((strbufsize+1) * sizeof(char));
```

```
nattr = HE5_PTinqgrpattrs(pointID, attrlist, strbufsize);
```

The variable, *attrlist*, will be set to: "GrpAttrOne,GrpAttr\_2".

**FORTTRAN** integer\*4 function he5\_ptinqgrpattrs(*pointid*,*attrlist*,*strbufsz*)

integer *pointid*

character \*(\*) *attrlist*

integer\*4 *nattr*,*strbufsz*

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

```
nattr = he5_ptinqgrpattrs(pointid, attrlist, strbufsz)
```

# Retrieve Information About Level Attributes

---

## HE5\_PTinqlocattr

long HE5\_PTinqlocattr(hid\_t *pointID*, const char \**levelname*, char \**attrlist*, long \**strbufsize*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach  
*levelname* IN: Level name  
*attrlist* OUT: Attribute list (entries separated by commas)  
*strbufsize* OUT: String length of attribute list  
Purpose Retrieve information about local attributes defined for a specified level in a point. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each attribute name separated by commas. If *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 local attributes defined for the *Observations* level in a point structure. In the first call, set the parameter *attrlist* to NULL. We assume that there are two attributes stored, *LocAttrOne* and *LocAttrTwo*:

```
nattr = HE5_PTinqlocattr(pointID, "Observations", NULL,  
strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 21.

```
attrlist = (char *)malloc((strbufsize+1) * sizeof(char));  
nattr = HE5_PTinqlocattr(pointID, levelname, attrlist,  
strbufsize);
```

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

*"LocAttrOne,LocAttrTwo"*.

FORTRAN integer\*4 function  
he5\_ptinqlocattr(*pointid,levelname,attrlist,strbufsz*)

```
integer      pointid  
character *(*) levelname, attrlist  
integer*4    nattr, strbufsz
```

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

```
nattr = he5_ptinqlocattrs(pointid, levelname, attrlist,  
strbufsz)
```

# Retrieve Point Structures Defined in HDF-EOS File

---

## HE5\_PTinqpoint

int HE5\_PTinqpoint(const char \* *filename*, char \**pointlist*, long \**strbufsize*)

*filename* 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 if successful or FAIL (-1) otherwise.

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, *Point.he5*. In the first call, set the parameter *pointlist* to NULL. We assume that there are two points stored, *PointOne* and *Point\_2*:

```
npoint = HE5_PTinqpoint("Point.he5", NULL, strbufsize);
```

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

```
pointlist = (char *)malloc((strbufsize+1) * sizeof(char));
```

```
npoint = HE5_PTinqpoint("Point.he5", pointlist, strbufsize);
```

The variable, *pointlist*, will be set to: "*PointOne,Point\_2*".

FORTRAN integer function he5\_ptinqpoint(*filename,pointlist,strbufsz*)

integer *npoint*

character \*(\*) *pointlist*

integer\*4 *strbufsz*

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

```
npoint = he5_ptinqpoint("Point.he5", pointlist, strbufsz)
```

## Return Index Number of a Named Level

---

### HE5\_PTlevelindx

int HE5\_PTlevelindx(hid\_t *pointID*, const char \**levelname*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*levelname* IN: Level Name

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

Return value Returns the level index if successful or 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 *HE5\_PTdeflevel*.

```
levindx = HE5_PTlevelindx(pointID2, "Observations");
```

The *levindx* variable will have the value 1.

FORTTRAN integer function he5\_ptlevelindx(*pointid*,*levelname*)

integer *pointid*,*levindx*

character \*(\*) *levelname*

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

```
levindx = he5_ptlevelindx(pointid, "Observations")
```

# Return Information on Fields in a Given Level

---

## HE5\_PTlevelinfo

herr\_t HE5\_PTlevelinfo(hid\_t *pointID*, int *level*, HE5\_CmpDTSinfo \*info)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*level* IN: Point level (0-based)

*info* OUT: C-data structure containing the level information

Purpose Returns information on fields in a given level.

Return value Returns SUCCEED (0) if successful or 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.

```
HE5_CmpDTSinfo      lev0_info;  
  
status = HE5_PTlevelinfo(pointID2, 0, &lev0_info);
```

The *lev0\_info.nfields* data member will be set to 5. The *lev0\_info.fieldname* array will be "Time,Longitude,Latitude,Channel,Value".

FORTTRAN See Example 4 from Section 7.1.1.2 of Volume 1 (Overview of Examples).

# Return Information About Level Attribute

---

## HE5\_PTlocattrinfo

herr\_t HE5\_PTlocattrinfo(hid\_t *pointID*, const char \**levelname*, const char \**attrname*, hid\_t \* *numbertype*, hsize\_t \**count*)

<i>pointID</i>	IN: Point ID returned by HE5_PTcreate or HE5_Ptattach
<i>levelname</i>	IN: Level name
<i>attrname</i>	IN: Attribute name
<i>numbertype</i>	OUT: Number type of attribute. See Appendix A for interpretation of number types.
<i>count</i>	OUT: Number of elements in attribute
Purpose	Returns information about local attribute in a specified level. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise.
Description	This routine returns number type and number of elements (count) of an attribute associated with a specified level.
Example	In this example, we return information about the <i>LocalFloat</i> attribute associated with the level <i>Observations</i> .

```
status = HE5_PTattrinfo(pointID,  
"Observations", "LocalFloat", &nt, &count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

FORTRAN integer function  
he5\_ptlocattrinfo(*pointid*, *levelname*, *attrname*, *ntype*, *count*)  
integer *pointid*, *status*, *ntype*  
character \*(\*) *levelname*, *attrname*  
integer\*4 *count*

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

```
status = he5_ptlocattrinfo(pointid, "Observations",  
"LocalFloat", ntype, count)
```



## Return Number of Fields Defined in a Level

---

### HE5\_PTnfields

int HE5\_PTnfields(hid\_t *pointID*, int *level*, char \**fieldlist*, long \**strbufsize*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*level* IN: Level number (0-based)

*fieldlist* OUT: Field list (entries separated by commas)

*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 or 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. In the first call, set the parameter *fieldlist* to NULL:

```
nflds = HE5_PTnfields(pointID2, 0, NULL, &strbufsize);  
fieldlist = (char *)malloc((strbufsize+1) * sizeof(char));  
nflds = HE5_PTnfields(pointID2, 0, fieldlist, &strbufsize);  
The nflds variable will be 5 and the strbufsize variable equal to 38.
```

FORTRAN integer function he5\_ptnfields(*pointid2,level,fieldlist,strbufsz*)

integer *pointid2,level,nflds*

character \*(\*) *fieldlist*

integer\*4 *strbufsz*

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

```
level = 0  
nflds = he5_ptnfields(pointid2, level, fieldlist, strbufsz)
```

# Return Number of Levels in a Point Structure

---

## HE5\_PTnlevels

int HE5\_PTnlevels(hid\_t *pointID*)

<i>pointID</i>	IN: Point ID returned by HE5_PTcreate or HE5_PTattach
Purpose	Returns number of levels in a point.
Return value	Returns number of levels if successful or 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: <pre>nlevels = HE5_PTnlevels(pointID2);</pre> The <i>nlevels</i> variable will be 2.

FORTRAN integer function he5\_ptnlevels(*pointid*)  
integer *pointid,nlevels*

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

```
nlevels = he5_ptnlevels(pointid)
```

# Return Number of Records in a Given Level

---

## HE5\_PTnrecs

hsize\_t HE5\_PTnrecs(hid\_t *pointID*, int *level*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_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 or 0 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 = HE5_PTnrecs(pointID2, 0);
```

FORTRAN integer function he5\_ptnrecs(*pointid*,*level*)

integer *pointid2*,*level*

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

```
level = 0
```

```
status = he5_ptnrecs(pointid2, level)
```

# Open HDF-EOS File

---

## HE5\_PTopen

hid\_t HE5\_PTopen(const char \*filename, uintn access)

*filename* IN: Complete path and filename for the file to be opened  
*access* IN: H5F\_ACC\_RDONLY, H5F\_ACC\_RDWR or H5F\_ACC\_TRUNC  
Purpose Opens or creates HDF-EOS file in order to create, read, or write a point.  
Return value Returns the point file ID (fid) if successful or FAIL (-1) otherwise.  
Description This routine creates a new file or opens an existing one, depending on the access parameter.

Access codes:

H5F\_ACC\_RDONLY Open for read only. If file does not exist, error

H5F\_ACC\_RDWR Open for read/write. If file does not exist, error

H5F\_ACC\_TRUNC If file exists, delete it, then open a new file for read/write

Example In this example, we create a new point file named, *Point.he5*. It returns the file handle, *fid*.

```
fid = HE5_PTopen("Point.he5", H5F_ACC_TRUNC);
```

FORTRAN integer function he5\_ptopen(*filename,flag*)

integer *fid*

character \*(\*) *filename*

integer *HE5F\_ACC\_TRUNC*

parameter (*HE5F\_ACC\_TRUNC=102*)

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

```
fid = he5_ptopen("Point.he5", HE5F_ACC_TRUNC)
```

## Read Point Attribute

---

### HE5\_PTreadattr

herr\_t HE5\_PTreadattr(hid\_t *pointID*, const char \**attrname*, void \**datbuf*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads attribute from a specific point. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Returns SUCCEED (0) if successful or 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 floating point attribute with the name "ScalarFloat":

```
status = HE5_PTreadattr(pointID, "ScalarFloat", &attr_val);
```

FORTTRAN integer function he5\_ptreadatt(*pointid*,*attrname*,*buffer*)

integer *pointid*,*status*

character \*(\*) *attrname*

<valid type> *buffer*(\*)

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

```
status = he5_ptreadatt(pointid, "ScalarFloat", buffer)
```

# Read Point Group Attribute

---

## HE5\_PTreadgrpattr

herr\_t HE5\_PTreadgrpattr(hid\_t *pointID*, const char \**attrname*, void \**datbuf*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads group attribute associated with the “Data” group in a point. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Returns SUCCEED (0) if successful or 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 floating point attribute with the name "GroupFloat":

```
status = HE5_PTreadgrpattr(pointID, "GroupFloat",
&attr_val);
```

FORTRAN integer function he5\_ptreadgrpattr(*pointid*,*attrname*,*buffer*)

integer *pointid*,*status*

character \*(\*) *attrname*

<valid type> *buffer*(\*)

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

```
status = he5_ptreadgrpattr(pointid, "GroupFloat", buffer)
```

## Read Point Level Attribute

---

### HE5\_PTreadlocattr

herr\_t HE5\_PTreadlocattr(hid\_t *pointID*, const char \**levelname*, const char \**attrname*, void \**datbuf*)

<i>pointID</i>	IN: Point ID returned by HE5_PTcreate or HE5_PTattach
<i>levelame</i>	IN: Level name
<i>attrname</i>	IN: Attribute name
<i>datbuf</i>	OUT: Buffer allocated to hold attribute values
Purpose	Reads local attribute associated with a specified level in a point. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).
Return value	Returns SUCCEED (0) if successful or 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	<p>In this example, we read floating point attribute with the name "LocalFloat" defined in the <i>Observations</i> level:</p> <pre>status = HE5_PTreadlocattr(pointID, "Observations", "LocalFloat", &amp;attr_val);</pre>
FORTRAN	<p>integer function he5_ptreadlocattr(<i>pointid</i>,<i>levelname</i>,<i>attrname</i>,<i>buffer</i>)</p> <p>integer            <i>pointid</i>,<i>status</i></p> <p>character *(*) <i>levelname</i>,<i>attrname</i></p> <p>&lt;valid type&gt;    <i>buffer</i>(*)</p> <p>The equivalent <i>FORTRAN</i> code for the example above is:</p> <pre>status = he5_ptreadlocattr(pointid, "Observations", "LocalFloat", buffer)</pre>

# Read Records From a Point Level

---

## HE5\_PTreadlevel

herr\_t HE5\_PTreadlevel(hid\_t *pointID*, int *level*, HE5\_CmpDTSinfo \**inStruct*, size\_t \**size*, void \**buffer*)

<i>pointID</i>	IN: Point ID returned by HE5_PTcreate or HE5_PTattach
<i>level</i>	IN: Level to read (0-based)
<i>inStruct</i>	IN: C-data structure containing information about specified level.
<i>size</i>	IN: Size (in bytes) of data structure to read data to.
<i>buffer</i>	OUT: Buffer to store data
Purpose	Reads data from a point level.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or level number.
Description	This routine reads data from the specified fields and records of a single level in a point. An appropriate read buffer must be defined by the user.
Example	In this example we read records from the first level in the point referred to by the point ID, <i>pointID1</i> . User should define data structure to store the output data, first. Suppose the user defined data structure to read the output data to is "Sensor".

```
CmpDTSinfo lev0_info;
CmpDTSinfo input_info;
Sensor      *buffer;

/* Get all necessary information about level first */
status = HE5_Ptlevelinfo(pointID1, 0, &lev0_info);
/* Set up input data structure and calculate the data size */
nrecs = HE5_Ptnrecs(pointID1, 0);
buffer = (Sensor *)calloc(nrecs, sizeof(Sensor));
status = HE5_PTreadlevel(pointID1, 0, &lev0_info, datasize,
buffer);
```

FORTRAN See Example 4 from Section 7.1.1.2 of Volume 1 (Overview of Examples).



## Update Records in a Point Structure

---

### HE5\_PTupdatelevel

herr\_t HE5\_PTupdatelevel(hid\_t *pointID*, int *level*, char\* *fieldlist*, hsize\_t *nrec*,  
hssize\_t *recs*[], void \**data*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_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: Data buffer to be written

Purpose Updates (corrects) data to a point level.

NOTE: Currently updating of a whole record is supported.

Return value Returns SUCCEED (0) if successful or 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 for the field *Concentration* in first level in the point referred to by the point ID, *pointID1*.

```
hsize_t recs[3] = {0,2,3};  
  
/* Fill Data Buffer */  
  
status = HE5_PTupdatelevel(pointID1, 0, "Concentration", 3,  
recs, datbuf);
```

The user may update a single record or all records .

FORTTRAN See Example 5 from Section 7.1.1.2 of Volume 1 (Overview of Examples).

# Write/Update Point Attribute

---

## HE5\_PTwriteattr

herr\_t HE5\_PTwriteattr(hid\_t *pointID*, const char \**attrname*, int *ntype*, hsize\_t *count*, void \**datbuf*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_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 specific point. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

**Return value** Returns SUCCEED (0) if successful or 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 floating point attribute with the name "ScalarFloat" and the value 3.14:

```
attr_val = 3.14;
```

```
status = HE5_PTwriteattr(pointid, "ScalarFloat",  
H5T_NATIVE_FLOAT, 1, &attr_val);
```

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

```
attr_val = 3.14159;
```

```
status = HE5_PTwriteattr(pointid, "ScalarFloat",  
H5T_NATIVE_FLOAT, 1, &attr_val);
```

**FORTRAN** integer function he5\_ptwriteattr(*pointid*,*attrname*,*ntype*,*count*,*buffer*)

integer *pointid*,*status*,*ntype*

character \*(\*) *attrname*

```
integer*4    count  
<valid type> buffer(*)  
integer      HE5T_NATIVE_FLOAT  
parameter    (HE5T_NATIVE_FLOAT=10)
```

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

```
count = 1  
  
status = he5_ptwriteattr(pointid, "ScalarFloat",  
HE5T_NATIVE_FLOAT, count, buffer)
```

# Write/Update Point Group Attribute

---

## HE5\_PTwritegrpattr

herr\_t HE5\_PTwritegrpattr(hid\_t *pointID*, const char \**attrname*, int *ntype*, hsize\_t *count*, void \**datbuf*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_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 group attribute associated with the “Data” group in a point. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

**Return value** Returns SUCCEED (0) if successful or 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 floating point group attribute with the name "GroupFloat" and the value 3.14:

```
attr_val = 3.14;
```

```
status = HE5_PTwritegrpattr(pointid, "GroupFloat",  
H5T_NATIVE_FLOAT, 1, &attr_val);
```

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

```
attr_val = 3.14159;
```

```
status = HE5_PTwritegrpattr(pointid, "GroupFloat",  
H5T_NATIVE_FLOAT, 1, &attr_val);
```

**FORTRAN** integer function he5\_ptwritegrpattr(*pointid*,*attrname*,*ntype*,*count*,*buffer*)

integer *pointid*,*status*,*ntype*

character \*(\*) *attrname*  
integer\*4 *count*  
<valid type> *buffer*(\*)  
integer *HE5T\_NATIVE\_FLOAT*  
parameter (*HE5T\_NATIVE\_FLOAT=10*)

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

```
count = 1  
  
status = he5_ptwritegrpattr(pointid, "GroupFloat",  
HE5T_NATIVE_FLOAT, count, buffer)
```

## Write New Records to a Point Level

---

### HE5\_PTwritelevel

herr\_t HE5\_PTwritelevel(hid\_t *pointID*, int *level*, hsize\_t *nrec*[], size\_t *\*size*, void *\*data*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*level* IN: Level to write (0-based)

*nrec* IN: Number of records to write

*size* IN: Data size (bytes) to write

*data* IN: Data buffer to be written to the level

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

Return value Returns SUCCEED (0) if successful or 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 buffer should be represented by the array of C-data type structures. The structure type should be consistent with that used in *HE5\_PTdeflevel()*.

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 */  
  
/* Calculate the data size (bytes) */  
status = HE5_PTwritelevel(pointID1, 0, 5, datasize, datbuf);
```

FORTTRAN See Example 3 from Section 7.1.1.2 of Volume 1 (Overview of Examples).

# Write/Update Point Level Attribute

---

## HE5\_PT writelocattr

herr\_t HE5\_PT writelocattr(hid\_t *pointID*, const char \**levelname*, const char \**attrname*, int *ntype*, hsize\_t *count*, void \**datbuf*)

*pointID* IN: Point ID returned by HE5\_PTcreate or HE5\_PTattach

*levelname* IN: Level name

*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 local attribute associated with a specified level in a point. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

**Return value** Returns SUCCEED (0) if successful or 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 floating point attribute with the name "LocalFloat" and the value 3.14 associated with the level "Observations":

```
attr_val = 3.14;
```

```
status = HE5_PT writelocattr(pointid, "Observations",  
"LocalFloat", H5T_NATIVE_FLOAT, 1, &attr_val);
```

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

```
attr_val = 3.14159;
```

```
status = HE5_PT writelocattr(pointid, "Observations",  
"LocalFloat", H5T_NATIVE_FLOAT, 1, &attr_val);
```

**FORTRAN** integer function  
he5\_ptwritelocattr(*pointid*,*levelname*,*attrname*,*ntype*,*count*,*buffer*)

integer        *pointid,status,ntype*

character \*(\*) *attrname,levelname*

integer\*4     *count*

<valid type> *buffer(\*)*

integer        *HE5T\_NATIVE\_FLOAT*

parameter     (*HE5T\_NATIVE\_FLOAT=10*)

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

```
count = 1
```

```
status = he5_ptwritelocattr(pointid, "Observations",  
"LocalFloat", HE5T_NATIVE_FLOAT, count, buffer)
```



### **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.

## Return Information About an Alias

---

### HE5\_SWaliasinfo

herr\_t HE5\_SWaliasinfo(hid\_t *swathID*, int *fldgroup*, const char \**aliasname*, int \**length*, char \**buffer*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fldgroup* IN: Field group flag

*aliasname* IN: Name of alias to retrieve information about

*length* IN/OUT: Size of buffer in bytes

*buffer* OUT: Buffer with original field name

Purpose Return information about an alias

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

Description This routine returns a buffer size and the buffer with an original field name.

Example In this example, we return the buffer size and the original field name *Temperature*. In the first call, set *buffer* to NULL and *length* is an output parameter. In the second call, *length* is an input parameter.

```
status = HE5_SWaliasinfo(swathID, HE5_HDFE_DATAGROUP,
aliasname, length, NULL);
namebuffer = (char *)calloc(length + 1, sizeof(char));
status = HE5_SWaliasinfo(swathID, HE5_HDFE_DATAGROUP,
aliasname, length, namebuffer);
```

FORTRAN integer function he5\_swaliasinfo (*swathid*, *fldgroup*, *aliasname*, *length*,  
*buffer*)

```
integer      swathid,status
character(*) fldgroup
character(*) aliasname
integer(*)  length
character(*) buffer
```

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

```
aliaslist = "temps 0 to 30"
status = he5_swaliasinfo(swathid, HE5_HDFE_DATAGROUP,
aliaslist, length, buffer)
```

## Attach to an Existing Swath Structure

---

### HE5\_SWattach

hid\_t HE5\_SWattach(hid\_t *fid*, const char \**swathname*)

*fid* IN: Swath file ID returned by HE5\_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 or FAIL (-1) otherwise. Typical reasons for failure are an improper swath file id or swath name.

Description This routine attaches to the swath using the *swathname* parameter as the identifier.

Example In this example, we attach to the previously created swath, "ExampleSwath", within the HDF-EOS file, *Swath.he5*, referred to by the handle, *fid*:

```
swathID = HE5_SWattach(fid, "ExampleSwath");
```

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

FORTTRAN integer function he5\_swattach(*fid*,*swathname*)

integer *fid*

character\*(\*) *swathname*

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

```
swathid = he5_swattach(fid, "ExampleSwath")
```

## Return Information About a Swath Attribute

---

### HE5\_SWattrinfo

herr\_t HE5\_SWattrinfo(hid\_t *swathID*, const char \**attrname*, hid\_t \**ntype*, hsize\_t \**count*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrname* IN: Attribute name

*ntype* OUT: Number type of attribute

*count* OUT: Number of elements in attribute

Purpose Returns information about an attribute in a specific swath. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

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

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

```
status = HE5_SWattrinfo(swathID, "ScalarFloat",&nt,&count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

FORTRAN integer function he5\_swattrinfo(*swathid*, *attrname*, *ntype*, *count*)

integer *swathid*

character\*(\*) *attrname*

integer *ntype*

integer\*4 *count*

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

```
status = he5_swattrinfo(swathid, "ScalarFloat",ntype,count)
```

# Retrieve Chunking Information about a Swath Field

---

## HE5\_SWchunkinfo

herr\_t HE5\_SWchunkinfo(hid\_t *swathID*, char \**fieldname*, int \**chunk\_rank*,  
hsize\_t *chunk\_dims*[])

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fieldname* IN: Fieldname

*chunk\_rank* OUT: The number of chunking dimensions

*chunk\_dims* OUT: Array containing the chunking dimension sizes of the field

Purpose Retrieve chunking information about a specific field in the swath.

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

Description This routine returns the chunking rank and chunking dimensions for a given field.

Example In this example, we retrieve the chunking information about the *Count* fields:

```
status = HE5_SWchunkinfo(swathID, "Count", &chunk_rank,  
chunk_dims);
```

The return parameters will have the following values:

```
chunk_rank=2, chunk_dims[2]={100,360}
```

FORTTRAN integer function he5\_swchunkinfo(*swathid*, *fieldname*, *chunk\_rank*,  
*chunk\_dims*)

integer *swathid*

character\*(\*) *fieldname*

integer *chunk\_rank*

integer\*4 *chunk\_dims*(\*)

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

```
status = he5_swchunkinfo(swathid, "Count", chunk_rank,  
chunk_dims)
```

The return parameters will have the following values:

*chunk\_rank=2, chunk\_dims[3]={360,100}*

Note that the dimensions array is in FORTRAN order.

# Close an HDF-EOS File

---

## HE5\_SWclose

herr\_t HE5\_SWclose(hid\_t *fid*)

*fid* IN: Swath file ID returned by HE5\_SWopen

Purpose Closes a file opened by HE5\_SWopen.

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

Description This routine closes the HDF-EOS Swath file.

Example

```
status = HE5_SWclose(fid);
```

FORTRAN integer function he5\_swclose(*fid*)

integer *fid*

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

```
status = he5_swclose(fid)
```

# Retrieve Compression Information for Field

---

## HE5\_SWcompinfo

herr\_t HE5\_SWcompinfo(hid\_t *swathID*, const char \**fieldname*, int \**compcode*,  
int *compparm*[])

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_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 or 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 *HE5\_SWdefcomp* function:

```
status = HE5_SWcompinfo(swathID, "Opacity", &compcode,  
compparm);
```

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

FORTRAN integer function he5\_swcompinfo(*gridid*,*fieldname* *compcode*, *compparm*)

integer *swathid*

character\*(\*) *fieldname*

integer *compcode*

integer *compparm*(\*)

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

```
status = he5_swcompinfo(swathid, 'Opacity', compcode,  
compparm)
```

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



## Create a New Swath Structure

---

### HE5\_SWcreate

hid\_t HE5\_SWcreate(hid\_t *fid*, const char \**swathname*)

*fid* IN: Swath file ID returned by HE5\_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 or FAIL (-1) otherwise.

Description The swath is created as a Group within the HDF-EOS file with the name *swathname*.

Example In this example, we create a new swath structure, *ExampleSwath*, in the previously created file, *Swath.he5*.

```
swathID = HE5_SWcreate(fid, "ExampleSwath");
```

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

FORTRAN integer function he5\_swcreate(*fid*,*swathname*)

integer *fid*

character\*(\*) *swathname*

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

```
swathid = he5_swcreate(fid, "ExampleSwath")
```

# Define a Longitude-Latitude Box Region for a Swath

---

## HE5\_SWdefboxregion

hid\_t HE5\_SWdefboxregion(hid\_t *swathID*, double *cornerlon*[], double *cornerlat*[], int *mode*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_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 or FAIL (-1) otherwise.

Description This routine defines a longitude-latitude box region for a swath. It returns a swath region ID which is used by the *HE5\_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" (HE5\_HDFE\_MIDPOINT), or 2) either of its endpoints is within the longitude-latitude "box" (HE5\_HDFE\_ENDPOINT), or 3) any point of the cross track is within the longitude-latitude "box" (HE5\_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 = HE5_SWdefboxregion(swathID, cornerlon, cornerlat,  
HE5_HDFE_MIDPOINT);
```

FORTTRAN integer function `he5_swdefboxreg(swathid, cornerlon, cornerlat, mode)`

integer *swathid*

real\*8 *cornerlon*(\*)

real\*8 *cornerlat*(\*)

integer *mode*

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

```
parameter (HE5_HDFE_MIDPOINT=0)

cornerlon(1) = 3.
cornerlat(1) = 5.
cornerlon(2) = 7.
cornerlat(2) = 12.

regionid = he5_swdefboxreg(swathid, cornerlon, cornerlat,
HE5_HDFE_MIDPOINT)
```

# Define Chunking Parameters

---

## HE5\_SWdefchunk

herr\_t HE5\_SWdefchunk(hid\_t *swathID*, int *chunk\_rank*, const hsize\_t  
\**chunk\_dims*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*chunk\_rank* IN: The number of chunk dimensions (a number other than zero)

*chunk\_dims* IN: Chunk dimensions (NULL cannot be used)

Purpose Defines chunking for subsequent field definitions

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

Description This routine defines the chunking dimensions for fields defined following this function call, analogous to the procedure for setting the field compression scheme using *HE5\_SWdefcomp*. The number of chunk dimensions and subsequent field dimensions must be the same.

Example We will define chunking for a two-dimensional field of size  
2400 x 3600 .

```
chunk_dims[0] = 100;
```

```
chunk_dims[1] = 360;
```

```
status = HE5_SWdefchunk(swathID, 2, chunk_dims);
```

FORTRAN integer function he5\_swdefchunk(*swathid*, *chunk\_rank*,*chunk\_dims*)

integer *swathid*

integer *chunk\_rank*

integer\*4 *chunk\_dims*(\*)

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

```
chunk_dims(1) = 360
```

```
chunk_dims(2) = 100
```

```
chunk_rank = 2
```

```
status = he5_swdefchunk(swathid, chunk_rank, chunk_dims)
```

# Define Compression with Data Chunking

---

## HE5\_SWdefcomchunk

herr\_t HE5\_SWdefcomchunk(hid\_t *swathID*, int *compcode*, int \**compparm*, int *ndims*, const hsize\_t \**dim*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*compcode* IN: Compression method flag

*compparm* IN: Array of compression parameters

*ndims* IN: Rank of a field to compress(a number other than zero)

*dim* IN: Array of sizes of chunk (NULL cannot be used)

Purpose Compress the data field

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

Description This function allows the user to set compression for a data field with automatic chunking

Example In this example, we set (DEFLATE) compression for a field that is defined right after this call

```
ndims      = 2
compcode   = 4;
compparm[0] = 6;
dim[0]     = 100;
dim[1]     = 200;

status = HE5_SWdefcomchunk(swathID, compcode, compparm,
ndims, dim);
```

FORTRAN integer function he5\_swdefcomch(swathid,compcode, compparm,  
ndims,dim)

```
integer      swathid
integer      compcode
integer      compparm(*)
integer      ndims
integer*4    dim(*)
```

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

```
compcode      = 4
compparm(1)   = 6
ndims         = 2
dim(1)        = 200
dim(2)        = 100
status = he5_swdefcomch(swathid, compcode, compparm, ndims,
dim)
```

# Set Swath Field Compression

---

## HE5\_SWdefcomp

herr\_t HE5\_SWdefcomp(hid\_t *swathID*, int *compcode*, int *\*compparm*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*compcode* IN: HDF compression code

*compparm* IN: Compression parameters (if applicable)

Note: Shuffling, szip, and deflate compression methods are available in this release.

Purpose Sets the field compression for all subsequent field definitions.

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

Description This routine sets the HDF field compression for subsequent swath field definitions. **The routine HE5\_SWdefchunk() must be called first, otherwise HE5\_SWdefcomp() doesn't work. Also the compression does not apply to one-dimensional fields.** The compression schemes currently supported are: deflate (gzip) (HE5\_HDFE\_COMP\_DEFLATE=4), compression exactly as in hardware (HE5\_HDFE\_COMP\_SZIP\_CHIP = 5), allowing k split = 13 compression mode (HE5\_HDFE\_COMP\_SZIP\_K13 = 6), entropy coding method (HE5\_HDFE\_COMP\_SZIP\_EC = 7), nearest neighbor coding method (HE5\_HDFE\_COMP\_SZIP\_NN = 8), allowing k split = 13 compression mode or entropy coding method (HE5\_HDFE\_COMP\_SZIP\_K13orEC = 9), allowing k split = 13 compression mode or nearest neighbor coding method (HE5\_HDFE\_COMP\_SZIP\_K13orNN = 10), shuffling + deflate(gzip) (HE5\_HDFE\_COMP\_SHUF\_DEFLATE = 11), shuffling + compression exactly as in hardware (HE5\_HDFE\_COMP\_SHUF\_SZIP\_CHIP = 12), shuffling + allowing k split = 13 compression mode (HE5\_HDFE\_COMP\_SHUF\_SZIP\_K13 = 13), shuffling + entropy coding method (HE5\_HDFE\_COMP\_SHUF\_SZIP\_EC = 14), shuffling + nearest neighbor coding method (HE5\_HDFE\_COMP\_SHUF\_SZIP\_NN = 15), shuffling + allowing k split = 13 compression mode or entropy coding method (HE5\_HDFE\_COMP\_SHUF\_SZIP\_K13orEC = 16), shuffling + allowing k split =13 compression mode or nearest neighbor coding method (HE5\_HDFE\_COMP\_SHUF\_SZIP\_K13orNN = 17), and no compression (HE5\_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.

Szip compression requires one parameter that is a `pixels_per_block` which must be even, with typical values being 8, 10, 16, 32. The more pixel values vary, the smaller this number should be. Compressed fields are written using the standard `HE5_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 `HE5_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* field using the entropy coding method, the *Opacity* field using the shuffling + deflate method, the *Spectra* field with deflate compression, and use no compression for the *Temperature* field.

```
compparm[0] = 16;

status = HE5_SWdefcomp(swathID, HE5_HDFE_COMP_SZIP_EC,
compparm);

status = HE5_SWdefdatafield(swathID, "Pressure",
"Track,Xtrack", NULL, H5T_NATIVE_FLOAT, 0);

compparm[0] = 9;

status = HE5_SWdefcomp(swathID, HE5_HDFE_COMP_SHUF_DEFLATE,
compparm);

status = HE5_SWdefdatafield(swathID, "Opacity",
"Track,Xtrack", NULL, H5T_NATIVE_FLOAT, 0);

status = HE5_SWdefcomp(swathID, HE5_HDFE_COMP_DEFLATE,
compparm);

status = HE5_SWdefdatafield(swathID, "Spectra",
"Bands,Track,Xtrack", NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);

status = HE5_SWdefcomp(swathID, HE5_HDFE_COMP_NONE,
compparm);

status = HE5_SWdefdatafield(swathID, "Temperature",
"Track,Xtrack", NULL, H5T_NATIVE_FLOAT, 0);
```

**Note that the HE5\_HDFE\_AUTOMERGE/MERGE parameter is ignored in the *Temperature* field definition.**

FORTRAN integer function `he5_swdefcomp(swathid, compcode, compparm)`



```
integer    swathid
integer    compcode
integer    compparm(*)
```

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

```
parameter (HE5_HDFE_NATIVE_FLOAT=1)
parameter (HE5_HDFE_COMP_NONE=0)
parameter (HE5_HDFE_COMP_DEFLATE=4)
parameter (HE5_HDFE_COMP_SZIP_EC=7)
parameter (HE5_HDFE_COMP_SHUF_DEFLATE=11)
integer    compparm(5)
compparm(1) = 16

status = he5_swdefcomp(swathid, HE5_HDFE_COMP_SZIP_EC,
compparm);

status = he5_swdefdfld(swathid, "Pressure", "Xtrack,Track",
" ", HE5_HDFE_NATIVE_FLOAT, 0);

compparm(1) = 9

status = he5_swdefcomp(swathid, HE5_HDFE_COMP_SHUF_DEFLATE,
compparm);

status = he5_swdefdfld(swathid, "Opacity", "Xtrack,Track", "
", HE5_HDFE_NATIVE_FLOAT, 0);

status = he5_swdefcomp(swathid, HE5_HDFE_COMP_DEFLATE,
compparm);

status = he5_swdefdfld(swathid, "Spectra", "Xtrack,
Track,Bands", " ", HE5_HDFE_NATIVE_FLOAT, 0)

status = he5_swdefcomp(swathid, HE5_HDFE_COMP_NONE,
compparm)

status = he5_swdefdfld(swathid, "Temperature",
"Xtrack,Track", " ", HE5_HDFE_NATIVE_FLOAT, 0)
```

# Define a New Data Field within a Swath

---

## HE5\_SWdefdatafield

herr\_t HE5\_SWdefdatafield(hid\_t *swathID*, const char \**fieldname*, char \**dimlist*, char \**maxdimlist*, hid\_t *ntype*, int *merge*)

<i>swathID</i>	IN:	Swath ID returned by HE5_SWcreate or HE5_SWattach
<i>fieldname</i>	IN:	Name of field to be defined
<i>dimlist</i>	IN:	The list of data dimensions defining the field
<i>maxdimlist</i>	IN:	The list of maximum data dimensions defining the field
<i>ntype</i>	IN:	The number type of the data stored in the field
<i>merge</i>	IN:	Merge code (HE5_HDFE_NOMERGE (0) - no merge, HE5_HDFE_AUTOMERGE (1) -merge)

**Note: Merging is not supported in this release of the library. There are three illegal characters for field names: “/”, “;”, “,”**

Purpose	Defines a new data field within the swath.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list.
Description	This routine defines data 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 <i>C</i> 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.

**Note: One should define both chunking and compression before HE5\_SWdefdatafield() if field is supposed to be compressed.**

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

```
status = HE5_SWdefdatafield(swathID, "Spectra",  
                             "Bands,DataTrack,DataXtrack", " ", H5T_NATIVE_FLOAT,  
                             0);
```

FORTRAN integer function he5\_swdefdfld(*swathid*, *fieldname*, *dimlist*, *maxdimlist*, *ntype*, *merge*)

integer        *swathid*  
character\*(\*) *fieldname*  
character\*(\*) *dimlist*  
character\*(\*) *maxdimlist*  
integer        *ntype*  
integer        *merge*

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

```
parameter (HE5T_NATIVE_FLOAT=10)  
parameter (HE5_HDFE_NOERGE=0)  
status = he5_swdefdfld(swathid, "Spectra", "DataXtrack,  
DataTrack, Bands", " ", HE5T_NATIVE_FLOAT, 0)
```

## Define a New Dimension within a Swath

---

### HE5\_SWdefdim

herr\_t HE5\_SWdefdim(hid\_t *swathID*, char \**dimname*, hsize\_t *dim*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*dimname* IN: Name of dimension to be defined

*dim* IN: The size of the dimension

**Note: There are three illegal characters for dimension names: “/”, “;”, “.”**

Purpose Defines a new dimension within the swath.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is an improper swath ID.

Description This routine defines dimensions that are used by the field definition routines (described subsequently) to establish the size of the field.

Example In this example, we define a track geolocation dimension, *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 = HE5_SWdefdim(swathID, "GeoTrack", 2000);
status = HE5_SWdefdim(swathID, "GeoXtrack", 1000);
status = HE5_SWdefdim(swathID, "DataTrack", 4000);
status = HE5_SWdefdim(swathID, "DataXtrack", 2000);
status = HE5_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 -1 or equivalently, *H5S\_UNLIMITED*:

```
status = HE5_SWdefdim(swathID, "Unlim", H5S_UNLIMITED);
```

FORTRAN integer function he5\_swdefdim(*swathid*,*dimname*,*dim*)

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

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

```
dim      = 2000
status = he5_swdefdim(swathid, "GeoTrack", dim)
```

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

```
parameter (HE5S_UNLIMITED_F=-1)  
status = he5_swdefdim(swathid, "Unlim", HE5S_UNLIMITED_F)
```

# Define Mapping between Geolocation and Data Dimensions

---

## HE5\_SWdefdimmap

herr\_t HE5\_SWdefdimmap(hid\_t *swathID*, char \**geodim*, char \**datadim*, hsize\_t *offset*, hsize\_t *increment*)

<i>swathID</i>	IN:	Swath ID returned by HE5_SWcreate or HE5_SWattach
<i>geodim</i>	IN:	Geolocation dimension name
<i>datadim</i>	IN:	Data dimension name
<i>offset</i>	IN:	The offset of the geolocation dimension with respect to the data dimension
<i>increment</i>	IN:	The increment of the geolocation dimension with respect to the data dimension

**Purpose** Defines monotonic mapping between the geolocation and data dimensions.

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

**Description** Typically the geolocation and data dimensions are of different size (resolution). This routine established the relation between the two where the offset gives the index of the data element (0-based) corresponding to the first geolocation element and the increment gives the number of data elements to skip for each geolocation element. If the geolocation dimension begins "before" the data dimension, then the offset is negative. Similarly, if the geolocation dimension has higher resolution than the data dimension, then the increment is negative.

**Example** In this example, we establish that (1) the first element of the *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 = HE5_SWdefdimmap(swathID, "GeoTrack", "DataTrack",  
0, 2);
```

```
status = HE5_SWdefdimmap(swathID, "GeoXtrack", "DataXtrack",  
1, 2);
```

```
FORTRAN integer function he5_swdefmap(swathid,geodim,datadim,offset,increment)
integer      swathid
character*(*) geodim
character*(*) datadim
integer*4    offset
integer*4    increment
```

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

```
offset      = 0
increment   = 2
status = he5_swdefmap(swathid, "GeoTrack", "DataTrack",
offset, increment)
offset      = 1
increment   = 2
status=he5_swdefmap(swathid, "GeoXtrack", "DataXtrack", offset,
increment)
```

# Define a New Geolocation Field within a Swath

---

## HE5\_SWdefgeofield

herr\_t HE5\_SWdefgeofield(hid\_t *swathID*, const char \**fieldname*, char \**dimlist*, char \**maxdimlist*, hid\_t *ntype*, int *merge*)

<i>swathID</i>	IN:	Swath ID returned by HE5_SWcreate or HE5_SWattach
<i>fieldname</i>	IN:	Name of field to be defined
<i>dimlist</i>	IN:	The list of geolocation dimensions defining the field
<i>maxdimlist</i>	IN:	The maximum dimension list of geolocation dimensions defining the field
<i>ntype</i>	IN:	The number type of the data stored in the field
<i>merge</i>	IN:	Merge code (HE5_HDFE_NOMERGE (0) - no merge, HE5_HDFE_AUTOMERGE(1) - merge)

**Note: Merging is not supported in this release of the library. There are three illegal characters for field names: “/”, “;”, “,”**

Purpose	Defines a new geolocation field within the swath.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list.
Description	This routine defines geolocation fields to be stored in the swath. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in <i>C</i> 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 <i>C</i> 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.
Example	In this example, we define the geolocation fields, <i>Longitude</i> and <i>Latitude</i> with dimensions <i>GeoTrack</i> and <i>GeoXtrack</i> and containing 4 byte floating point numbers. We allow these fields to be merged into a single object:

```
status = HE5_SWdefgeofield(swathID, "Longitude",  
                           "GeoTrack,GeoXtrack", NULL, H5T_NATIVE_FLOAT, 0);
```



```

status = HE5_SWdefgeofield(swathID, "Latitude",
    "GeoTrack,GeoXtrack", NULL, H5T_NATIVE_FLOAT,
HE5_HDFE_NOMERGE);

```

FORTRAN integer function *he5\_swdefgfld*(*swathid*, *fieldname*, *dimlist*, *maxdimlist*, *ntype*, *merge*)

```

integer      swathid
character*(*) fieldname
character*(*) dimlist
character*(*) maxdimlist
integer      ntype
integer      merge

```

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

```

parameter (HE5T_NATIVE_FLOAT=10)
parameter (HE5_HDFE_NOMERGE=0)

status=he5_swdefgfld(swathid, "Longitude", "GeoXtrack,GeoTrack",
    " ", HE5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE)

```

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

# Define Indexed Mapping between Geolocation and Data Dimension

---

## HE5\_SWdefidxmap

herr\_t HE5\_SWdefidxmap(hid\_t *swathID*, char \**geodim*, char \**datadim*, long *index*[]),

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_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 or FAIL (-1) otherwise. Typical reason for failure is incorrect geolocation or data dimension name.

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

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

```
long index[5] = {0, 2, 3, 6, 7};  
status = HE5_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.

FORTTRAN integer function he5\_swdefimap(*swathid*, *geodim*, *datadim*, *index*)

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

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

```
status = he5_swdefimap(swathid, "IdxGeo", "IdxData", index)
```

## Define a Time Period of Interest

---

### HE5\_SWdeftimeperiod

hid\_t HE5\_SWdeftimeperiod(hid\_t *swathID*, double *starttime* , double *stoptime*, int *mode*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_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 or FAIL (-1) otherwise.

Description This routine defines a time period for a swath. It returns a swath period ID which is used by the *HE5\_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 = HE5_SWdeftimeperiod(swathID, starttime, stoptime,  
HE5_HDFE_ENDPOINT);
```

FORTRAN integer function he5\_swdeftmeper(*swathid*, *starttime*, *stoptime*, *mode*)  
integer *swathid*  
real\*8 *starttime*  
real\*8 *stoptime*  
integer *mode*

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

```
parameter (HE5_HDFE_ENDPOINT=1)
starttime = 35232487.2
stoptime = 36609898.1
periodID = he5_swdeftmeper(swathID, starttime, stoptime,
HE5_HDFE_ENDPOINT)
```

## Define a Vertical Subset Region

---

### HE5\_SWdefvrtregion

hid\_t HE5\_SWdefvrtregion(hid\_t *swathID*, hid\_t *regionID*, char \**vertObj*, double *range*[])

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_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 or FAIL (-1) otherwise.

Description Whereas the *HE5\_SWdefboxregion* and *HE5\_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: INT, LONG, FLOAT, DOUBLE.) This routine may be called after *HE5\_SWdefboxregion* or *HE5\_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 region ID to HE5\_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 *HE5\_SWregioninfo* and *HE5\_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 *HE5\_SWregioninfo* and *HE5\_SWextractregion*) must contain the dimension used explicitly in the call to *HE5\_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 = HE5_SWdefvrtregion(swathID, HE5_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 = HE5_SWdefvrtregion(swathID, HE5_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 = HE5_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 = HE5_SWdefvrtregion(swathID, regionID,
"FreqRange", freq);
```

FORTTRAN integer function `he5_swdefvrtreg(swathid, regionid, vertobj, range)`  
integer *swathid*  
integer *regionid*  
character\*(\*) *vertobj*  
real\*8 *range*(\*)

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

```
parameter (HE5_HDFE_NOPREVSUB=-1)  
range(1) = 500.  
range(2) = 800.  
regionid = he5_swdefvrtreg(swathid, HE5_HDFE_NOPREVSUB,  
"Pressure", range)
```

## Detach from a Swath Structure

---

### HE5\_SWdetach

herr\_t HE5\_SWdetach(hid\_t *swathID*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

Purpose Detaches from swath interface.

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

Description This routine should be run before exiting from the swath file for every swath opened by *HE5\_SWcreate* or *HE5\_SWattach*.

Example In this example, we detach the swath structure, *ExampleSwath*:

```
status = HE5_SWdetach(swathID);
```

FORTTRAN integer function he5\_swdetach(swathid)

```
integer      swathid
```

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

```
status = he5_swdetach(swathid)
```



## Retrieve Size of Specified Dimension

---

### HE5\_SWdiminfo

hsize\_t HE5\_SWdiminfo(hid\_t *swathID*, char \**dimname*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*dimname* IN: Dimension name

Purpose Retrieve size of a specified dimension.

Return value Size of dimension if successful or 0 otherwise. A typical reason for failure is an improper grid id or dimension name.

Description This routine retrieves the size of specified dimension.

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

```
dimsize = HE5_SWdiminfo(swathID, "GeoTrack");
```

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

FORTTRAN integer\*4 function he5\_swdiminfo(*swathid*,*dimname*)

```
integer      swathid
```

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

```
integer*4    dimsize
```

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

```
dimsize = he5_swdiminfo(swathid, "GeoTrack")
```

# Remove an Alias for Swath Data Field

---

## HE5\_SWdropalias

herr\_t HE5\_SWdropalias(hid\_t *swathID*, int *fldgroup*, const char \**aliasname*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fldgroup* IN: Field group flag

*aliasname* IN: Name of alias to remove

Purpose Remove an alias for Swath data field

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

Description Removes alias associated with a Swath data field.

Example In this example, we remove an alias for the data field *Temperature*.

```
strcpy(aliasname, "temps 0 to 30");
```

```
status = HE5_SWdropalias(swathID, HE5_HDFE_DATAGROUP,  
aliasname);
```

FORTRAN integer function he5\_swdropalias (*swathid*, *fldgroup*, *aliasname*)  
integer *swathid*  
character\*(\*) *fldgroup*  
character\*(\*) *aliasname*

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

```
aliasname = "temps 0 to 30"
```

```
status = he5_swdropalias(swathid, HE5_HDFE_DATAGROUP,  
aliasname)
```

# Duplicate a Region or Period

---

## HE5\_SWdupregion

hid\_t HE5\_SWdupregion(hid\_t *regionID*)

*regionID* IN: Region or period ID returned by HE5\_SWdefboxregion, HE5\_SWdeftimeperiod, or HE5\_SWdefvrtregion.

Purpose Duplicates a region.

Return value Returns new region or period ID if successful or FAIL (-1) otherwise.

Description This routine copies the information stored in a current region or period to a new region or period and generates a new id. It is usefully when the user wishes to further subset a region (period) in multiple ways.

Example In this example, we first subset a swath with *HE5\_SWdefboxregion*, duplicate the region creating a new region ID, *regionID2*, and then perform two different vertical subsets of these (identical) geographic subset regions:

```
regionID = HE5_SWdefboxregion(swathID, cornerlon, cornerlat,  
                             HE5_HDFE_MIDPOINT);
```

```
regionID2 = HE5_SWdupregion(regionID);
```

```
regionID = HE5_SWdefvrtregion(swathID, regionID, "Pressure",  
                             rangePres);
```

```
regionID2 = HE5_SWdefvrtregion(swathID, regionID2,  
                              "Temperature", rangeTemp);
```

FORTRAN integer he5\_swdupreg(*regionid*)

integer *regionid*

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

```
parameter (HE5_HDFE_MIDPOINT=0)
```

```
regionid = he5_swdefboxreg(swathid, cornerlon, cornerlat,  
                          HE5_HDFE_MIDPOINT)
```

```
regionid2 = he5_swdupreg(regionid)
```

```
regionid = he5_swdefvrtreg(swathid, regionid, 'Pressure',  
                          rangePres)
```

```
regionid2 = he5_swdefvrtreg(swathid, regionid2,  
                          'Temperature', rangeTemp)
```

# Read Data from a Defined Time Period

---

## HE5\_SWextractperiod

herr\_t HE5\_SWextractperiod(hid\_t *swathID*, hid\_t *periodID*, char \**fieldname*, int *externalflag*, void \**buffer*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*periodID* IN: Period id returned by HE5\_SWdeftimeperiod

*fieldname* IN: Field to subset

*externalflag* IN: External geolocation mode

*buffer* OUT: Data buffer

Purpose Extracts (reads) from subsetted time period.

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

**NOTE: External file functionality not available in this release**

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 *HE5\_HDFE\_EXTERNAL* (1) then the geolocation fields and the data field can be in different swaths. If set to *HE5\_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 *HE5\_SWdeftimeperiod* from the *Spectra* field. Both the geolocation fields and the *Spectra* data field are in the same swath.

```
status = HE5_SWextractperiod(SwathID, periodID, "Spectra",  
                             HE5_HDFE_INTERNAL, datbuf);
```

FORTRAN integer function he5\_swextper(*swathID*, *periodid*, *fieldname*, *externalflag*, *buffer*)

integer *swathID*

integer *periodid*

character\*(\*) *fieldname*

integer *externalflag*

<valid type> *buffer*(\*)

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

```
parameter (HE5_HDFE_INTERNAL=0)

status = he5_swextper(swathid, periodid, "Spectra",
HE5_HDFE_INTERNAL, datbuf)
```

# Read Data from a Geographic Region

---

## HE5\_SWextractregion

herr\_t HE5\_SWextractregion(hid\_t *swathID*, hid\_t *regionID*, char \**fieldname*, int *externalflag*, void \**buffer*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*regionID* IN: Region ID returned by HE5\_SWdefboxregion

*fieldname* IN: Field to subset

*externalflag* IN: External geolocation mode

*buffer* OUT: Data buffer

Purpose Extracts (reads) from subsetted region.

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

**NOTE: External file functionality not available in this release**

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 *HE5\_HDFE\_EXTERNAL* (1) then the geolocation fields and the data field can be in different swaths. If set to *HE5\_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 *HE5\_SWdefboxregion* from the *Spectra* field. Both the geolocation fields and the *Spectra* data field are in the same swath.

```
status = HE5_SWextractregion(Swid, regionID, "Spectra",  
HE5_HDFE_INTERNAL, datbuf);
```

FORTRAN integer function he5\_swextreg(*swathid*, *regionid*, *fieldname*, *externalflag*,  
*buffer*)

```
integer      swathid  
integer      regionid  
integer      externalflag  
character(*) fieldname  
<valid type> buffer(*)
```

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

```
parameter (HE5_HDFE_INTERNAL=0)  
status = he5_swextreg(swathid, regionid, "Spectra",  
HE5_HDFE_INTERNAL, datbuf)
```

# Retrieve Information about a Swath Field

---

## HE5\_SWfieldinfo

herr\_t HE5\_SWfieldinfo(hid\_t *swathID*, char \**fieldname*, int \**rank*, hsize\_t *dims*[], hid\_t *ntype*[], char \**dimlist*, char \**maxdimlist*)

<i>swathID</i>	IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
<i>fieldname</i>	IN: Fieldname
<i>rank</i>	OUT: Rank of field
<i>dims</i>	OUT: Array containing the dimension sizes of the field
<i>ntype</i>	OUT: Array containing the numbertype of the field. See Appendix A for interpretation of number types.
<i>dimlist</i>	OUT: List of dimensions in field
<i>maxdimlist</i>	OUT: List of maximum dimensions in field

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

**Return value** Returns SUCCEED (0) if successful or FAIL (-1) otherwise. A typical reason for failure is the specified field does not exist.

**Description** This routine retrieves information on a specific data field.

**Example** In this example, we retrieve information about the *Spectra* data fields:

```
status = HE5_SWfieldinfo(swathID, "Spectra", &rank, dims,
                        numbertype, dimlist, maxdimlist);
```

The return parameters will have the following values:

*rank=3, numbertype=10, 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 `he5_swfldinfo(swathid, fieldname, rank, dims, ntype, dimlist, maxdimlist)`

integer *swathid*

character\*(\*) *fieldname*

integer *rank*

integer\*4 *dims*(\*)

integer *ntype*(\*)

character\*(\*) *dimlist*

character\*(\*) *maxdimlist*

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

```
status = he5_swfldinfo(swathid, "Spectra", rank, dims,  
numbertype, dimlist, maxdimlist)
```

The return parameters will have the following values:

*rank*=3, *numbertype*=10, *dims*[3]={2000,4000,5} and  
*dimlist*="DataXtrack, DataTrack,Bands"

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



# Rename Swath Data Field

---

## HE5\_SWfldrename

herr\_t HE5\_SWfldrename(hid\_t *swathID*, char \**oldfieldname*, const char \**newfieldname*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*oldfieldname* IN: Current name of field

*newfieldname* IN: New name of field

Purpose Rename swath data field

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

Description This function allows the user to change the name of a field. This is useful in case the user would want to update the data field to reflect a version change in the calibration of a data field and show that in the name of the field.

Example In this example, we give a new name for the data field *Temperature*.

```
strcpy(newfieldname, "temps 0 to 30");
```

```
status = HE5_SWfldrename(swathID, "Temperature",  
newfieldname);
```

FORTRAN integer function he5\_swfldrename (*swathid*, *oldfieldname*, *newfieldname*)  
integer *swathid*  
character\*(\*) *oldfieldname*  
character\*(\*) *newfieldname*

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

```
newfieldname = "temps 0 to 30"
```

```
status = he5_swfldrename(swathid, "Temperature",  
newfieldname)
```

# Return Information about a Swath Attribute in Group “Geolocation Fields”

---

## HE5\_SWgeogrpatrinfo

herr\_t HE5\_SWgeogrpatrinfo(hid\_t *swathID*, const char \**attrname*, hid\_t \**ntype*,  
hsize\_t \**count*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*attrname* IN: Attribute name  
*numbertype* OUT: Number type of attribute. See Appendix A for interpretation of number types.  
*count* OUT: Number of attribute elements

Purpose Returns information about a group attribute in “Geolocation Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description This routine returns number type and number of elements (count) of a swath attribute in “Geolocation Fields” group.

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

```
status = HE5_SWgeogrpatrinfo(swathID, "ScalarFloat", &nt,  
&count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

FORTRAN integer function he5\_swgeogattrinfo(*swathid*, *attrname*, *ntype*, *count*,)  
integer *swathid*  
character\*(\*) *attrname*  
integer *ntype*  
integer \*4 *count*

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

```
status = he5_swgeogattrinfo(swathid, "ScalarFloat",nt,count)
```

# Retrieve Type of Dimension Mapping when First Dimension is Geodim

---

## HE5\_SWgeomapinfo

herr\_t HE5\_SWgeomapinfo(hid\_t *swathID*, char \**geodim*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*geodim* IN: Dimension name

Purpose Retrieve type of dimension mapping for a dimension.

Return value Returns (2) for indexed mapping, (1) for regular mapping, (0) if dimension is not mapped, or FAIL (-1) otherwise.

Description This routine checks the type of mapping (regular or indexed).

Example In this example, we retrieve information about the type of mapping between the “IdxGeo” and “IdxData” dimensions, defined by *HE5\_SWdefidxmap*.

```
status = HE5_SWgeomapinfo(swathID, geodim);
```

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

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

FORTTRAN integer function he5\_swgmapinfo(*swathid*,*geodim*)

integer *swathid*

character\*(\*) *geodim*

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

```
status = he5_swgmapinfo(swathid, geodim)
```

# Retrieve Alias List for a Swath Geo/Data Field Groups

---

## HE5\_SWgetaliaslist

long HE5\_SWgetaliaslist(hid\_t *swathID*, int *fldgroup*, char \**aliaslist*, long \**strbufsize*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fldgroup* IN: Field group flag (geo or data)

*aliaslist* OUT: List of alias(es) in the “Data Fields” or “Geo Fields” group (comma separated list)

*strbufsize* OUT: Length of aliases list

Purpose To retrieve the number and list of aliases in a swath

Return value Returns number of aliases in "Data Fields" or “Geo Fields” groups if successful or FAIL (-1) otherwise.

Description Retrieves list of aliases in the “Data Fields” or “Geo Fields” group (comma separated list) of a Swath and returns their number. The Geo and Data group flags are **HE5\_HDFE\_GEOGROUP** and **HE5\_HDFE\_DATAGROUP**, respectively.

Example In this example, we get the alias list for the “data fields” group .

```
/* first get the size of the list in bytes */
```

```
nalias = HE5_SWgetaliaslist(swathID, HE5_HDFE_DATAGROUP,  
NULL, strbufsize);
```

```
aliaslist = (char *)malloc(strbufsize *sizeof(char));
```

```
nalias = HE5_SWgetaliaslist(swathID, HE5_HDFE_DATAGROUP,  
aliaslist, strbufsize);
```

FORTRAN integer function he5\_swgetaliaslist (*swathid*, *fldgroup*, *aliaslist*, *strbufsize*)  
integer *swathid*  
integer *fldgroup*  
integer *strbufsize*  
character\*(\*) *aliaslist*

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

```
integer nalias
```

```
nalias = he5_swgetaliaslist(swathid, HE5_HDFE_DATAGROUP,  
aliaslist, strbufsize)
```

# Get External Data File Information

---

## HE5\_SWgetextdata

int HE5\_SWgetextdata(hid\_t *swathID*, char \**fieldname*, size\_t *namelength*, char \**filelist*, off\_t *offset[]*, hsize\_t *size[]*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*fieldname* IN: External field name  
*namelength* OUT: Length of each name entry  
*filelist* OUT: List of file names  
*offset[]* OUT: Array of offsets (in byte) from the beginning of file to the location in file where the data starts  
*size[]* OUT: Array of sizes (in bytes) reserved in the file for the data

Purpose Retrieves information about external data file(s) associated with the data set.

Return value Returns number of external data files if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath ID or field name.

Example In this example, we get information about the *ExtData* field:

```
nfiles = HE5_SWgetextdata(swathID, "ExtData", namlen,  
filenames, offset, size);
```

FORTTRAN integer function he5\_swgetxdat(*swathid*,*fieldname*,*nlen*,*flist*,*offset*,*size*)

```
integer      swathid  
integer      nfiles  
integer*4    nlen  
integer*4    offset(*)  
integer*4    size(*)  
character*(*) fieldname  
character*(*) flist
```

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

```
nfiles = he5_swgetxdat(swathid, "ExtData",nlen,  
flist,offset,size)
```

# Get Fill Value for a Specified Field

---

## HE5\_SWgetfillvalue

herr\_t HE5\_SWgetfillvalue(hid\_t *swathID*, char \**fieldname*, void \**fillval*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fieldname* IN: Fieldname

*fillval* OUT: Space allocated to store the fill value

Purpose Retrieves fill value for the specified field.

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

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

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

```
status = HE5_SWgetfillvalue(swathID, "Temperature",
&tempfill);
```

FORTTRAN integer function he5\_swgetfill(*swathid*,*fieldname*,*fillval*)

integer *swathid*

character\*(\*) *fieldname*

<valid type> *fillval*(\*)

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

```
status = he5_swgetfill(swathid, "Temperature", tempfill)
```

# Return Information about a Swath Attribute in Group “Data Fields”

---

## HE5\_SWgrpattrinfo

herr\_t HE5\_SWgrpattrinfo(hid\_t *swathID*, const char \**attrname*, hid\_t \**ntype*,  
hsize\_t \**count*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*attrname* IN: Attribute name  
*numbertype* OUT: Number type of attribute. See Appendix A for interpretation of number types.  
*count* OUT: Number of attribute elements  
Purpose Returns information about a group attribute in “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).  
Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.  
Description This routine returns number type and number of elements (count) of a swath attribute in “Data Fields” group.

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

```
status = HE5_SWgrpattrinfo(swathID, "ScalarFloat", &nt,  
&count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

FORTRAN integer function he5\_swgattrinfo(*swathid*, *attrname*, *ntype*, *count*,)  
integer *swathid*  
character\*(\*) *attrname*  
integer *ntype*  
integer \*4 *count*

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

```
status = he5_swgattrinfo(swathid, "ScalarFloat",nt,count)
```

# Retrieve Indexed Geolocation Mapping

---

## HE5\_SWidxmapinfo

hsize\_t HE5\_SWidxmapinfo(hid\_t *swathID*, char \**geodim*, char \**datadim*, long *index*[])

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*geodim* IN: Indexed Geolocation dimension name

*datadim* IN: Indexed Data dimension name

*index* OUT: Index mapping array

Purpose Retrieve indexed array of specified geolocation mapping.

Return value Returns size of indexed array if successful or 0 otherwise. A typical reason for failure is the specified mapping does not exist.

Description This routine retrieves the size of the indexed array and the array of indexed elements of the specified geolocation mapping.

Example In this example, we retrieve information about the indexed mapping between the "*IdxGeo*" and "*IdxData*" dimensions:

```
idxsz = HE5_SWidxmapinfo(swathID, "IdxGeo", "IdxData",  
index);
```

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

FORTRAN integer\*4 function he5\_swimapinfo(*swathid*, *geodim*, *datadim*, *index*)

integer *swathid*

character\*(\*) *geodim*

character\*(\*) *datadim*

integer\*4 *index*(\*)

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

```
idxsz = he5_swimapinfo(swathid, "IdxGeo", "IdxData", index)
```



# Retrieve the Indices of a Subsetted Region

---

## HE5\_SWindexinfo

herr\_t HE5\_SWindexinfo(hid\_t *regionID*, char \**object*, int \**rank*, char \**dimlist*, hsize\_t \**indices*[HE5\_DTSETRANKMAX])

*regionID* IN: Region ID returned by HE5\_SWdefboxregion and/or HE5\_SWdefvrtregion

*object* IN: Field name

*rank* OUT: Rank of field

*dimlist* OUT: List of dimensions in field

*indices* OUT: The array (0-based) containing the indices for start and stop of region

Purpose Retrieve the indices information about a subsetted region.

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

Description This routine returns the indices information about a subsetted region for a particular field. It retrieves the indices for start and stop of region.

Example In this example, we retrieve the indices information about the *Longitude* field defined by HE5\_SWdefboxregion:

```
status = HE5_SWindexinfo(regionID, "Longitude", &rank,  
                        dimlist, indices);
```

The return parameters will have the following values:

*Rank*=2, *dimlist*="DataTrack, DataXtrack", and *indices*[0][0]=4,

*indices*[0][1]=11, *indices*[1][0]=0, *indices*[1][1]=10

FORTTRAN integer function he5\_swindexinfo(*regionid*, *object*, *rank*, *dimlist*, *indices*)

integer *regionid*

character\*(\*) *object*

integer *rank*

character\*(\*) *dimlist*

integer\*4     *indices*

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

```
status = he5_swindeinfo(regionid, "Longitude", rank,  
dimlist, indices)
```

The return parameters will have the following values:

*rank=2, dimlist="DataXtrack,DataTrack", and indices(1,1)=0,  
indices(1,2)=10, indices(2,1)=4, indices(2,2)=11*

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

# Retrieve Information Swath Attributes

---

## HE5\_SWinqattrs

long HE5\_SWinqattrs(hid\_t *swathID*, char \**attrnames*, long \**strbufsize*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrnames* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

Purpose Retrieve information about object attributes defined in a specific swath. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each attribute name separated by commas. If *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. In the first call, set the parameter *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_SWinqattrs(swathID, NULL, &strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_SWinqattrs(swathID, attrnames, &strbufsize);
```

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

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

FORTRAN integer\*4 function he5\_swinqattrs(*swathid*,*attrnames*,*strbufsize*)

integer *swathid*

character\*(\*) *attrnames*

integer\*4 *strbufsize*

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

```
nattr = he5_swinqattrs(swathid, attrnames, strbufsize)
```

# Retrieve Information about Data Fields Defined in Swath

---

## HE5\_SWinqdatafields

long HE5\_SWinqdatafields(hid\_t *swathID*, char \**fieldlist*, int *rank*[], hid\_t *ntype*[])

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*fieldlist* OUT: Listing of data fields (entries separated by commas)  
*rank* OUT: Array containing the rank of each data field  
*ntype* OUT: Array containing the numbertype of each data field. See Appendix A for interpretation of number types.

Purpose Retrieve information about all of the data fields defined in swath.

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

Description The field list is returned as a string with each data field separated by commas. The *rank* and *ntype* 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 = HE5_SWinqdatafields(swathID, fieldlist, rank,
                             ntype);
```

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

"Spectra" with *rank*[1]={3}, *ntype*[1]={10}

FORTTRAN integer\*4 function he5\_swinqdflds(*swathid*, *fieldlist*, *rank*, *ntype*)

integer *swathid*

character\*(\*) *fieldlist*

integer *rank*(\*)

integer *ntype*(\*)

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

```
nflds = he5_swinqdflds(swathid, fieldlist, rank, ntype)
```

# Return Data Type Information about Data Fields in Swath

---

## HE5\_SWinqdatatype

```
herr_t HE5_SWinqdatatype(hid_t swathID, const char *fieldname, const char
                        *attrname, int fieldgroup, hid_t *datatype, H5T_class_t
                        *classid, H5T_order_t *order, size_t *size)
```

<i>swathID</i>	IN:	Swath ID returned by HE5_SWcreate or HE5_SWattach
<i>fieldname</i>	IN:	Field name
<i>attrname</i>	IN:	Attribute name
<i>fieldgroup</i>	IN:	Field group flag: HE5_HDFE_GEOGROUP - 0 HE5_HDFE_DATAGROUP - 1 HE5_HDFE_ATTRGROUP - 2 HE5_HDFE_GRPATTRGROUP - 3 HE5_HDFE_LOCATTRGROUP - 4 HE5_HDFE_PROFGROUP - 5 HE5_HDFE_PROFGRPATTRGROUP - 6 HE5_HDFE_GEOGRPATTRGROUP - 7
<i>datatype</i>	OUT:	Data type ID
<i>classID</i>	OUT:	Data type class ID
<i>order</i>	OUT:	Data type byte order
<i>size</i>	OUT:	Data type size (in bytes)
Purpose		Returns data type information about a specified field in swath.
Return value		Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath ID or field name.
Description		This routine returns information about field data in a swath.
Example		In this example we return the data type information for the <i>Spectra</i> field in the swath defined in the <i>HE5_SWdefdatafield</i> routine.  <pre>status = HE5_SWinqdatatype(swathID, "Spectra", NULL,                           fieldgroup, &amp;datatype, &amp;classid, &amp;order, &amp;size);</pre>

FORTRAN integer function  
 he5\_swidtype(*swathid,fieldname,attrname,fldgrp,dtype,classid,order, size*)

integer *swathid*

integer *dtype,classid,order*

integer\*4 *size*

character \*(\*) *fieldname*

integer *HE5\_HDFE\_DATAGROUP*

parameter (*HE5\_HDFE\_DATAGROUP=1*)

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

```
status = he5_swidtype(swathid, "Spectra", " ",
HE5_HDFE_DATAGROUP, dtype, classid, order, size)
```

# Retrieve Information about Data Fields and Aliases Defined in Swath

---

## HE5\_SWinqdfldalias

long HE5\_SWinqdfldalias(hid\_t *swathID*, char \**fldalias*, long \**strbufsize*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*fldalias* OUT: List of data fields and aliases (entries separated by commas)  
*strbufsize* OUT: String length of data fields and aliases list

Purpose Retrieve information about data fields & aliases defined in swath.

Return value Number of data fields and aliases found if successful or FAIL (-1) otherwise.

Description The list of data fields and aliases is returned as a string with each name separated by commas. If *fldalias* 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 data fields and aliases defined for the “Data Fields” group. In the first call, set the parameter *fldalias* to NULL. We assume that there are one data field and one alias stored, *Temperature* and *Temp*:

```
nfldalias = HE5_SWinqdfldalias(swathID, NULL, &strbufsize);
```

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

```
fldalias = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nfldalias = HE5_SWinqdfldalias(swathID, fldalias,  
&strbufsize);
```

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

*"Temperature,Temp"*.

FORTRAN integer\*4 function he5\_swinqdfldalias(*swathid*, *fldalias*, *strbufsize*)

integer *swathid*

character\*(\*) *fldalias*

integer\*4 *strbufsize*

integer\*4 *nfldalias*

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

```
nfldalias = he5_swingdfldalias(swathid, fldalias,  
strbufsize)
```



# Retrieve Information about Dimensions Defined in Swath

---

## HE5\_SWinqdims

long HE5\_SWinqdims(hid\_t *swathID*, char \**dimnames*, hsize\_t *dims*[])

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*dimnames* 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 successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id.

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

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

```
ndims = HE5_SWinqdims(swathID, dimnames, dims);
```

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

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

```
with ndims = 6, dims[6]={2000,1000,4000,2000,5,-1}
```

FORTRAN integer\*4 function he5\_swinqdims(*swathid*,*dimnames*,*dims*)

integer *swathid*

character\*(\*) *dimnames*

integer\*4 *dims*(\*)

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

```
ndims = he5_swinqdims(swathid, dimnames, dims)
```

# Retrieve Information about Geolocation Fields Defined in Swath

---

## HE5\_SWinqgeofields

long HE5\_SWinqgeofields(hid\_t *swathID*, char \**fieldlist*, int *rank*[], hid\_t *ntype*[])

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*fieldlist* OUT: Listing of geolocation fields (entries separated by commas)  
*rank* OUT: Array containing the rank of each geolocation field  
*ntype* OUT: Array containing the numbertype of each geolocation field. See Appendix A for interpretation of number types.  
Purpose Retrieve information about all of the geolocation fields defined in swath.  
Return value Number of geolocation fields found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id.  
Description The field list is returned as a string with each geolocation field separated by commas. The *rank* and *ntype* 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 = HE5_SWinqgeofields(swathID, fieldlist, rank,  
                           ntype);
```

The parameter, *fieldlist*, will have the value: "Longitude,Latitude" with *nflds* = 2, *rank*[2]={2,2}, *ntype*[2]={10,10}

FORTTRAN integer\*4 function he5\_swinqgflds(swathid, fieldlist, rank, ntype)

integer *swathid*

character\*(\*) *fieldlist*

integer *rank*(\*)

integer *ntype*(\*)

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

```
nflds = he5_swinqgflds(swathid, fieldlist, rank, ntype)
```

# Retrieve Information about Group “Geolocation Fields” Attributes

---

## HE5\_SWinqgeogrpattrs

long HE5\_SWinqgeogrpattrs(hid\_t *swathID*, char \**attrnames*, long \**strbufsize*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrnames* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

**Purpose** Retrieve information about group attributes defined in the “Geolocation Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

**Return value** Number of attributes found if successful or FAIL (-1) otherwise.

**Description** The attribute list is returned as a string with each group attribute name separated by commas. If *attrnames* 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 group attributes defined for the “Geolocation Fields” group. In the first call, set the parameter *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_SWinqgeogrpattrs(swathID, NULL, &strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_SWinqgeogrpattrs(swathID, attrnames,  
&strbufsize);
```

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

*"attrOne,attr\_2"*.

**FORTRAN** integer\*4 function he5\_swinqgeogattr(*swathid*, *attrnames*, *strbufsize*)

integer *swathid*

character\*(\*) *attrnames*

integer\*4 *strbufsize*

integer\*4     *nattr*

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

```
nattr = he5_swinqgeogattrs(swathid, attrnames, strbufsize)
```

# Retrieve Information about Geolocation Fields and Aliases Defined in Swath

---

## HE5\_SWinqgfldalias

long HE5\_SWinqgfldalias(hid\_t *swathID*, char \**fldalias*, long \**strbufsize*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fldalias* OUT: List of geolocation fields and aliases (entries separated by commas)

*strbufsize* OUT: String length of geolocation fields and aliases list

Purpose Retrieve information about geolocation fields & aliases defined in swath.

Return value Number of geolocation fields and aliases found if successful or FAIL (-1) otherwise.

Description The list of geolocation fields and aliases is returned as a string with each name separated by commas. If *fldalias* 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 geolocation fields and aliases defined for the “Geolocation Fields” group. In the first call, set the parameter *fldalias* to NULL. We assume that there are one geolocation field and one alias stored, *Latitude* and *Lat*:

```
nfldalias = HE5_SWinqgfldalias(swathID, NULL, &strbufsize);
```

The parameter, *nfldalias*, will have the value 2 and *strbufsize* will have value 12.

```
fldalias = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nfldalias = HE5_SWinqgfldalias(swathID, fldalias,  
&strbufsize);
```

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

*"Latitude,Lat"*.

FORTRAN integer\*4 function he5\_swinqgfldalias(*swathid*, *fldalias*, *strbufsize*)

integer *swathid*

character\*(\*) *fldalias*

integer\*4 *strbufsize*

integer\*4     *nfldalias*

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

```
nfldalias = he5_swinggfldalias(swathid, fldalias,  
strbufsize)
```

# Retrieve Information about Swath Attributes in Group “Data Fields”

---

## HE5\_SWinqgrpattrs

long HE5\_SWinqgrpattrs(hid\_t *swathID*, char \**attrnames*, long \**strbufsize*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrnames* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

Purpose Retrieve information about group attributes in “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each group attribute name separated by commas. If *attrnames* 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 group attributes defined for the “Data Fields” group. In the first call, set the parameter *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_SWinqgrpattrs(swathID, NULL, &strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_SWinqgrpattrs(swathID, attrnames, &strbufsize);
```

The variable, *attrnames*, will be set to: "*attrOne,attr\_2*".

FORTRAN integer\*4 function he5\_swinqgattr(*swathid*, *attrnames*, *strbufsize*)

integer *swathid*

character\*(\*) *attrnames*

integer\*4 *strbufsize*

integer\*4 *nattr*

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

```
nattr = he5_swinqgattr(swathid, attrnames, strbufsize)
```

# Retrieve Information about Indexed Mappings Defined in Swath

---

## HE5\_SWinqidxmaps

long HE5\_SWinqidxmaps(hid\_t *swathID*, char \**idxmap*, hsize\_t *idxsizes*[])

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*idxmap* OUT: Indexed Dimension mapping list (entries separated by commas)

*idxsizes* OUT: Array containing the sizes of the corresponding index arrays.

Purpose Retrieve information about all of the indexed geolocation/data mappings defined in swath.

Return value Number of indexed mapping relations found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper Swath ID.

Description The dimension mapping list is returned as a string with each mapping separated by commas. The two dimensions in each mapping are separated by a slash (/). Output parameters set to *NULL*, will not be returned.

Example In this example, we retrieve information about the indexed dimension mappings:

```
nidxmaps = HE5_SWinqidxmaps(swathID, idxmap, idxsizes);
```

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

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

FORTTRAN integer\*4 function he5\_swinqimaps(*swathid*,*dimmap*,*idxsizes*)

integer *swathid*

character\*(\*) *dimmap*

integer\*4 *idxsizes*(\*)

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

```
nidxmaps = he5_swinqimaps(swathid, dimmap, idxsizes)
```



# Retrieve Information Swath Local Attributes

---

## HE5\_SWinqlocattr

long HE5\_SWinqlocattr(hid\_t *swathID*, const char \**fieldname*, char \**attrnames*, long \**strbufsize*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fieldname* IN: Fieldname to retrieve local attribute information

*attrnames* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

Purpose Retrieve information about local attributes defined for a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each local attribute name separated by commas. If *attrnames* 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 local attributes defined for a field "DataField". In the first call, set the parameter *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_SWinqlocattr(swathID, "DataField", NULL,
&strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_SWinqlocattr(swathID, "DataField", attrnames,
&strbufsize);
```

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

"*attrOne,attr\_2*".

FORTRAN integer\*4 function he5\_swinqlattr(*swathid*, *fieldname*, *attrnames*, *strbufsize*)

integer *swathid*

character\*(\*) *fieldname*

character\*(\*) *attrnames*

integer\*4 *strbufsize*

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

```
nattr = he5_swinqlattr(swathid, "DataField", attrnames,  
strbufsize)
```

# Retrieve Information about Dimension Mappings Defined in Swath

---

## HE5\_SWinqmaps

long HE5\_SWinqmaps(hid\_t *swathID*, char \**dimmap*, long *offset*[], long *increment*[])

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*dimmap* OUT: Dimension mapping list (entries separated by commas)  
*offset* OUT: Array containing the offset of each geolocation relation  
*increment* OUT: Array containing the increment of each geolocation relation  
Purpose Retrieve information about all of the (non-indexed) geolocation relations defined in swath.

Return value Number of geolocation relation entries found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper Swath ID.

Description The dimension mapping list is returned as a string with each mapping separated by commas. The two dimensions in each mapping are separated by a slash (/). Output parameters set to *NULL* will not be returned.

Example In this example, we retrieve information about the dimension mappings in the *ExampleSwath* structure:

```
nmaps = HE5_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}.

FORTTRAN integer\*4 function  
he5\_swinqmaps(*swathid*,*dimmap*,*offset*,*increment*)  
integer *swathid*  
character\*(\*) *dimmap*  
integer\*4 *offset*(\*)  
integer\*4 *increment*(\*)

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

```
nmaps = he5_swinqmaps(swathid, dimmap, offset, increment)
```

# Retrieve Swath Structures Defined in HDF-EOS File

---

## HE5\_SWinqswath

long HE5\_SWinqswath(const char \* *filename*, char \**swathlist*, long \**strbufsize*)

*filename* IN: The HDF-EOS file name

*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 if successful or FAIL (-1) otherwise.

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, *Swath.he5*. In the first call, set *swathlist* to NULL. We assume that there are two swaths stored, *SwathOne* and *Swath\_2*:

```
nswath = HE5_SWinqswath("Swath.he5", NULL, &strbufsize);
```

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

```
swathlist = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nswath = HE5_SWinqswath("Swath.he5", swathlist,  
&strbufsize);
```

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

*"SwathOne,Swath\_2"*.

FORTTRAN integer\*4 function he5\_swinqswath(*filename*,*swathlist*,*strbufsize*)

character\*(\*) *filename*

character\*(\*) *swathlist*

integer\*4 *strbufsize*

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

```
nswath = he5_swinqswath('Swath.he5', swathlist, strbufsize)
```

# Return Information about a Local Swath Attribute

---

## HE5\_SWlocattrinfo

herr\_t HE5\_SWlocattrinfo(hid\_t *swathID*, const char \**fieldname*, const char \**attrname*, hid\_t \**ntype*, hsize\_t \**count*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fieldname* IN: Field name

*attrname* IN: Attribute name

*ntype* OUT: Number type of attribute

*count* OUT: Number of attribute elements

**Purpose** Returns information about local attribute(s) in a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

**Description** This routine returns number type and number of elements (count) of a data field's local attribute.

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

```
status = HE5_SWlocattrinfo(swathID, "DataField", attrname,
&ntype, &count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

**FORTRAN** integer function he5\_swlattrinfo(*swathid*, *fieldname*, *attrname*, *ntype*, *count*)

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

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

```
status = he5_swlattrinfo(swathid, "DataField", attrname,
ntype, count)
```

# Retrieve Offset and Increment of Specific Dimension Mapping

---

## HE5\_SWmapinfo

herr\_t HE5\_SWmapinfo(hid\_t *swathID*, char \**geodim*, char \**datadim*, long \**offset*, long \**increment*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_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 or FAIL (-1) otherwise. A typical reason for failure is the specified mapping does not exist.

Description This routine retrieves offset and increment of the specified geolocation mapping.

Example In this example, we retrieve information about the mapping between the *GeoTrack* and *DataTrack* dimensions:

```
status = HE5_SWmapinfo(swathID, "GeoTrack", "DataTrack",  
                        &offset, &increment);
```

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

FORTTRAN integer function he5\_swmapinfo(*swathid*, *geodim*, *datadim*, *offset*,  
*increment*)

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

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

```
status = he5_swmapinfo(swathid, "GeoTrack", "DataTrack",  
                        offset, increment)
```

# Mount External Data File

---

## HE5\_SWmountexternal

hid\_t HE5\_SWmountexternal(hid\_t *swathID*, int *fldgroup*, const char  
\**extfilename*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fldgroup* IN: Field group flag

*extfilename* IN: External file name

Purpose Mount external data file

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

Description This function allows the user to store required data needed by multiple data files into a separate file so it is not repeated throughout the data files.

Example In this example, we mount a file that contains calibration information needed by the data fields in another file

```
strcpy(extfilename, "/home/user/data/calibration.hdf5");  
fileID = HE5_SWmountexternal(swathID, HE5_HDFE_DATAGROUP,  
extfilename);
```

FORTRAN Not available with this release.

# Return Number of Specified Objects in a Swath

---

## HE5\_SWnentries

long HE5\_SWnentries(hid\_t *swathID*, int *entrycode*, long \**strbufsize*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_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 successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id or entry code.

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

The entry codes are:

HE5\_HDFE\_NENTDIM (0) - Dimensions

HE5\_HDFE\_NENTMAP (1) - Dimension Mappings

HE5\_HDFE\_NENTIMAP (2) - Indexed Dimension Mappings

HE5\_HDFE\_NENTGFLD (3) - Geolocation Fields

HE5\_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 = HE5_SWnentries(swathID, HE5_HDFE_NENTMAP, &bufsize);
```

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

FORTRAN integer\*4 function he5\_swnentries(*swathid*, *entrycode*, *bufsize*)

integer *swathid*

integer *entrycode*

integer\*4 *bufsize*

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

```
parameter (HE5_HDFE_NENTMAP=1)
```

```
nmaps = he5_swnentries(swathid, HE5_HDFE_NENTMAP, bufsize)
```



# Open HDF-EOS File

---

## HE5\_SWopen

hid\_t HE5\_SWopen(const char \*filename, uintn access)

*filename* IN: Complete path and filename for the file to be opened  
*access* IN: H5F\_ACC\_RDONLY, H5F\_ACC\_RDWR or H5F\_ACC\_TRUNC  
Purpose Opens or creates HDF-EOS file in order to create, read, or write a Swath.  
Return value Returns the swath file id handle (fid) if successful or FAIL (-1) otherwise.  
Description This routine creates a new file or opens an existing one, depending on the access parameter.

Access codes:

H5F\_ACC\_RDONLY Open for read only. If file does not exist, error  
H5F\_ACC\_RDWR Open for read/write. If file does not exist, error  
H5F\_ACC\_TRUNC If file exist, delete it, then open a new file for read/write

Example In this example, we create a new swath file named, *Swath.he5*. It returns the file handle, *fid*.

```
fid = HE5_SWopen("Swath.he5", H5F_ACC_TRUNC);
```

FORTRAN integer function he5\_swopen(*filename*, *access*)

character\*(\*) *filename*

integer *access*

The access codes should be defined as parameters:

```
parameter (HE5F_ACC_RDWR = 100)
```

```
parameter (HE5F_ACC_RDONLY = 101)
```

```
parameter (HE5F_ACC_TRUNC = 102)
```

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

```
fid = he5_swopen("Swath.he5", HE5F_ACC_TRUNC)
```

**Note to users of the SDP Toolkit:** Please refer to the *Release 7 SDP Toolkit User Guide for the ECS Project (333-EMD-001Revision 05)*, 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

---

## HE5\_SWperiodinfo

herr\_t HE5\_SWperiodinfo(hid\_t *swathID*, hid\_t *periodID*, char \* *fieldname*, hid\_t \**ntype*, int \**rank*, hsize\_t *dims*[], size\_t \**size*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*periodID* IN: Period ID returned by HE5\_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 or FAIL (-1) otherwise.

Description This routine returns information about a subsetted time period for a particular field. It is useful when allocating space for a data buffer for the subset. Because of differences in number type and geolocation mapping, a given time period will give different values for the dimensions and size for various fields.

Example In this example, we retrieve information about the time period defined in *HE5\_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 = HE5_SWperiodinfo(SWid, periodID, "Spectra", &ntype,
                          &rank, dims, &size);

/* Allocate space */
datbuf = (double *)calloc(size, sizeof(double));
```

FORTRAN integer function `he5_swperinfo`(*swathid*, *periodid*, *fieldname*, *ntype*, *rank*,  
*dims*, *size*)

integer	<i>swathid</i>
integer	<i>periodid</i>
character*(*)	<i>fieldname</i>
integer	<i>ntype</i>
integer	<i>rank</i>
integer*4	<i>dims</i> (*)
integer*4	<i>size</i>

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

```
status=he5_swperinfo(swid,periodid,"Spectra",ntype,rank,dim,  
size)
```

# Read Swath Attribute

---

## HE5\_SWreadattr

herr\_t HE5\_SWreadattr(hid\_t *swathID*, const char \**attrname*, void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads object attribute from a specific swath. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.

Example In this example, we read a floating point attribute with the name "ScalarFloat":

```
status = HE5_SWreadattr(swathID, "ScalarFloat", &data);
```

FORTTRAN integer function he5\_swrdatr(*swathid*,*attrname*,*datbuf*)

integer *swathid*

<valid type> *datbuf*(\*)

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

```
status = he5_swrdatr(swathid, "ScalarFloat", datbuf)
```

# Read External Data Set

---

## HE5\_SWreadexternal

herr\_t HE5\_SWreadexternal(hid\_t *swathID*, int *fldgroup*, const char \**fieldname*, void \**buffer*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fldgroup* IN: Field group flag

*fieldname* IN: Name of field to read

*buffer* OUT: Output data buffer

Purpose Read external data set

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

Description This function allows the user to get the data required from the external data file.

Example In this example, the field “Cal data” is read from the external file:

```
strcpy(fieldname, "Cal data");  
  
status = HE5_SWreadexternal(swathID, HE5_HDFE_DATAGROUP,  
fieldname, buffer);
```

FORTTRAN Not available with this release.

# Read Data from a Swath Field

---

## HE5\_SWreadfield

herr\_t HE5\_SWreadfield(hid\_t *swathID*, char \**fieldname*, const hssize\_t *start*[], const hssize\_t *stride*[], const hssize\_t *edge*[], void \**buffer*)

<i>swathID</i>	IN:	Swath ID returned by HE5_SWcreate or HE5_SWattach
<i>fieldname</i>	IN:	Name of field to read
<i>start</i>	IN:	Array specifying the starting location within each dimension
<i>stride</i>	IN:	Array specifying the number of values to skip along each dimension
<i>edge</i>	IN:	Array specifying the number of values to read along each dimension
<i>buffer</i>	OUT:	Buffer to store the data read from the field
Purpose		Reads data from a swath field.
Return value		Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are improper swath id or unknown fieldname.
Description		The values within <i>start</i> , <i>stride</i> , and <i>edge</i> arrays refer to the swath field (input) dimensions. The output data in <i>buffer</i> is written to contiguously. The default 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. Note that to allocate a string buffer size for reading an array of strings, first using <i>HE5_SWreadlocattr</i> to get the value of maximum string length in the local attribute <i>StringLengthAttribute</i> .

Example In this example, we read data from the 10th track (0-based) of the *Longitude* field.

```
float    track[1000];
hssize_t start[2] = {9,1};
hssize_t edge[2]  = {1,1000};
status = HE5_SWreadfield(swathID, "Longitude", start, NULL,
                        edge, track);
```

FORTRAN integer function

he5\_swrdfld(*swathid, fieldname, start, stride, edge,buffer*)

he5\_swrdfld(*swathid, fieldname, elemlen, numelem, start, stride, edge,buffer*)

integer *swathid*

character\*(\*) *fieldname*

integer *elemlen* (each element length in array of string)

integer *numelem* (number of elements in declared buffer array)

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.

**Note: he5\_swrdfld() is only for reading an array of character string field. For reading an array of single character field, please use he5\_swrdfld().**

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=he5_swrdfld(swathid, "Longitude", start, stride,
edge, track)
```

# Read Group Swath Attribute in Group “Geolocation Fields”

---

## HE5\_SWreadgeogrpatrr

herr\_t HE5\_SWreadgeogrpatrr(hid\_t *swathID*, const char \**attrname*, void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads group attribute from the “Geolocation Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.

Example In this example, we read a floating point attribute with the name "ScalarFloat":

```
status = HE5_SWreadgeogrpatrr(swathID, "ScalarFloat",
&data);
```

FORTRAN integer function he5\_swrddgeogattr(*swathid*,*attrname*,*datbuf*)

```
integer      swathid
```

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

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

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

```
status = he5_swrddgeogattr(swathid, "ScalarFloat", datbuf)
```



# Read Group Swath Attribute in Group “Data Fields”

---

## HE5\_SWreadgrpattr

herr\_t HE5\_SWreadgrpattr(hid\_t *swathID*, const char \**attrname*, void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads group attribute from the “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.

Example In this example, we read a floating point attribute with the name "ScalarFloat":

```
status = HE5_SWreadgrpattr(swathID, "ScalarFloat", &data);
```

FORTTRAN integer function he5\_swrddgattr(*swathid*,*attrname*,*datbuf*)

```
integer      swathid
```

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

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

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

```
status = he5_swrddgattr(swathid, "ScalarFloat", datbuf)
```

# Read Local Swath Attribute

---

## HE5\_SWreadlocattr

herr\_t HE5\_SWreadlocattr(hid\_t *swathID*, const char \**fieldname*, const char \**attrname*, void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*fieldname* IN: Field name  
*attrname* IN: Attribute name  
*datbuf* OUT: Buffer allocated to hold attribute values  
Purpose Reads a local attribute from a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).  
Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type or incorrect attribute name.  
Description The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.

Example In this example, we read a single precision (32 bit) floating point attribute with the name "ScalarFloat":

```
status = HE5_SWreadlocattr(swathID, "DataField",  
"ScalarFloat", &data);
```

FORTTRAN integer function he5\_swrldattr(*swathid*, *fieldname*, *attrname*, *datbuf*)

```
integer      swathid  
character*(*) fieldname  
character*(*) attrname  
<valid type> datbuf(*)
```

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

```
status = he5_swrldattr(swathid, "DataField", "ScalarFloat",  
datbuf)
```

# Define a Longitude-Latitude Box Region for a Swath

---

## HE5\_SWregionindex

hid\_t HE5\_SWregionindex(hid\_t *swathID*, double *cornerlon*[], double *cornerlat*[], int *mode*, char \**geodim*, hsize\_t *idxrange*[])

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_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

*geodim* OUT: Geolocation track dimension

*idxrange* OUT: The indices of the region in the geolocation track dimension.

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

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

Description The difference between this routine and *HE5\_SWdefbboxregion* is the geolocation track dimension name and the range of that dimension are returned in addition to a regionID. Other than that difference they are the same function and this function is used just like *HE5\_SWdefbboxregion*. This routine defines a longitude-latitude box region for a swath. It returns a swath region ID which is used by the *HE5\_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" (HE5\_HDFE\_MIDPOINT), or 2) either of its endpoints is within the longitude-latitude "box" (HE5\_HDFE\_ENDPOINT), or 3) any point of the cross track is within the longitude-latitude "box" (HE5\_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 = HE5_SWregionindex(swathID, cornerlon, cornerlat,  
HE5_HDFE_MIDPOINT, geodim, idxrange);
```

FORTRAN integer function he5\_swregidx(*swathid, cornerlon, cornerlat, mode,*  
*geodim, idxrange*)

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

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

```
parameter (HE5_HDFE_MIDPOINT=0)  
cornerlon(1) = 3.  
cornerlat(1) = 5.  
cornerlon(2) = 7.  
cornerlat(2) = 12.  
regionid = he5_swregidx(swathid, cornerlon, cornerlat,  
HE5_HDFE_MIDPOINT, geodim, idxrange)
```

## Return Information about a Defined Region

---

### HE5\_SWregioninfo

herr\_t HE5\_SWregioninfo(hid\_t *swathID*, hid\_t *regionID*, char \**fieldname*, hid\_t \**ntype*, int \**rank*, hsize\_t *dims*[], size\_t \**size*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*regionID* IN: Region ID returned by HE5\_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 or FAIL (-1) otherwise.

Description This routine returns information about a subsetted region for a particular field. It is useful when allocating space for a data buffer for the region. Because of differences in number type and geolocation mapping, a given region will give different values for the dimensions and size for various fields.

Example In this example, we retrieve information about the region defined in *HE5\_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 = HE5_SWregioninfo(SWid, regionID, "Spectra", &ntype,
    &rank, dims, &size);

/* Allocate space */
datbuf = (double *)calloc(size, sizeof(double));
```

FORTRAN integer function `he5_swreginfo`(*swathid*, *regionid*, *fieldname*, *ntype*, *rank*,  
*dims*, *size*)

integer	<i>swathid</i>
integer	<i>regionid</i>
character*(*)	<i>fieldname</i>
integer	<i>ntype</i>
integer	<i>rank</i>
integer*4	<i>dims</i> (*)
integer*4	<i>size</i>

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

```
status =  
he5_swreginfo(swid,regionid,"Spectra",ntype,rank,dims,size)
```

# Create an Alias for Swath Data Field

---

## HE5\_SWsetalias

herr\_t HE5\_SWsetalias(hid\_t *swathID*, char \**fieldname*, const char \**aliaslist*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fieldname* IN: Field name

*aliaslist* IN: List of alias(es) to associate with the Data Field

Purpose Create an alias for Swath data field

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

Description Creates aliases that can be used to refer to a Swath data field in addition to the name of the field.

Example In this example, we create an alias for the data field *Temperature*.

```
strcpy(aliaslist, "temps 0 to 30");  
status = HE5_SWsetalias(swathID, "Temperature", aliaslist);
```

FORTRAN integer function he5\_swsetalias (*swathid*, *fieldname*, *aliaslist*)  
integer *swathid*  
character\*(\*) *fieldname*  
character\*(\*) *aliaslist*

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

```
aliaslist = "temps 0 to 30"  
status = he5_swsetalias(swathid, "Temperature", aliaslist)
```

## Set External Data File(s)

---

### HE5\_SWsetextdata

herr\_t HE5\_SWsetextdata(hid\_t *swathID*, const char \**filelist*, off\_t *offset[]*, hsize\_t *size[]*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*filelist* IN: List of external file names  
*offset[]* IN: Array of offsets (in byte) from the beginning of file to the location in file where the data starts  
*size[]* IN: Array of sizes (in bytes) reserved in the file for the data

Purpose Sets the external data file(s) associated with the data set.

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

Example In this example, we set the *ExtData* field:

```
status = HE5_SWsetextdata(swathID, "ext-1.dat,ext-2.dat,ext-3.dat", offset, size);
```

FORTRAN integer function he5\_swsetxdat(*swathid*,*fllist*,*offset*,*size*)

integer *swathid*

integer *status*

integer\*4 *offset*(\*)

integer\*4 *size*(\*)

character\*(\*) *fllist*

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

```
status = he5_swsetxdat(swathid,fllist,offset,size)
```



## Set Fill Value for a Specified Field

---

### HE5\_SWsetfillvalue

herr\_t HE5\_SWsetfillvalue(hid\_t *swathID*, char \**fieldname*, hid\_t *ntype*, void \**fillvalue*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*fieldname* IN: Field name  
*ntype* IN: Number type of fill value (should match the number type of a specified field)  
*fillvalue* IN: Pointer to the fill value to be used

**NOTE: THIS FUNCTION MUST BE CALLED BEFORE THE FUNCTION CALL TO DEFINE THE FIELD IT IS TO BE APPLIED. SETS A FILL VALUE FOR A CHARACTER STRING FIELD IS NOT AVAILABLE IN THIS RELEASE.**

Purpose Sets fill value for the specified field.

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

Description The fill value is placed in all elements of the field which have not been explicitly defined.

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

```
tempfill = -999.0;  
  
status = HE5_SWsetfillvalue(swathID, "Temperature", ntype,  
&tempfill);
```

FORTTRAN integer function he5\_swsetfill(*swathid*, *fieldname*, *ntype*, *fillvalue*)

```
integer      swathid  
character*(*) fieldname  
integer      ntype  
<valid type> fillvalue(*)
```

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

```
fillvalue = -999.0
```

```
status = he5_swsetfill(swathid, "Temperature", ntype,  
fillvalue)
```

# Dismount External Data File

---

## HE5\_SWunmount

herr\_t HE5\_SWunmount(hid\_t *swathID*, int *fldgroup*, hid\_t *fileID*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fldgroup* IN: Field group flag

*fileID* IN: ID of file returned by HE5\_SWmountexternal

Purpose Dismount external data file

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

Description This function dismounts from the external file once the user has completed using the data in the file.

Example In this example, we dismount from the file used in the previous function

```
status = HE5_SWunmount(swathID, HE5_HDFE_DATAGROUP, fileID);
```

FORTRAN Not available with this release.

# Update Map Index for a Specified Region

---

## HE5\_SWupdateidxmap

long HE5\_SWupdateidxmap(int *swathID*, hid\_t *regionID*, long *indexin*[], long *indexout*[], long *indices*[])

<i>swathID</i>	IN: Swath ID returned by HE5_SWcreate or Swattach.
<i>regionID</i>	IN: Region ID returned by HE5_SWdefboxregion.
<i>indexin</i>	IN: The array containing the indices of the data dimension to which each geolocation element corresponds.
<i>indexout</i>	OUT: The array containing the indices of the data dimension to which each geolocation corresponds in the subsetted region. The <i>indexout</i> set to NULL, will not be returned.
<i>indices</i>	OUT: The array containing the indices for start and stop of region.
Purpose	Retrieve indexed array of specified geolocation mapping for a specified region.
Return value	Returns size of updated indexed array if successful or FAIL (-1) otherwise. A typical reason for failure is the specified mapping does not exist.
Description	This routine retrieves the size of the indexed array and the array of indexed elements of the specified geolocation mapping for the specified region.
Example	In this example, we retrieve information about the indexed mapping between the “IdxGeo” and “IdxData” dimensions, defined by <i>HE5_Swdefboxregion</i> . In the first call, set <i>index_region</i> to NULL:

```
/* Get size of index_region array */
idxsz = HE5_SWupdateidxmap(swathID, regionID, index, NULL,
indices);

/* Allocate memory for index_region */
index_region = (long)malloc(sizeof(long) * idxsz);

/* Get the array index_region */
idxsz = HE5_SWupdateidxmap(swathID, regionID, index,
index_region, indices);
```

FORTTRAN    integer\*4 function he5\_swupimap(*swathid, regionid, indexin, indexout, indices*)

              integer *swathid*

              integer *regionid*

              integer\*4 *indexin*(\*)

              integer\*4 *indexout*(\*)

              integer\*4 *indices*(2)

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

```
status = he5_swupdateidxmap(swathid, regionid, index, index_region, indices)
```

# Write/Update Swath Attribute

---

## HE5\_SWwriteattr

herr\_t HE5\_SWwriteattr(hid\_t *swathID*, const char \**attrname*, hid\_t *ntype*, hsize\_t *count*[], void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_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 an object attribute in a specific swath. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

**Description** If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call.

**Example** In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14:

```
attr_val = 3.14;
```

```
status = HE5_SWwriteattr(swathid, "ScalarFloat",  
H5T_NATIVE_FLOAT, 1, &attr_val);
```

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

```
attr_val = 3.14159;
```

```
status = HE5_SWwriteattr(swathid, "ScalarFloat",  
H5T_NATIVE_FLOAT, 1, &attr_val);
```

FORTRAN integer function `he5_swwrattr(swathid, attrname, ntype, count, datbuf)`  
integer *swathid*  
character\*(\*) *attrname*  
integer\*4 *count*(\*)  
<valid type> *datbuf*(\*)

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

```
parameter (HE5T_NATIVE_FLOAT = 10)
datbuf = 3.14
count = 1
status = he5_swwrattr(swathid, "ScalarFloat",
HE5T_NATIVE_FLOAT, count, datbuf)
```

# Write Field Metadata for an Existing Swath Data Field

---

## HE5\_SWwritedatameta

herr\_t HE5\_SWwritedatameta(hid\_t *swathID*, const char \**fieldname*, char \**dimlist*, int *mvalue*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fieldname* IN: Name of field

*dimlist* IN: The list of data dimensions defining the field

*mvalue* 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 or 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 *HE5\_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 = HE5_SWwritedatameta(swathID, "Band_1", "GeoTrack,
                             GeoXtrack", H5T_NATIVE_FLOAT);
```

FORTRAN integer function

```
he5_sw wrdmeta(swathid,fieldname,dimlist,mvalue)
```

```
integer swathid
```

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

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

```
integer mvalue
```

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

```
parameter (HE5T_NATIVE_FLOAT = 10)
```

```
status = he5_sw wrdmeta(swathID, "Band_1", "GeoXtrack,
                       GeoTrack", HE5T_NATIVE_FLOAT)
```

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



# Write Data to a Swath Field

---

## HE5\_SWwritefield

herr\_t HE5\_SWwritefield(hid\_t *swathID*, char \**fieldname*, const hssize\_t *start*[], const hsize\_t *stride*[], const hsize\_t *edge*[], void \**data*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fieldname* IN: Name of field to write

*start* IN: Array specifying the starting location within each dimension (0-based)

*stride* IN: Array specifying the number of values to skip along each dimension

*edge* IN: Array specifying the number of values to write along each dimension

*data* IN: Values to be written to the field

Purpose Writes data to a swath field.

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

Description The values within *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.

```
float longitude [2000][1000];

/* Define elements of longitude array */

status = HE5_SWwritefield(swathID, "Longitude", NULL, NULL,
                          NULL, longitude);
```

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

```
float newtrack[1000];  
hssize_t start[2]={10,0}; hsize_t edge[2]={1,1000};  
/* Define elements of newtrack array */  
status = HE5_SWwritefield(swathID, "Longitude", start, NULL,  
edge, newtrack);
```

FORTTRAN

integer function

he5\_swwrflld(*swathid,fieldname,start,stride,edge,data*)

he5\_swwrcharfld(*swathid,fieldname,elemen,numelem,start,stride,edge,  
data*)

integer *swathid*

character\*(\*) *fieldname*

integer *elemen* (*each element length in array of string*)

integer *numelem* (*number of elements in declared buffer array*)

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.

**Note: he5\_swwrcharfld() is only for writing an array of character string field. For writing an array of single character field, please use he5\_swwrflld().**

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 = he5_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 = he5_swwrfld(swathid, "Longitude", start, stride,  
edge, newtrack)
```

# Write/Update Group Attribute in “Geolocation Fields” Group

---

## HE5\_SWwritegeogrpatr

herr\_t HE5\_SWwritegeogrpatr(hid\_t *swathID*, const char \**attrname*, hid\_t *ntype*, hsize\_t *count*[], void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrname* IN: Attribute name

*ntype* IN: Data type of attribute

*count* IN: Number of values to store in attribute

*datbuf* IN: Attribute values

**Purpose** Writes/Updates group attribute in the “Geolocation Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

**Description** If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to the group “Geolocation Fields” in the swath file.

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

```
count[0] = 1;

attr_val = 3.14;

status = HE5_SWwritegeogrpatr(swathid, "ScalarFloat",
H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```
attr_val = 3.14159;

status = HE5_SWwritegeogrpatr(swathid, "ScalarFloat",
H5T_NATIVE_FLOAT, count, &attr_val);
```

FORTRAN integer function `he5_swvrgeogattr(swathid, attrname, ntype, count,  
datbuf)`

integer *swathid*

character\*(\*) *attrname*

integer *ntype*

integer\*4 *count*(\*)

<valid type> *datbuf*(\*)

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

```
parameter (HE5T_NATIVE_FLOAT=10)
datbuf = 3.14
count = 1
status = he5_swvrgeogattr(swathid, "ScalarFloat",
HE5T_NATIVE_FLOAT, count, datbuf)
```

# Write Field Metadata to an Existing Swath Geolocation Field

---

## HE5\_SWwritegeometa

herr\_t HE5\_SWwritegeometa(hid\_t *swathID*, const char \**fieldname*, char \**dimlist*, int *mvalue*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fieldname* IN: Name of field

*dimlist* IN: The list of geolocation dimensions defining the field

*mvalue* 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 or 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 *HE5\_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 = HE5_SWwritegeometa(swathID, "Latitude",  
                             "GeoTrack,GeoXtrack",HE5T_NATIVE_FLOAT);
```

FORTRAN integer function

```
he5_swrgmeta(swathid,fieldname,dimlist,mvalue)
```

```
integer      swathid
```

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

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

```
integer      mvalue
```

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

```
parameter (HE5T_NATIVE_FLOAT = 10)  
status = he5_swrgmeta(swathID, "Latitude",  
                     "GeoXtrack,GeoTrack",HE5T_NATIVE_FLOAT)
```

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

# Write/Update Group Attribute in “Data Fields” Group

---

## HE5\_SWwritegrpattr

herr\_t HE5\_SWwritegrpattr(hid\_t *swathID*, const char \**attrname*, hid\_t *ntype*, hsize\_t *count*[], void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrname* IN: Attribute name

*ntype* IN: Data type of attribute

*count* IN: Number of values to store in attribute

*datbuf* IN: Attribute values

**Purpose** Writes/Updates group attribute in the “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

**Description** If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to the “Data Fields” group in the swath file.

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

```
count[0] = 1;
attr_val = 3.14;
status = HE5_SWwritegrpattr(swathid, "ScalarFloat",
H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```
attr_val = 3.14159;
status = HE5_SWwritegrpattr(swathid, "ScalarFloat",
H5T_NATIVE_FLOAT, count, &attr_val);
```



FORTRAN integer function `he5_swwrgattr(swathid, attrname, ntype, count, datbuf)`

integer *swathid*

character\*(\*) *attrname*

integer *ntype*

integer\*4 *count*(\*)

<valid type> *datbuf*(\*)

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

```
parameter (HE5T_NATIVE_FLOAT=10)

datbuf = 3.14

count = 1

status = he5_swwrgattr(swathid, "ScalarFloat",
HE5T_NATIVE_FLOAT, count, datbuf)
```

## Write/Update Local Swath Attribute

---

### HE5\_SWwritelocatrr

herr\_t HE5\_SWwritelocatrr(hid\_t *swathID*, const char \**fieldname*, const char \**attrname*, hid\_t *ntype*, hsize\_t *count*[], void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*fieldname* IN: Field name

*attrname* IN: Attribute name

*ntype* IN: Data type of attribute

*count* IN: Number of values to store in attribute

*datbuf* IN: Attribute values

**Purpose** Writes/Updates local attribute in a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

**Description** If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to a particular field "DataField" in the swath file.

**Example** In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14:

```
countt[0] = 1;
attr_val = 3.14;

status = HE5_SWwritelocatrr(swathid, "DataField",
"ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```
attr_val = 3.14159;

status = HE5_SWwritelocatrr(swathid, "DataField",
"ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

FORTRAN integer function `he5_swwrlattr(swathid, fieldname, attrname, ntype,  
count, datbuf)`

integer *swathid*  
character\*(\*) *fieldname*  
character\*(\*) *attrname*  
integer *ntype*  
integer\*4 *count*(\*)  
<valid type> *datbuf*(\*)

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

```
parameter (HE5T_NATIVE_FLOAT=10)  
  
datbuf = 3.14  
  
count = 1  
  
status = he5_swwrlattr(swathid, "DataField", "ScalarFloat",  
HE5T_NATIVE_FLOAT, count, datbuf)
```

# Define Profile Data Structure

---

## HE5\_PRdefine

herr\_t HE5\_PRdefine(hid\_t *swathID*, const char \**profilename*, chat \**dimlist*, char \**maxdimlist*, hid\_t *datatype\_id*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*profilename* IN: Profile name

*dimlist* IN: List of profile dimensions (separated by comma)

*maxdimlist* IN: List of profile maximum dimensions (separated by comma)

*dtype* IN: Base data type ID

Purpose Sets up a specified profile structure in a swath.

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

Description The profile is linked to the “Profile Fields” group in the swath file.

Example In this example, we define a profile with the name *SimpleProfile* and with the base ‘unsigned int’ data type. The profile is represented by a single dataset with 4 dimensions.

```
status = HE5_PRdefine(swathid, "SimpleProfile", dimlist,
maxdimlist, H5T_NATIVE_UINT);
```

FORTRAN integer function he5\_prdefine(*swathid*, *profilename*, *rank*, *dim*, *datatype\_id*)

integer *swathid*

character\*(\*) *profilename*

character\*(\*) *dimlist*

character\*(\*) *maxdimlist*(\*)

integer *datatype\_id*

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

```
parameter (HE5T_NATIVE_INT = 0)
```

```
status = he5_prdefine(swathid, "SimpleProfile", dimlist, maxdimlist,
HE5T_NATIVE_INT)
```

# Return Information about a Profile Group Swath Attribute

---

## HE5\_PRgrpattrinfo

herr\_t HE5\_PRgrpattrinfo(hid\_t *swathID*, const char \**attrname*, hid\_t \**ntype*, hsize\_t \**count*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*attrname* IN: Attribute name  
*numbertype* OUT: Number type of attribute. See Appendix A for interpretation of number types.  
*count* OUT: Number of attribute elements  
Purpose Returns information about a group attribute in the “Profile Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description This routine returns number type and number of elements (count) of a swath group attribute.

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

```
status = HE5_PRgrpattrinfo(swathID, "ScalarFloat", &nt, &count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

FORTRAN integer function he5\_prgattrinfo(*swathid*, *attrname*, *ntype*, *count*,)  
integer *swathid*  
character\*(\*) *attrname*  
integer *ntype*  
integer \*4 *count*

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

```
status = he5_prgattrinfo(swathid, "ScalarFloat",nt,count)
```

## Return Information about a Profile in a Swath

---

### HE5\_PRinfo

herr\_t HE5\_PRinfo(hid\_t *swathID*, const char \**profname*, int \**rank*, hsize\_t *dims*[], hsize\_t *maxdims*[], hid\_t \**ntype*, char \**dimlist*, char \**maxdimlist*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*profname* IN: Profile name  
*rank* OUT: Rank of profile dataset  
*dims* OUT: Array of dimension sizes  
*maxdims* OUT: Array of maximum dimension sizes  
*ntype* OUT: Base-number type ID  
*dimlist* OUT: Comma separated list of dimension names  
*maxdimlist* OUT: Comma separated list of maximum dimension names  
Purpose Retrieve information about specified profile dataset in a Swath  
Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.  
Description This routine returns rank, array of dimension and maximum dimension sizes, base number type ID, comma separated list of dimension and maximum dimension names of profile dataset.

Example In this example, we retrieve information about profile "Profile-2000":

```
status = HE5_PRinfo(swathID, "Profile-2000", rank, dims,
maxdims, ntype, dimlist, maxdimlist);
```

FORTRAN integer function he5\_prinfo( *swathid*, *profname*, *rank*, *dims*, *maxdims*,  
*ntype*, *dimlist*, *maxdimlist*)  
integer *swathid*  
character\*(\*) *profname*  
integer *rank*  
integer\*4 *dims*(\*)  
integer\*4 *maxdims*(\*)  
integer *ntype*  
character\*(\*) *dimlist*  
character\*(\*) *maxdimlist*

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

```
profname = "Profile-2000"  
  
status = he5_prinfo(swathid, profname, rank, dims, maxdims,  
ntype, dimlist, maxdimlist)
```

# Retrieve Information about Profile Group Attributes

---

## HE5\_PRinqgrpattrs

long HE5\_PRinqgrpattrs(hid\_t *swathID*, char \**attrnames*, long \**strbufsize*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach  
*attrnames* OUT: Attribute list (entries separated by commas)  
*strbufsize* OUT: String length of attribute list  
Purpose Retrieve information about profile group attributes defined in the “Profile Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each group attribute name separated by commas. If *attrnames* 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 group attributes defined for the “Profile Fields” group. In the first call, set *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_PRinqgrpattrs(swathID, NULL, &strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_PRinqgrpattrs(swathID, attrnames, &strbufsize);
```

The variable, *attrnames*, will be set to: "*attrOne,attr\_2*".

FORTRAN integer\*4 function he5\_prinqgattr(*swathid*, *attrnames*, *strbufsize*)

integer *swathid*

character\*(\*) *attrnames*

integer\*4 *strbufsize*

integer\*4 *nattr*

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

```
nattr = he5_prinqgattr(swathid, attrnames, strbufsize)
```



## Retrieve Information about Profiles in a Swath

---

### HE5\_PRInquire

long HE5\_PRInquire(hid\_t *swathID*, char \**profnames*, int \**rank*, H5T\_class\_t \**classID*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*profnames* OUT: Buffer for returned comma separated list of profile names

*rank* OUT: Array of ranks of profile datasets

*classID* OUT: Array of base-data type class IDs of profiles

Purpose Retrieve information about profile datasets in a specified Swath

Return value Returns number of profiles if successful or FAIL (-1) otherwise.

Description A comma separated list of profile datasets is returned. The *rank* and (base data type) *classID* arrays will have an entry for each profile.

Example In this example, we retrieve information about profiles:

```
nprof = HE5_PRInquire(swathID, profnames, rank, classID);
```

FORTTRAN integer\*4 function he5\_prinquire (*swathid*, *profnames*, *rank*, *classID*)  
integer *swathid*  
character\*(\*) *profnames*  
integer *rank*(\*)  
integer *classID*(\*)

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

```
nprof = he5_prinquire(swathid, profnames, rank, classID)
```

# Read Data from Profile Structure

---

## HE5\_PRread

herr\_t HE5\_PRread(hid\_t *swathID*, const char \**profilename*, const hssize\_t *start*[], const hsize\_t *stride*[], const hsize\_t *edge*[], void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*profilename* IN: Profile structure name

*start* IN: Array specifying starting location within each dimension

*stride* IN: Array specifying the number of values to skip along each dimension

*edge* IN: Array specifying the number of values to write along each dimension

*datbuf* OUT: Buffer allocated to hold profile values

Purpose Reads profile data set from a swath.

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

Description After reading the data a call to *HE5\_PRreclaimspace()* should be made to release allocated memory.

Example In this example, we read an 'unsigned int' type profile with the name "SimpleProfile":

```
/* Native HDF5 Datatype used in this example: Variable
Length Datatype struct */
typedef struct {
    size_t len; /* Length of VL data (for base type)*/
    void *p; /* Pointer to VL data */
} hvl_t;

hvl_t buffer[4];

start[0] = 0;

stride[0] = 1;
```

```

edge[0]    = 4;

status = HE5_PRread(swathID, "SimpleProfile", start, stride,
edge, buffer);

for (i=0; i<4; i++){
printf("The length of %d-th element is %d \n",
i,(unsigned)buffer[i].len);

    for (j=0; j<2; j++)
        printf("%d \n", ((unsigned int*)buffer[i].p)[j]);
}

status = HE5_PRreclaimspace(swathID, "SimpleProfile",
buffer);

```

FORTRAN    integer function  
he5\_prread(*swathid, profname, start, stride, count, len, buffer*)  
integer        *swathid, status*  
character \*(\*) *profname*  
integer\*4      *start(2), stride(2), count(2), len(4)*

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

```

start(1) = 0
stride(1) = 1
count(1) = 4

status = he5_prread(swathid, "SimpleProfile", start, stride,
count, len, buffer)

```

# Read Profile Group Swath Attribute

---

## HE5\_PRreadgrpattr

herr\_t HE5\_PRreadgrpattr(hid\_t *swathID*, const char \**attrname*, void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads group attribute from the “Profile Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.

Example In this example, we read a floating point attribute with the name "ScalarFloat":

```
status = HE5_PRreadgrpattr(swathID, "ScalarFloat", &data);
```

FORTRAN integer function he5\_prrdgattr(*swathid*,*attrname*,*datbuf*)

integer *swathid*

character\*(\*) *attrname*

<valid type> *datbuf*(\*)

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

```
status = he5_prrdgattr(swathid, "ScalarFloat", datbuf)
```

# Reclaim Memory used by “Read” Buffer

---

## HE5\_PRreclaimspace

herr\_t HE5\_PRreclaimspace(hid\_t *swathID*, const char \**profilename*, void \**buffer*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*profilename* IN: Profile name

*buffer* IN: Data buffer used to read profile dataset

Purpose Release memory used by the buffer in the call HE5\_PRread()

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

Description Reclaims memory space allocated to the data buffer in the call *HE5\_PRread()*.

Example In this example, we reclaim memory allocated for the “read” buffer “buffer”

```
status = HE5_PRreclaimspace(swathID, "Profile-2000",  
buffer);
```

FORTTRAN Not needed.

# Write Data to the Profile Swath Structure

---

## HE5\_PRwrite

herr\_t HE5\_PRwrite(int *swathID*, const char \**profilename*, const hssize\_t *start*[], const hsize\_t *stride*[], const hsize\_t *edge*[], size\_t *size*, void \**datbuf*)

<i>swathID</i>	IN:	Swath ID returned by HE5_SWcreate or HE5_SWattach
<i>profilename</i>	IN:	Profile structure name
<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>size</i>	IN:	Size of data buffer (in bytes) for memory allocation routine
<i>datbuf</i>	IN:	Profile data values
Purpose		Writes profile data set in a swath.
Return value		Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or profile name.
Description		The specified profile is linked to a "Profile Fields" group in the swath file.
Example		In this example, we write data to "SimpleProfile" :

```
size_t      datasize = 0;
hvl_t      buf[4];
for (i = 0; i < 4; i++){
    buf[i].p = malloc(25*(i+1)*sizeof(unsigned int));
    buf[i].len = 25*(i+1);
    datasize += buf[i].len *sizeof(unsigned int);
    for (j = 0; j < 25*(i+1); j++)
        ((unsigned int )buf[i].p)[j] = (i+1)*10+j;
}
```

```
status = HE5_PRwrite(swathid, "SimpleProfile", start,  
stride, edge, datasize, buf);
```

FORTTRAN integer function he5\_prwrite(*swathid, profname, start, stride, count,*  
*datasize, len, buffer*)

integer *swathid, status*

integer\*4 *start(3), stride(3), count(3), len(4), datasize*

integer *buffer(\*), i, j, counter*

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

```
datasize = 0
```

```
counter = 0
```

```
do i=1,4
```

```
    len(i) = i*25
```

```
    datasize = datasize + len(i)
```

```
    do j = 1, (25*i)
```

```
        counter = counter + 1
```

```
        buffer(counter) = (i)*1000+j-1
```

```
    enddo
```

```
enddo
```

```
start(1) = 0
```

```
stride(1) = 1
```

```
count(1) = 4
```

```
status = he5_prwrite(swathid, "SimplePrifile", start,  
stride, count, datasize, len, buffer)
```

# Write/Update Profile Group Swath Attribute

---

## HE5\_PRwritegrpattr

herr\_t HE5\_PRwritegrpattr(hid\_t *swathID*, const char \**attrname*, hid\_t *ntype*, hsize\_t *count*[], void \**datbuf*)

*swathID* IN: Swath ID returned by HE5\_SWcreate or HE5\_SWattach

*attrname* IN: Attribute name

*ntype* IN: Data type of attribute

*count* IN: Number of values to store in attribute

*datbuf* IN: Attribute values

**Purpose** Writes/Updates group attribute in the “Profile Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

**Description** If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to the “Profile Fields” group in the swath file.

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

```
count[0] = 1;
attr_val = 3.14;

status = HE5_PRwritegrpattr(swathid, "ScalarFloat",
H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```
attr_val = 3.14159;

status = HE5_PRwritegrpattr(swathid, "ScalarFloat",
H5T_NATIVE_FLOAT, count, &attr_val);
```



FORTRAN integer function `he5_prwrgattr(swathid, attrname, ntype, count, datbuf)`  
integer *swathid*  
character\*(\*) *attrname*  
integer *ntype*  
integer\*4 *count*(\*)  
<valid type> *datbuf*(\*)

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

```
parameter (HE5T_NATIVE_FLOAT=10)  
  
datbuf = 3.14  
  
count = 1  
  
status = he5_prwrgattr(swathid, "ScalarFloat",  
HE5T_NATIVE_FLOAT, count, datbuf)
```

### **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.

# Return Information About an Alias

---

## HE5\_GDaliasinfo

herr\_t HE5\_GDaliasinfo(hid\_t *gridID*, int *fldgroup*, const char \**aliasname*, int \**length*, char \**buffer*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*fldgroup* IN: Field group flag

*aliasname* IN: Name of alias to retrieve information about

*length* IN/OUT: Size of buffer in bytes

*buffer* OUT: Buffer with original field name

Purpose Return information about an alias

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

Description This routine returns buffer size and the buffer with original field name.

Example In this example, we return the buffer size and the original data field name *Temperature*. In the first call, set *buffer* to NULL and *length* is an output parameter. In the second call, *length* is an input parameter.

```
status = HE5_GDaliasinfo(gridID, HE5_HDFE_DATAGROUP,
aliasname, length, NULL);
```

```
namebuffer = (char *)calloc(length + 1, sizeof(char));
```

```
status = HE5_GDaliasinfo(gridID, HE5_HDFE_DATAGROUP,
aliasname, length, namebuffer);
```

FORTRAN integer function he5\_gdaliasinfo (*gridid*, *fldgroup*, *aliasname*, *length*,  
*buffer*)

integer *gridid*, *status*

character\*(\*) *fldgroup*

character\*(\*) *aliasname*

integer\*(\*) *length*

character\*(\*) *buffer*

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

```
aliaslist = "temps 0 to 30"
```

```
status = he5_gdaliasinfo(gridid, HE5_HDFE_DATAGROUP,
aliaslist, length, buffer)
```

# Attach to an Existing Grid Structure

---

## HE5\_GDattach

hid\_t HE5\_GDattach(hid\_t *fid*, char \**gridname*)

*fid* IN: Grid file ID returned by HE5\_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 or FAIL(-1) otherwise. Typical reasons for failure are improper grid file id or grid name.

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

Example In this example, we attach to the previously created grid, "*ExampleGrid*", within the HDF-EOS file, *Grid.he5*, referred to by the handle, *fid*:

```
gridID = HE5_GDattach(fid, "ExampleGrid");
```

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

FORTRAN integer function he5\_gdattach(*fid*, *gridname*)

integer *fid*

character\*(\*) *gridname*

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

```
gridid = he5_gdattach(fid, "ExampleGrid")
```

# Return Information about a Grid Attribute

---

## HE5\_GDattrinfo

herr\_t HE5\_GDattrinfo(hid\_t *gridID*, const char \**attrname*, hid\_t \* *ntype*, hsize\_t \**count*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*attrname* IN: Attribute name

*ntype* OUT: Number type of attribute

*count* OUT: Number of elements in attribute

Purpose Returns information about an object attribute in a specific grid. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

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

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

```
status = HE5_GDattrinfo(gridID, "ScalarFloat", &ntype, &count);
```

The *ntype* variable will have the value 10 and *count* will have the value 1.

FORTRAN integer function he5\_gdatrinfo(*gridid*, *attrname*, *ntype*, *count*,)

integer *gridid*

character\*(\*) *attrname*

integer *ntype*

integer\*4 *count*

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

```
status = he5_gdatrinfo(gridid, "ScalarFloat", ntype, count)
```

# Write Block SOM Offset

---

## HE5\_GDbkSOMoffset

herr\_t HE5\_GDbkSOMoffset(hid\_t *gridID*, long *offset[]*, hsize\_t *count*, char *\*code*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*offset* IN: Offset values for SOM Projection data

*count* IN: Number of offset values to write

*code* IN: Write/Read code

Purpose Write block SOM offset values.

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

Description The routine supports structures that contain data which have been written in the Solar Oblique Mercator (SOM) projection. The structure can contain one to many blocks, each with corner points defined by latitude and longitude. The routine can only be used by grids that use the SOM projection. The routine writes the offset values, in pixels, from a standard SOM projection. There is an offset value for every block in the grid except for the first block. The count parameter is used as a check for the number of offset values. This routine will also return the offset values, but the user must know how large the offset array needs to be before calling the function, in that case the code value would be "r" and the count parameter has to be provided also.

Example In this example, we first show how the SOM projection is defined using HE5\_GDdefproj, then we show how the SOM projection is modified using HE5\_GDbkSOMoffset:

The first parameter is the Grid ID, the second is the projection code for the SOM projection, the third is the zone code, not needed for the SOM projection, the fourth is the sphere code, not needed for the SOM projection and the last parameter is the projection parameter array. Each projection supported by the Grid interface has a unique set of variables that are used by the GCTP library and they are passed to the GCTP library through this array. As you can see below, the twelfth parameter is set to a non-zero value, it is set to the size of the number of blocks in the data field. This is required if the function HE5\_GDbkSOMoffset is going to be called. The GCTP library doesn't use this parameter for the SOM projection so that is used by the HDF-EOS library only. The HE5\_GDbkSOMoffset function checks that parameter first before anything else is done.

```

projparm[0] = 6378137.0;
projparm[1] = 0.006694348;
projparm[3] = HE5_EHconvAng(98.161, HE5_HDFE_DEG_DMS);
projparm[4] =
HE5_EHconvAng(87.11516945924, HE5_HDFE_DEG_DMS);
projparm[8] = 0.068585416 * 1440;
projparm[9] = 0.0;
projparm[11] = 6;

status = HE5_GDdefproj(GDdid_som, HE5_GCTP_SOM, NULL, NULL,
projparm);

```

Now that the projection has been defined, HE5\_GDblkSOMoffset can be called:

```

offset[5] = {5, 10, 12, 8, 2};
count = 5;
code = "w";

status = HE5_GDblkSOMoffset(gridID, offset, count, code);

```

This set the offset for the second block to 5 pixels, the third block to 10 pixels, fourth block to 12 pixels, fifth to 8 pixels and the sixth block to 2 pixels.

NOTE: This routine is currently implemented in "C" only. If the need arises, a FORTRAN function will be added.

Interblock subsetting is not currently supported by the ECS Science Data Server, at this time. That is, a response to a request to return data contained within a specified latitude/longitude box, will be in an integral number of blocks.

#### Related Documents

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

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

# Close an HDF-EOS File

---

## HE5\_GDclose

herr\_t HE5\_GDclose(hid\_t *fid*)

*fid* IN: Grid file ID returned by HE5\_GDopen

Purpose Closes file.

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

Description This routine closes the HDF-EOS grid file.

Example

```
status = HE5_GDclose(fid);
```

FORTRAN integer function he5\_gdclose(*fid*)

integer *fid*

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

```
status = he5_gdclose(fid)
```



# Retrieve Compression Information for Field

---

## HE5\_GDcompinfo

herr\_t HE5\_GDcompinfo(hid\_t *gridID*, const char \**fieldname*, int \**compcode*, int *compparm*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_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 or 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 *HE5\_GDdefcomp* section:

```
status = HE5_GDcompinfo(gridID, "Opacity", compcode,
compparm);
```

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

FORTRAN integer function he5\_gdcompinfo(*gridid*,*fieldname* *compcode*, *compparm*)

integer *gridid*

character\*(\*) *fieldname*

integer\*(\*) *compcode*

integer *compparm*(\*)

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

```
status = he5_gdcompinfo(gridid, 'Opacity', compcode,
compparm)
```

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

# Create a New Grid Structure

---

## HE5\_GDcreate

hid\_t HE5\_GDcreate(hid\_t *fid*, const char \**gridname*, long *xdimsize*, long *ydimsize*, double *upleftpt*[], double *lowrightpt*[])

*fid* IN: Grid file ID returned by HE5\_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*) or FAIL(-1) otherwise.

Description The grid is created as a group within the HDF-EOS file with the name *gridname*. 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

```
upleft[0]=210584.50041;  
upleft[1]=3322395.95445;  
lowrgt[0]=813931.10959;  
lowrgt[1]=2214162.53278;  
  
xdim=120;  
ydim=200;  
  
gridID = HE5_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 (DDDMMMSSS.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 id geographic (i.e., projcode=0) then the X-Y coordinates should be specified in degrees/minutes/seconds (DDDMMMSSS.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*

FORTTRAN integer function `he5_gdcreate(fid, gridname, xdimsize, ydimsize, upleftpt, lowrightpt)`

integer *fid*

character\*(\*) *gridname*

integer\*4 *xdimsize*

integer\*4 *ydimsize*

real\*8 *upleftpt(2)*

real\*8 *lowrightpt(2)*

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

```
gridid = he5_gdcreate(fid, "UTMGrid", xdim, ydim, uplft,  
                    lowrgt)
```

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

# Define Region of Interest by Latitude/Longitude

---

## HE5\_GDdefboxregion

hid\_t HE5\_GDdefboxregion(hid\_t *gridID*, double *cornerlon*[], double *cornerlat*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_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 or FAIL (-1) otherwise.

Description This routine defines a longitude-latitude box region as a subset region for a grid. It returns a grid region ID, used by the *HE5\_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 = HE5_GDdefboxregion(GDdid, cornerlon, cornerlat);
```

FORTRAN integer function he5\_gddefboxreg(*gridid*, *cornerlon*, *cornerlat*)

```
integer  gridid  
real*8  cornerlon(2)  
real*8  cornerlat(2)
```

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

```
cornerlon(1) = 0.  
cornerlat(1) = 90.  
cornerlon(2) = 90.  
cornerlat(2) = 0.  
regionid = he5_gddefboxreg(gridid, cornerlon, cornerlat)
```

# Set Grid Field Compression

---

## HE5\_GDdefcomp

herr\_t HE5\_GDdefcomp(hid\_t *gridID*, int *compcode*, int *compparm*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*compcode* IN: HDF compression code

*compparm* IN: Compression parameters (if applicable)

**Note: Shuffling, szip, and deflate compression are supported in this release.**

Purpose Sets the field compression for all subsequent field definitions.

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

Description This routine sets the HDF field compression for subsequent grid field definitions. The routine HE5\_GDdeftile() must be called first, otherwise HE5\_GDdefcomp doesn't work. The compression does not apply to one-dimensional fields. The compression schemes currently supported are: deflate (gzip) (HE5\_HDFE\_COMP\_DEFLATE=4), compression exactly as in hardware (HE5\_HDFE\_COMP\_SZIP\_CHIP = 5), allowing k split = 13 compression mode (HE5\_HDFE\_COMP\_SZIP\_K13 = 6), entropy coding method (HE5\_HDFE\_COMP\_SZIP\_EC = 7), nearest neighbor coding method (HE5\_HDFE\_COMP\_SZIP\_NN = 8), allowing k split = 13 compression mode or entropy coding method (HE5\_HDFE\_COMP\_SZIP\_K13orEC = 9), allowing k split = 13 compression mode or nearest neighbor coding method (HE5\_HDFE\_COMP\_SZIP\_K13orNN = 10), shuffling + deflate(gzip) (HE5\_HDFE\_COMP\_SHUF\_DEFLATE = 11), shuffling + compression exactly as in hardware (HE5\_HDFE\_COMP\_SHUF\_SZIP\_CHIP = 12), shuffling + allowing k split = 13 compression mode (HE5\_HDFE\_COMP\_SHUF\_SZIP\_K13 = 13), shuffling + entropy coding method (HE5\_HDFE\_COMP\_SHUF\_SZIP\_EC = 14), shuffling + nearest neighbor coding method (HE5\_HDFE\_COMP\_SHUF\_SZIP\_NN = 15), shuffling + allowing k split = 13 compression mode or entropy coding method (HE5\_HDFE\_COMP\_SHUF\_SZIP\_K13orEC = 16), shuffling + allowing k split = 13 compression mode or nearest neighbor coding method (HE5\_HDFE\_COMP\_SHUF\_SZIP\_K13orNN = 17), and no compression (HE5\_HDFE\_COMP\_NONE = 0, the default, with compparm[0] = 0). **Deflate compression requires a single integer compression parameter in the range of one to nine with higher values corresponding to greater compression. Szip compression requires one**

**parameter that is a pixels\_per\_block which must be even, with typical values being 8, 10, 16, 32. The more pixel values vary, the smaller this number should be.** Compressed fields are written using the standard *HE5\_GDwritefield* routine, however, the entire field must be written in a single call. If this is not possible, the user should consider tiling. See *HE5\_GDdefile* for further information. Any portion of a compressed field can then be accessed with the *HE5\_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* field using the entropy coding method, the *Opacity* field using the shuffling + deflate method, the *Spectra* field with deflate compression, and use no compression for the *Temperature* field.

```

compparm[0] = 16;

status = HE5_GDdefcomp(gridID, HE5_HDFE_COMP_SZIP_EC,
compparm);

status = HE5_GDdeffield(gridID, "Pressure",
"YDim,XDim",NULL, H5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE);

compparm[0] = 9;

status = HE5_GDdefcomp(gridID, HE5_HDFE_COMP_SHUF_DEFLATE,
compparm);

status = HE5_GDdeffield(gridID, "Opacity", "YDim,XDim",
NULL, H5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE);

status = HE5_GDdefcomp(gridID, HE5_HDFE_COMP_DEFLATE,
compparm);

status = HE5_GDdeffield(gridID, "Spectra", "Bands,YDim,XDim",
NULL, H5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE);

status = HE5_GDdefcomp(gridID, HE5_HDFE_COMP_NONE,
compparm);

status = HE5_GDdeffield(gridID, "Temperature", "YDim,XDim",
NULL, H5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE);

```

Note that the HE5\_HDFE\_NOMERGE parameter will be ignored in the field definitions.

FORTRAN integer function he5\_gddefcomp(*gridid*, *compcode*, *compparm*)

integer        *gridid*  
integer        *compcode*  
integer        *compparm*(\*)

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

```
parameter (HE5T_NATIVE_FLOAT=10)
parameter (HE5_HDFE_COMP_NONE=0)
parameter (HE5_HDFE_COMP_DEFLATE=4)
parameter (HE5_HDFE_COMP_SZIP_EC=7)
parameter (HE5_HDFE_COMP_SHUF_DEFLATE=11)
parameter (HE5_HDFE_NOMERGE = 0)

compparm(1) = 16
status = he5_gddefcomp(gridid, HE5_HDFE_COMP_SZIP_EC,
compparm)
status = he5_gddeffld(gridid, "Pressure", "YDim,XDim", " ",
HE5T_NATIVE_FLOAT,HE5_HDFE_NOMERGE)
compparm(1) = 9
status = he5_gddefcomp(gridid, HE5_HDFE_COMP_SHUF_DEFLATE,
compparm)
status = he5_gddeffld(gridid, "Opacity", "YDim,XDim",
" ", HE5T_NATIVE_FLOAT,HE5_HDFE_NOMERGE)
status = he5_gddefcomp(gridid, HE5_HDFE_COMP_DEFLATE,
compparm)
status = he5_gddeffld(gridid, "Spectra", "Bands,YDim,XDim",
" ", HE5T_NATIVE_FLOAT,HE5_HDFE_NOMERGE)
status = he5_gddefcomp(gridid, HE5_HDFE_COMP_NONE, compparm)
status = he5_gddeffld(gridid, "Temperature", "YDim,XDim", "
", HE5T_NATIVE_FLOAT,HE5_HDFE_NOMERGE)
```



# Define Compression with Data Tiling

---

## HE5\_GDdefcomtile

herr\_t HE5\_GDdefcomtile(hid\_t *gridID*, int *compcode*, int \**compparm*, int *tilerank*, const hsize\_t \**tiledim*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*compcode* IN: Compression method flag

*compparm* IN: Array of compression parameters

*tilerank* IN: Rank of a field to compress (a number other than zero)

*tiledim* IN: Array of sizes of tile (NULL cannot be used)

Purpose Compress the data field

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

Description This function allows the user to set compression for a data field with automatic tiling (see notes for HE5\_Gddeffield)

Example In this example, we set (DEFLATE) compression for a field that is defined right after this call

```
compcode = 4;
```

```
compparm[0] = 6;
```

```
status = HE5_GDdefcomtile(gridID, compcode, compparm,  
tilerank, tiledim);
```

FORTRAN integer function he5\_gddefcomtle(gridid, compcode, compparm, tilerank, tiledim)

integer *gridid*

integer *compcode*

integer *compparm*(\*)

integer *tilerank*

integer\*4 *tiledim*

The equivalent *FORTRAN* code for the example above is

```
status = he5_gddefcomtle(gridid, compcode, compparm,tilerank,tiledim)
```

## Define a New Dimension within a Grid

---

### HE5\_GDdefdim

herr\_t HE5\_GDdefdim(hid\_t *gridID*, char \**dimname*, hsize\_t *dim*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*dimname* IN: Name of dimension to be defined

*dim* IN: The size of the dimension

**Note: Merging is not supported in this release of the library. There are three illegal characters for field names: “/”, “;”, “,”**

Purpose Defines a new dimension within the grid.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reason for failure is an improper grid id.

Description This routine defines dimensions that are used by the field definition routines (described subsequently) to establish the size of the field.

Example In this example, we define a dimension, *Band*, with size 15.

```
status = HE5_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, *H5S\_UNLIMITED*:

```
status = HE5_GDdefdim(gridID, "Unlim", H5S_UNLIMITED);
```

FORTRAN integer function he5\_gddefdim(*gridid*, *fieldname*, *dim*)

integer *gridid*

character\*(\*) *fieldname*

integer\*4 *dim*

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

```
parameter (HE5S_UNLIMITED_F=-1)
```

```
dim = 15
```

```
status = he5_gddefdim(gridid, "Band", dim)
```

```
status = he5_gddefdim(gridid, "Unlim", HE5S_UNLIMITED_F)
```

# Define a New Data Field within a Grid

---

## HE5\_GDdeffield

herr\_t HE5\_GDdeffield(hid\_t *gridID*, const char \**fieldname*, char \**dimlist*, char \**maxdimlist* hid\_t *ntype*, int *merge*)

<i>gridID</i>	IN:	Grid ID returned by HE5_GDcreate or HE5_GDattach
<i>fieldname</i>	IN:	Name of field to be defined
<i>dimlist</i>	IN:	The list of data dimensions defining the field
<i>maxdimlist</i>	IN:	The maximum dimensions list defining the field
<i>ntype</i>	IN:	The number type of the data stored in the field
<i>merge</i>	IN:	Merge code (HE5_HDFE-NOMERGE (0) - no merge, HE5_HDFE_AUTOMERGE (1) -merge)

**Note: Merging is not supported in this release of the library. There are three illegal characters for field names: “/”, “;”, “,”. Also although use of Unlim dimension in *maxdimlist* is allowed, it may cause problem later if *xdim* or *ydim* of the data written to the field exceed *XDim* and *YDim* values for the grid.**

Purpose	Defines a new data field within the grid.
Return value	Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reason for failure is an unknown dimension in the dimension list.
Description	This routine defines data fields to be stored in the grid. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in <i>C</i> order, that is, the last dimension is incremented first.
Example	<p>In this example, we define a grid field, <i>Temperature</i> with dimensions <i>XDim</i> and <i>YDim</i> (as established by the <i>HE5_GDcreate</i> routine) containing 4-byte floating point numbers and a field, <i>Spectra</i>, with dimensions <i>XDim</i>, <i>YDim</i>, and <i>Bands</i>:</p> <pre>status = HE5_GDdeffield(gridID, "Temperature", "YDim,XDim", NULL, H5T_NATIVE_FLOAT, 0);  status = HE5_GDdeffield(gridID, "Spectra", "Bands,YDim,XDim", NULL, H5T_NATIVE_FLOAT, 0);</pre>
FORTRAN	integer function he5_gddeffld( <i>gridid</i> , <i>fieldname</i> , <i>dimlist</i> , <i>maxdimlist</i> , <i>ntype</i> , <i>merge</i> )

integer        *gridid*  
character\*(\*) *fieldname*  
character\*(\*) *dimlist*  
character\*(\*) *maxdimlist*  
integer        *ntype, merge*

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

```
parameter (HE5T_NATIVE_FLOAT=10)
parameter (HE5_HDFE_NOMERGE=0)

status = he5_gddeffld(gridid, "Temperature", "XDim,YDim", "
", HE5T_NATIVE_FLOAT,HE5_HDFE_NOMERGE)

status = he5_gddeffld(gridid, "Spectra", "XDim,YDim,Bands",
" ", HE5T_NATIVE_FLOAT,HE5_HDFE_NOMERGE)
```

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

**Note:**

**User must call HE5\_GDdefcomtile, or alternatively HE5\_GDdeftile followed by HE5\_GDdefcomp, before calling HE5\_GDdefield in order to be able to internally compress the defined field. If after this is done user desires to define another field that is not compressed and not tiled, user must call HE5\_GDdefcomtile() again as**

```
compcode = 0;
compparm[0] = 0;

status = HE5_GDdefcomtile(gridID, compcode, compparm,
tilerank, tiledims);
```

**or alternatively call**

```
status = HE5_GDdeftile(gridID, HE5_HDFE_NOTILE, tilerank,
tiledims);

status = HE5_GDdefcomp(gridID, HE5_HDFE_COMP_NONE,
compparm);
```

**where tilerank and tiledims must be the same as the rank and dims, respectively, for the field to be defined. Please note that 1-D fields cannot be compressed. So if user has already used GDdefcomtile (or HE5\_GDdeftile, HE5\_GDdefcomp combination) to define compression, user must follow the steps above for setting no-tiling and no-compression. Please also note that with this release user cannot use tilerank =0 and NULL for tiledims.**

## Define the Origin of the Grid Data

---

### HE5\_GDdeforigin

herr\_t HE5\_GDdeforigin(hid\_t *gridID*, int *origincode*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

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

Purpose Defines the origin of the grid pixel data

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

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

Origin Codes:

HE5\_HDFE\_GD\_UL(Default) (0) Upper Left corner of grid

HE5\_HDFE\_GD\_UR (1) Upper Right corner of grid

HE5\_HDFE\_GD\_LL (2) Lower Left corner of grid

HE5\_HDFE\_GD\_LR (3) Lower Right corner of grid

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

```
status = HE5_GDdeforigin(gridID, HE5_HDFE_GD_LR);
```

FORTRAN integer function he5\_gddeforg(*gridid*, *origincode*)

integer *gridid*

integer *origincode*

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

```
parameter (HE5_HDFE_GD_LR=3)
```

```
status = he5_gddeforg(gridid, HE5_HDFE_GD_LR)
```

# Define a Pixel Registration within a Grid

---

## HE5\_GDdefpixreg

herr\_t HE5\_GDdefpixreg(hid\_t *gridID*, int *pixregcode*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*pixregcode* IN: Pixel registration code

Purpose Defines pixel registration within grid cell

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

Description This routine is used to define whether the pixel center or pixel corner (as defined by the *HE5\_GDdeforigin* routine) is used when requesting the location (longitude and latitude) of a given pixel.

Registration Codes:

HE5\_HDFE\_CENTER (0) (Default) Center of pixel cell

HE5\_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 = HE5_GDdefpixreg(gridID, HE5_HDFE_CORNER);
```

FORTRAN integer function he5\_gddefpixreg(*gridid*, *pixregcode*)

```
integer      gridid
```

```
integer      pixregcode
```

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

```
parameter (HE5_HDFE_CORNER=1)
```

```
status = he5_gddefpixreg(gridid, HE5_HDFE_CORNER)
```

# Define Grid Projection

---

## HE5\_GDdefproj

herr\_t HE5\_GDdefproj(hid\_t *gridID*, int *projcode*, int *zonecode*, int *spherecode*, double *projparm*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_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 or FAIL(-1) otherwise

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

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

```
spherecode = 0;
```

```
zonecode = 40;
```

```
status = HE5_GDdefproj(gridID, HE5_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 = HE5_GDdefproj(gridID, HE5_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 = HE5_GDdefproj(gridID, HE5_GCTP_GEO, NULL, NULL,  
NULL)
```

FORTRAN integer function he5\_gddefproj(gridid, projcode, zonecode, spherecode,  
projparm)

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

The equivalent FORTRAN code for the examples above is:

```
parameter (HE5_GCTP_UTM=1)  
  
spherecode = 0  
zonecode = 40  
  
status = he5_gddefproj(gridid, HE5_GCTP_UTM, zonecode,  
spherecode, dummy)  
  
parameter (HE5_GCTP_PS=6)  
  
spherecode = 6  
  
do i=1,13  
    projparm(i) = 0  
  
enddo  
  
projparm(6) = 90000000.00  
  
status = he5_gddefproj(gridid, HE5_GCTP_PS, dummy,  
spherecode, projparm)  
  
parameter (GCTP_GEO=0)  
  
status = he5_gddefproj(gridid, HE5_GCTP_GEO, dummy, dummy,  
dummy)
```

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



## Define Tiling Parameters

---

### HE5\_GDdeftile

herr\_t HE5\_GDdeftile(hid\_t *gridID*, int *tilecode*, int *tilerank*, const hsize\_t *tiledims*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*tilecode* IN: Tile code: HE5\_HDF\_TILE, HE5\_HDF\_NOTILE (default)  
*tilerank* IN: The number of tile dimensions (a number other than zero)  
*tiledims* IN: Tile dimensions (NULL cannot be used)

Purpose Defines tiling dimensions for subsequent field definitions

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

Description This routine defines the tiling dimensions for fields defined following this function call, analogous to the procedure for setting the field compression scheme using *HE5\_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 = HE5_GDdeftile(gridID, HE5_HDFE_TILE, 2, tiledims);
status = HE5_GDdeffield(gridID, "Pressure", "YDim,XDim",
NULL, H5T_NATIVE_INT, 0);
status = HE5_GDdeffield(gridID, "Temperature", "YDim,XDim",
NULL, H5T_NATIVE_FLOAT, 0);
tiledims[0] = 1;
tiledims[1] = 150;
tiledims[2] = 100;
status = HE5_GDdeftile(gridID, HE5_HDFE_TILE, 3, tiledims);
```

```

status = HE5_GDdefield(gridID, "Spectra",
" Bands,YDim,XDim", NULL, H5T_NATIVE_FLOAT,
HE5_HDFE_NOMERGE);

tiledims[0] = ydim;

tiledims[1] = xdim;

status = HE5_GDdeftile(gridID, HE5_HDFE_NOTILE, 2,
tiledims);

status = HE5_GDdefield(gridID, "Communities", "YDim,XDim",
NULL, H5T_NATIVE_INT, HE5_HDFE_AUTOMERGE);

```

FORTRAN integer function *he5\_gddeftle(gridid, tilecode,tilerank,tiledims)*

```

integer      gridid
integer      tilecode
integer      tilerank
integer*4    tiledims(*)

```

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

```

parameter (HE5T_NATIVE_INT=0)
parameter (HE5T_NATIVE_FLOAT=10)
parameter (HE5_HDFE_NOTILE=0)
parameter (HE5_HDFE_TILE=1)
parameter (HE5_HDFE_NOMERGE = 0)

tiledims(1) = 200

tiledims(2) = 100

tilerank    = 2

status = he5_gddeftle(gridid, HE5_HDFE_TILE,tilerank,
tiledims)

status = he5_gddeffld(gridid, 'Pressure', 'XDim,YDim', " ",
HE5T_NATIVE_INT, HE5_HDFE_NOMERGE)

status = he5_gddeffld(gridid, 'Temperature', 'XDim,YDim',
" ", HE5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE)

tiledims(1) = 100

tiledims(2) = 150

tiledims(30 = 1

```

```
tilerank      = 3

status = he5_gddefltle(gridid, HE5_HDFE_TILE, tilerank,
tiledims)

status = he5_gddeffld(gridid, `Spectra`, `XDim,YDim,Bands`,
" ", HE5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE)

tilerank      = 2

tiledims(1) = xdim
tiledims(2) = ydim

status = he5_gddefltle(gridid, HE5_HDFE_NOTILE, tilerank,
tiledims);

status = he5_gddeffld(gridid, `Communities`, `XDim,YDim`, "
", HE5T_NATIVE_INT, HE5_HDFE_AUTOMERGE)
```

## Define a Time Period of interest

---

### HE5\_GDdeftimeperiod

herr\_t HE5\_GDdeftimeperiod(hid\_t *gridID*, hid\_t *periodID*, double *starttime*, double *stoptime*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_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 or FAIL (-1) otherwise.

Description This routine defines a time period for a grid. It returns a grid period ID which is used by the *HE5\_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 *HE5\_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 *HE5\_GDdefvrtregion* to further refine a region. This routine may also be called “stand-alone” by setting the input id to HE5\_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 = HE5_GDdeftimeperiod(gridID, HE5_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 = HE5_GDdeftimeperiod(gridID, regionID, starttime,  
stoptime);
```

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

FORTRAN integer function `he5_gddefmeper(gridid, periodID, starttime, stoptime)`  
integer *gridid*  
integer *periodid*  
real\*8 *starttime*  
real\*8 *stoptime*

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

```
parameter (HE5_HDFE_NOPREVSUB=-1)
starttime = 35232487.2
stoptime = 36609898.1
periodid = he5_gddefmeper(gridid, HE5_HDFE_NOPREVSUB,
starttime, stoptime)
periodid = he5_gddefmeper(gridid, regionid, starttime,
stoptime)
```

## Define a Vertical Subset Region

---

### HE5\_GDdefvrtregion

hid\_t HE5\_GDdefvrtregion(hid\_t *gridID*, hid\_t *regionID*, char \**vertObj*, double *range*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_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 or FAIL (-1) otherwise.

Description Whereas the *HE5\_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: INT, LONG, FLOAT, DOUBLE.) This routine may be called after *HE5\_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 HE5\_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 *HE5\_GDregioninfo* and *HE5\_GDextractregion* routines work as before, however the field to be subsetted, (the field specified in the call to *HE5\_GDregioninfo* and *HE5\_GDextractregion*) must contain the dimension used explicitly in the call to *HE5\_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 = HE5_GDdefvrtregion(gridID, HE5_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 = HE5_GDdefvrtregion(gridID, HE5_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 = HE5_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 function he5_gddefvrtreg(gridid, regionid, vertobj, range)
integer      gridid
integer      regionid
character*(*) vertobj
real*8      range(2)

```

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

```

parameter (HE5_HDFE_NOPREVSUB=-1)
range(1) = 500.
range(2) = 800.
regionid = he5_gddefvrtreg(gridid, HE5_HDFE_NOPREVSUB,
"Pressure", range)
range(1) = 3      ! Note 1-based element numbers
range(2) = 6
regionid = he5_gddefvrtreg(gridid, HE5_HDFE_NOPREVSUB,
"DIM:Height", range)
regionid = he5_gddefvrtreg(gridid, subsetid, "Pressure",
range)

```

# Detach from Grid Structure

---

## HE5\_GDdetach

herr\_t HE5\_GDdetach(hid\_t *gridID*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

Purpose Detaches from grid interface.

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

Description This routine should be run before exiting from the grid file for every grid opened by *HE5\_GDcreate* or *HE5\_GDattach*.

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

```
status = HE5_GDdetach(gridID);
```

FORTRAN integer function he5\_gddetach(gridid)

```
integer      gridid
```

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

```
status = he5_gddetach(gridid)
```



## Retrieve Size of Specified Dimension

---

### HE5\_GDdiminfo

hsize\_t HE5\_GDdiminfo(hid\_t *gridID*, char \**dimname*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*dimname* IN: Dimension name

Purpose Retrieve size of specified dimension.

Return value Size of dimension if successful or 0 otherwise. A typical reason for failure is an improper grid id or dimension name.

Description This routine retrieves the size of specified dimension.

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

```
dimsize = HE5_GDdiminfo(gridID, "Bands");
```

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

FORTRAN integer\*4 function he5\_gddiminfo(*gridid*,*dimname*)

```
integer      gridid
```

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

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

```
dimsize = he5_gddiminfo(gridid, "Bands")
```

# Remove an Alias for Grid Data Field

---

## HE5\_GDdropalias

herr\_t HE5\_GDdropalias(hid\_t *gridID*, int *fldgroup*, const char \**aliasname*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*fldgroup* IN: Field group flag

*aliasname* IN: Name of alias to remove

Purpose Remove an alias for Grid data field

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

Description Removes alias associated with a Grid data field.

Example In this example, we remove an alias for the data field *Temperature*.

```
strcpy(aliasname, "temps 0 to 30");
```

```
status = HE5_GDdropalias(gridID, HE5_HDFE_DATAGROUP,  
aliasname);
```

FORTRAN integer function he5\_gddropalias (*gridid*, *fldgroup*, *aliasname*)

integer *gridid*

character\*(\*) *fldgroup*

character\*(\*) *aliasname*

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

```
aliasname = "temps 0 to 30"
```

```
status = he5_grdropalias(gridid, HE5_HDFE_DATAGROUP,  
aliasname)
```

# Duplicate a Region or Period

---

## HE5\_GDdupregion

hid\_t HE5\_GDdupregion(hid\_t *oldregionID*)

*oldregionID* IN: Region or period ID returned by HE5\_GDdefboxregion, HE5\_GDdeftimeperiod, or HE5\_GDdefvrtregion.

Purpose Duplicates a region.

Return value Returns new region or period ID if successful or FAIL (-1) otherwise.

Description This routine copies the information stored in a current region or period to a new region or period and generates a new id. It is usefully when the user wishes to further subset a region (period) in multiple ways.

Example In this example, we first subset a grid with *HE5\_GDdefboxregion*, duplicate the region creating a new region ID, *regionID2*, and then perform two different vertical subsets of these (identical) geographic subset regions:

```
regionID = HE5_GDdefboxregion(gridID, cornerlon, cornerlat);
regionID2 = HE5_GDdupregion(regionID);

regionID = HE5_GDdefvrtregion(gridID, regionID, "Pressure",
rangePres);

regionID2 = HE5_GDdefvrtregion(gridID, regionID2,
"Temperature", rangeTemp);
```

FORTRAN integer he5\_gddupreg(*oldregionid*)

integer *oldregionid*

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

```
regionid = he5_gddefboxreg(gridid, cornerlon, cornerlat)
regionid2 = he5_gddupreg(regionid)

regionid = he5_gddefvrtreg(gridid, regionid, 'Pressure',
rangePres)

regionid2 = he5_gddefvrtreg(gridid, regionid2,
'Temperature', rangeTemp)
```

# Read a Region of interest from a Field

---

## HE5\_GDextractregion

herr\_t HE5\_GDextractregion(hid\_t *gridID*, hid\_t *regionID*, const char \**fieldname*, void \**buffer*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*regionID* IN: Region (period) ID returned by HE5\_GDdefboxregion (HE5\_GDdeftimeperiod)

*fieldname* IN: Field to subset

*buffer* OUT: Data Buffer

Purpose Extracts (reads) from subsetted region.

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

Description This routine reads data into the data buffer from a subsetted region as defined by *HE5\_GDdefboxregion*.

Example In this example, we extract data from the *Temperature* field from the region defined in *HE5\_GDdefboxregion*. We first allocate space for the data buffer. The size of the subsetted region for the field is given by the HE5\_GDregioninfo routine.

```
datbuf = (float *)calloc(size, sizeof(float));
status = HE5_GDextractregion(GDdid, regionID, "Temperature",
datbuf);
```

FORTRAN integer function he5\_gdextreg(*gridid*, *regionid*, *fieldname*, *datbuf*)

integer *gridid*

integer *regionid*

character\*(\*) *fieldname*

<valid type> *buffer*(\*)

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

```
status = he5_gdextreg(gridid, regionid, "Temperature",
datbuf)
```

# Retrieve Information about Data Field in a Grid

---

## HE5\_GDfieldinfo

herr\_t HE5\_GDfieldinfo(hid\_t *gridID*, const char \**fieldname*, int \**rank*, hsize\_t *dims*[], hid\_t *ntype*[], char \**dimlist*, char \**maxdimlist*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*fieldname* IN: Fieldname  
*rank* OUT: Pointer to rank of the field  
*dims* OUT: Array containing the dimension sizes of the field  
*ntype* OUT: Pointer to the numbertype of the field. See Appendix A for interpretation of number types.  
*dimlist* OUT: Dimension list  
*maxdimlist* OUT: Maximum dimensions allowed for field

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

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise. A typical reason for failure is the specified field does not exist.

Description This routine retrieves information on a specific data field.

Example In this example, we retrieve information about the *Spectra* data fields:

```
status = HE5_GDfieldinfo(gridID, "Spectra", &rank, dims,  
                        &ntype, dimlist, maxdimlist);
```

The return parameters will have the following values:

*rank*=3, *ntype*=10, *dims*[3]={15,200,120} and

*dimlist*="Bands,YDim,XDim"

FORTRAN integer function he5\_gdfldinfo(*gridid*, *fieldname*, *rank*, *dims*, *ntype*,  
*dimlist*, *maxdimlist*)

integer *gridid*

character\*(\*) *fieldname*

integer(\*) *rank*

integer\*4     *dims*(\*)  
integer        *ntype*(\*)  
character\*(\*) *dimlist*  
character\*(\*) *maxdimlist*

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

```
status = he5_gdfldinfo(gridid, "Spectra", dims, rank, ntype,  
dimlist, maxdimlist)
```

The return parameters will have the following values:

*rank*=3, *ntype*=10, *dims*[3]={120,200,15} and

*dimlist*="XDim,YDim,Bands"

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

# Retrieve Alias List for a Grid Data Fields Group

---

## HE5\_GDgetaliaslist

long HE5\_GDgetaliaslist(hid\_t *gridID*, int *fldgroup*, char \**aliaslist*, long \**strbufsize*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*fldgroup* IN: Field group flag for "Data Fields" group  
*aliaslist* OUT: List of alias(es) in the "Data Fields" group (comma separated list)  
*strbufsize* OUT: Length of aliases list  
Purpose To retrieve the number and list of aliases in a grid.  
Return value Returns number of aliases in "Data Fields" group if successful or returns FAIL (-1) otherwise.  
Description Retrieves list of aliases in the "Data Fields" group (comma separated list) of a Grid and returns their number. The Data group flag is **HE5\_HDFE\_DATAGROUP**.

Example In this example, we get the alias list for the "Data Fields" group of a grid.

```
/* first get the size of the list in bytes */  
  
nalias = HE5_GDgetaliaslist(gridID, HE5_HDFE_DATAGROUP,  
NULL, strbufsize);  
  
aliaslist = (char *)malloc(strbufsize *sizeof(char));  
  
nalias = HE5_GDgetaliaslist(gridID, HE5_HDFE_DATAGROUP,  
aliaslist, strbufsize);
```

FORTRAN integer function he5\_gdgetaliaslist (*gridid*, *fldgroup*, *aliaslist*, *strbufsize*)  
integer *gridid*  
integer *fldgroup*  
integer *strbufsize*  
character\*(\*) *aliaslist*

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

```
integer nalias  
  
nalias = he5_gdgetaliaslist(gridid, HE5_HDFE_DATAGROUP,  
aliaslist, strbufsize)
```

# Get External Data File Information

---

## HE5\_GDgetextdata

int HE5\_GDgetextdata(hid\_t *gridID*, char \**fieldname*, size\_t *namelength*, char \**filelist*, off\_t *offset[]*, hsize\_t *size[]*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*fieldname* IN: External field name  
*namelength* OUT: Length of each name entry  
*filelist* OUT: List of file names  
*offset[]* OUT: Array of offsets (in byte) from the beginning of file to the location in file where the data starts  
*size[]* OUT: Array of sizes (in bytes) reserved in the file for the data

Purpose Retrieves information about external data file(s) associated with the data set.

Return value Returns number of external data files if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper grid ID or field name.

Example In this example, we get information about the *ExtData* field:

```
nfiles = HE5_GDgetextdata(gridID, "ExtData", namlen,  
filenames, offset, size);
```

FORTTRAN integer function he5\_gdgetxdat(*gridid*,*fieldname*,*nlen*, *flist*,*offset*, *size*)

```
integer      gridid  
integer      nfiles  
integer*4    nlen  
integer*4    offset(*)  
integer*4    size(*)  
character*(*) fieldname  
character*(*) flist
```

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

```
nfiles = he5_gdgetxdat(gridid, "ExtData", nlen,  
flist, offset, size)
```



## Get Fill Value for Specified Field

---

### HE5\_GDgetfillvalue

herr\_t HE5\_GDgetfillvalue(hid\_t *gridID*, const char \**fieldname*, void \**fillvalue*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_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 or FAIL(-1) otherwise. Typical reasons for failure are an improper Grid ID or number type or incorrect fill value.

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

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

```
status = HE5_GDgetfillvalue(gridID, "Temperature",
&tempfill);
```

FORTTRAN integer function he5\_gdgetfill(*gridid*,*fieldname*,*fillvalue*)

integer *gridid*

character\*(\*) *fieldname*

<valid type> *fillvalue*(\*)

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

```
status = he5_gdgetfill(gridid, "Temperature", tempfill)
```

# Get Row/Columns for Specified Longitude/Latitude Pairs

---

## HE5\_GDgetpixels

herr\_t HE5\_GDgetpixels(hid\_t *gridID*, long *nLonLat*, double *lonVal[]*, double *latVal[]*, long *pixRow[]*, long *pixCol[]*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*nLonLat* IN: Number of longitude/latitude pairs

*lonVal* IN: Longitude values in degrees

*latVal* IN: Latitude values in degrees

*pixRow* 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 or FAIL(-1) otherwise.

Description This routine converts longitude/latitude pairs into (0 - based) pixel rows and columns. The origin is the upper left-hand corner of the grid. This routine is the pixel subsetting equivalent of *HE5\_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 = HE5_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 `he5_gdgetpix(gridid, nlonlat, lonval, latval, pixrow, pixcol)`

integer *gridid*

integer\*4 *nlonlat*

real\*8 *lonval*(\*)

real\*8 *latval*(\*)

integer\*4 *pixrow*(\*)

integer\*4 *pixcol*(\*)

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

```
lonarr(1) = 134.2
```

```
latarr(1) = -20.8
```

```
lonarr(2) = 15.8
```

```
latarr(2) = 84.6
```

```
nlonlat = 2
```

```
status = he5_gdgetpix(gridid, nlonlat, lonarr, latarr,  
rowarr, colarr)
```

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

# Get Field Values for Specified Row/Columns

---

## HE5\_GDgetpixvalues

long HE5\_GDgetpixvalues(hid\_t *gridID*, long *nPixels*, long *pixRow[]*, long *pixCol[]*, const char *\*fieldname*, void *\*buffer*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*nPixels* IN: Number of pixels

*pixRow* 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 or FAIL(-1) otherwise.

Description This routine reads data from a data field for the specified pixels. It is the pixel subsetting equivalent of *HE5\_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. In the first call, set the parameter *datbuf* to NULL:

```
double      *datbuf;

bufsiz = HE5_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 = (double *)calloc(bufsiz, sizeof(double));

bufsiz = HE5_GDgetpixvalues(gridID, 2, rowArr, colArr,
    "Spectra", datbuf);
```

FORTRAN integer\*4 function he5\_gdgetpixval(*gridid*, *npixels*, *pixrow*, *pixcol*,  
*fieldname*, *buffer*)

integer *gridid*

integer\*4 *npixels*

integer\*4 *bufsiz*

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)
```

```
npixels = 2
```

```
bufsiz = he5_gdgetpixval(gridid, npixels, rowarr, colarr,  
"Spectra", datbuf)
```

## Return Information about a Grid Structure

---

### HE5\_GDgridinfo

herr\_t HE5\_GDgridinfo(hid\_t *gridID*, long \**xdimsize*, long \**ydimsize*, double *upleftpt*[], double *lowrightpt*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*xdimsize* OUT: Number of columns in grid

*ydimsize* OUT: Number of rows in grid

*upleftpt* OUT: Location, in meters, of upper left corner

*lowrightpt* OUT: Location, in meters, of lower right corner

Purpose Returns position and size of grid

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

Description This routine returns the number of rows, columns and the location, in meters, of the upper left and lower right corners of the grid image.

Example In this example, we retrieve information from a previously created grid with a call to *HE5\_GDattach*:

```
status = HE5_GDgridinfo(gridID, &xdimsize, &ydimsize,
upleft, lowrgt);
```

FORTRAN integer function he5\_gdgridinfo(*gridid*, *xdimsize*, *ydimsize*, *upleftpt*, *lowrightpt*)

integer *gridid*

integer\*4 *xdimsize*

integer\*4 *ydimsize*

real\*8 *upleftpt*(2)

real\*8 *lowrightpt*(2)

The equivalent FORTRAN code for the example above is:

```
status = he5_gdgridinfo(gridid, xdimsize, ydimsize, upleft,
lowrgt)
```

# Return Information about a Group Grid Attribute

---

## HE5\_GDgrpattrinfo

herr\_t HE5\_GDgrpattrinfo(hid\_t *gridID*, const char \**attrname*, hid\_t \**ntype*, hsize\_t \**count*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*attrname* IN: Attribute name

*ntype* OUT: Number type of attribute

*count* OUT: Number of attribute elements

**Purpose** Returns information about a group attribute in the “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

**Description** This routine returns number type and number of elements (*count*) of a group attribute in the “Data Fields” group.

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

```
status = HE5_GDgrpattrinfo(gridID, "ScalarFloat", &ntype, &count);
```

The *ntype* variable will have the value 10 and *count* will have the value 1.

**FORTTRAN**

```
integer function he5_gdgetattrinfo(gridid, attrname, ntype, count,)  
integer      gridid  
character*(*) attrname  
integer      ntype  
integer *4    count
```

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

```
status = he5_gdgetattrinfo(gridid, "ScalarFloat", ntype,  
count)
```

# Retrieve Information about Grid Attributes

---

## HE5\_GDinqattrs

long HE5\_GDinqattrs(hid\_t *gridID*, char \**attrnames*, long \**strbufsize*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*attrnames* OUT: Attribute list (entries separated by commas)  
*strbufsize* OUT: String length of attribute list  
Purpose Retrieve information about object attributes defined in a specific grid. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each attribute name separated by commas. If *attrnames* 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. In the first call, set the parameter *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_GDinqattrs(gridID, NULL, strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_GDinqattrs(gridID, attrnames, strbufsize);
```

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

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

FORTTRAN integer\*4 function he5\_gdinqattrs(*gridid*,*attrnames*,*strbufsize*)

```
integer      gridid
```

```
character*(*) attrnames
```

```
integer*4    strbufsize
```

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

```
nattr = he5_gdinqattrs(gridid, attrnames, strbufsize)
```



# Return Data Type Information about Data Fields in Grid

---

## HE5\_GDinqdatatype

herr\_t HE5\_GDinqdatatype(hid\_t *gridID*, const char \**fieldname*, const char \**attrname*, int *fieldgroup*, hid\_t \**datatype*, H5T\_class\_t \**classid*, H5T\_order\_t \**order*, size\_t \**size*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*fieldname* IN: Field name

*attrname* IN: Attribute name

*fieldgroup* IN: Field group flag: HE5\_HDFE\_DATAGROUP - 1  
HE5\_HDFE\_ATTRGROUP - 2  
HE5\_HDFE\_GRPATTRGROUP - 3  
HE5\_HDFE\_LOCATTRGROUP - 4

*datatype* OUT: Data type ID

*classID* OUT: Data type class ID

*order* OUT: Data type byte order

*size* OUT: Data type size (in bytes)

Purpose Returns data type information about a specified field in grid.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper grid ID or field name.

Description This routine returns information about field data in a grid.

Example In this example we return the data type information for the *Spectra* field in the grid defined in the *HE5\_GDdeffield* routine.

```
status = HE5_GDinqdatatype(gridID, "Spectra", NULL,  
fieldgroup, &datatype, &classid, &order, &size);
```

FORTTRAN integer function  
he5\_gdinqdatatype(*gridid*,*fieldname*,*attrname*,*fldgrp*,*dtype*,*classid*,*order*,  
*size*)

integer *gridid*

integer        *dtype,classid,order*

integer\*4     *size*

character \*(\*) *fieldname*

integer        *HE5\_HDFE\_DATAGROUP*

parameter     (*HE5\_HDFE\_DATAGROUP=1*)

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

```
status = he5_gdingdatatype(gridid, "Spectra", " ",  
HE5_HDFE_DATAGROUP, dtype, classid, order, size)
```

# Retrieve Information about Dimensions Defined in Grid

---

## HE5\_GDinqdims

int HE5\_GDinqdims(hid\_t *gridID*, char \**dimnames*, hsize\_t *dims*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*dimnames* 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 successful or FAIL(-1) otherwise. A typical reason for failure is an improper grid id.

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

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

```
ndim = HE5_GDinqdims(gridID, dimnames, dims);
```

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

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

FORTRAN integer function he5\_gdinqdims(*gridid*,*dimnames*,*dims*)

integer *gridid*

character\*(\*) *dimnames*

integer\*4 *dims*(\*)

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

```
ndim = he5_gdinqdims(gridid, dimnames, dims)
```

# Retrieve Information about Data Fields Defined in Grid

---

## HE5\_GDinqfields

int HE5\_GDinqfields(hid\_t *gridID*, char \**fieldlist*, int *rank*[], hid\_t *nstype*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_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. See Appendix A for interpretation of number types.

Purpose Retrieve information about the data fields defined in grid.

Return value Number of data fields found if successful or FAIL(-1) otherwise. A typical reason is an improper grid id.

Description The field list is returned as a string with each data field separated by commas. The *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 = HE5_GDinqfields(gridID, fieldlist, rank, numbertype);
```

  
The parameter, *fieldlist*, will have the value: "*Temperature,Spectra*"  
with *rank*[2]={2,3}, *numbertype*[2]={10,10}

FORTTRAN integer function he5\_gdinqflds(*gridid*, *fieldlist*, *rank*, *numbertype*)  
integer *gridid*  
character\*(\*) *fieldlist*  
integer *rank*(\*)  
integer *numbertype*(\*)

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

```
nfld = he5_gdinqflds(gridID, fieldlist, rank, numbertype)
```

  
The parameter, *fieldlist*, will have the value: "*Spectra,Temperature*"  
with *rank*[2]={3,2}, *numbertype*[2]={10,10}

# Retrieve Information about Data Fields and Aliases Defined in Grid

---

## HE5\_GDinqfldalias

long HE5\_GDinqfldalias(hid\_t *gridID*, char \**fldalias*, long \**strbufsize*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*fldalias* OUT: List of data fields and aliases (entries separated by commas)  
*strbufsize* OUT: String length of data fields and aliases list  
Purpose Retrieve information about data fields & aliases defined in grid.  
Return value Number of data fields and aliases found if successful or FAIL (-1) otherwise.  
Description The list of data fields and aliases is returned as a string with each name separated by commas. If *fldalias* 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 data fields and aliases defined for the “Data Fields” group. In the first call, set the parameter *fldalias* to NULL. We assume that there are one data field and one alias stored, *Temperature* and *Temp*:

```
nfldalias = HE5_GDinqfldalias(gridID, NULL, &strbufsize);
```

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

```
fldalias = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nfldalias = HE5_GDinqfldalias(gridID, fldalias,  
&strbufsize);
```

The variable, *fldalias*, will be set to: "*Temperature,Temp*".

FORTRAN integer\*4 function he5\_gdinqfldalias(*gridid*, *fldalias*, *strbufsize*)

```
integer      gridid  
integer *4   strbufsize  
integer *4   nfldalias  
character*(*) fldalias
```

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

```
nfldalias = he5_gdinqfldalias(gridid, fldalias, strbufsize)
```

# Retrieve Grid Structures Defined in HDF-EOS File

---

## HE5\_GDinqgrid

long HE5\_GDinqgrid(const char \* *filename*, char \**gridlist*, long \**strbufsize*)

*filename* IN: HDF-EOS file name

*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 of successful or FAIL (-1) otherwise.

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, *Grid.he5*. In the first call, set the parameter *gridlist* to NULL. We assume that there are two grids stored, *GridOne* and *Grid\_2*:

```
ngrid = HE5_GDinqgrid("Grid.he5", NULL, strbufsize);
```

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

```
gridlist = (char *)calloc(strbufsize+1, sizeof(char));
```

```
ngrid = HE5_GDinqgrid("Grid.he5", gridlist, strbufsize);
```

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

“*GridOne,Grid\_2*”.

FORTRAN integer\*4 function he5\_gdinqgrid(*filename,gridlist,strbufsize*)

character\*(\*) *filename*

character\*(\*) *gridlist*

integer\*4 *strbufsize*

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

```
ngrid = he5_gdinqgrid('Grid.he5', gridlist, strbufsize)
```

# Retrieve Information Grid Group Attributes

---

## HE5\_GDinqgrpattrs

long HE5\_GDinqgrpattrs(hid\_t *gridID*, char \**attrnames*, long \**strbufsize*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*attrnames* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

Purpose Retrieve information about group attributes defined in the “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each group 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 group attributes defined for the “Data Fields” group. In the first call, set *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_GDinqgrpattrs(gridID, NULL, &strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_GDinqgrpattrs(gridID, attrnames, &strbufsize);
```

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

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

FORTRAN integer\*4 function he5\_gdinqgattr(*gridid*, *attrnames*, *strbufsize*)

integer *gridid*

character\*(\*) *attrnames*

integer\*4 *strbufsize*

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

```
nattr = he5_gdinqgattr(gridid, attrnames, strbufsize)
```

## Retrieve Information Grid Local Attributes

---

### HE5\_GDinqlocattrs

long HE5\_GDinqlocattrs(hid\_t *gridID*, const char \**fieldname*, char \**attrnames*, long \**strbufsize*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*fieldname* IN: Fieldname to retrieve local attribute information

*attrnames* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

Purpose Retrieve information about local attributes defined for a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each local attribute name separated by commas. If *attrnames* 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 local attributes defined for a field "DataField". In the first call, set *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_GDinqlocattrs(gridID, "DataField", NULL,
&strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_GDinqlocattrs(gridID, "DataField", attrnames,
&strbufsize);
```

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

"*attrOne,attr\_2*".

FORTRAN integer\*4 function he5\_gdinqlattr(*gridid*, *fieldname*, *attrnames*, *strbufsize*)

integer *gridid*

character\*(\*) *fieldname*



character\*(\*) *attrnames*

integer\*4 *strbufsize*

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

```
nattr = he5_gdinqlattr(gridid, "DataField", attrnames,  
strbufsize)
```

# Perform Bilinear Interpolation on Grid Field

---

## HE5\_GDinterpolate

long HE5\_GDinterpolate(hid\_t *gridID*, long *nValues*, double *lonVal*[], double *latVal*[], const char \**fieldname*, double *interpVal*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*nValues* 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 or 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 *HE5\_GDdefpixreg* routine) then the value is located at one of the four corners (*HE5\_HDFE\_GD\_UL*, *\_UR*, *\_LL*, *\_LR*) specified by the *HE5\_GDdeforigin* routine. All entries along the non-geographic dimensions (ie, NOT *XDim* and *YDim*) are interpolated and all interpolated values are returned as DOUBLE. 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* (2<sup>nd</sup> ed). (Note for the current version of this routine, the number type of the field to be interpolated is restricted to INT, LONG, FLOAT, DOUBLE.)

Example To interpolate the *Spectra* field at two geographic data points. In the first call, set the parameter *interpVal* to NULL:

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

```
latVal[0] = -20.8;
```

```
lonVal[1] = 15.8;
```

```
latVal[1] = 84.6;
```

```

double      *interpVal;

bufsiz = HE5_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 = (double *)calloc(bufsiz, sizeof(double));

bufsiz = HE5_GDinterpolate(gridID, 2, lonVal, latVal,
" Spectra", interpVal);

```

FORTTRAN integer\*4 function he5\_gdinterpolate(*gridid, ninterp, lonval, latval, fieldname, interpval*)

```

integer      gridid
integer*4    ninterp
real*8      lonval(*)
real*8      latval(*)
character*(*) fieldname
real*8      interpval(*)

```

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

```

real*8      interpval(NBANDS, 2)

ninterp = 2

bufsiz = he5_gdinterpolate(gridid, ninterp, lonval, latval,
" Spectra", interpval)

```

# Return Information about a Local Grid Attribute

---

## HE5\_GDlocattrinfo

herr\_t HE5\_GDlocattrinfo(hid\_t *gridID*, const char \**fieldname*, const char \**attrname*, hid\_t \**ntype*, hsize\_t \**count*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*fieldname* IN: Field name

*attrname* IN: Attribute name

*numbertype* OUT: Number type of attribute. See Appendix A for interpretation of number types.

*count* OUT: Number of attribute elements

Purpose Returns information about local attribute(s) in a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description This routine returns number type and number of elements (*count*) of a data field's local attribute(s).

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

```
status = HE5_GDlocattrinfo(gridID, "DataField", attrname,
&ntype, &count);
```

The *ntype* variable will have the value 10 and *count* will have the value 1.

FORTTRAN integer function he5\_gdlatrinfo(*gridid*, *fieldname*, *attrname*, *ntype*,  
*count*)

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

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

```
status = he5_gdlatrinfo(gridid, "DataField", attrname,
ntype, count)
```

# Return Number of specified Objects in a Grid

---

## HE5\_GDnentries

long HE5\_GDnentries(hid\_t *gridID*, int *entrycode*, long *\*strbufsize*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*entrycode* IN: Entry code

*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 successful or FAIL(-1) otherwise. A typical reason for failure is an improper Grid ID or entry code.

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

The entry codes are: HE5\_HDFE\_NENTDIM (0) - Dimensions

HE5\_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 = HE5_GDnentries(gridID, HE5_HDFE_NENTDFLD, &bufsize);
```

FORTRAN integer\*4 function he5\_gdnentries(*gridid*,*entrycode*, *bufsize*)

integer *gridid*

integer *entrycode*

integer\*4 *bufsize*

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

```
entrycode = 4
```

```
ndims = he5_gdnentries(gridid, entrycode, bufsize)
```

# Open HDF-EOS File

---

## HE5\_GDopen

hid\_t HE5\_GDopen(const char \*filename, uintn access)

<i>filename</i>	IN: Complete path and filename for the file to be opened
<i>access</i>	IN: H5F_ACC_RDONLY, H5F_ACC_RDWR or H5F_ACC_TRUNC
Purpose	Opens or creates HDF file in order to create, read, or write a grid.
Return value	Returns the grid file ID handle(fid) if successful or FAIL(-1) otherwise.
Description	This routine creates a new file or opens an existing one, depending on the access parameter.  Access codes:  H5F_ACC_RDONLY Open for read only. If file does not exist, error  H5F_ACC_RDWR Open for read/write. If file does not exist, error  H5F_ACC_TRUNC If file exists, delete it, then open a new file for read/write

Example In this example, we create a new grid file named, *Grid.he5*. It returns the file handle, *fid*.

```
fid = HE5_GDopen("Grid.he5", H5F_ACC_TRUNC);
```

FORTRAN integer function he5\_gdopen(*filename*, *access*)

character\*(\*) *filename*

integer *access*

The access codes should be defined as parameters:

parameter (HE5F\_ACC\_RDWR=100)

parameter (HE5F\_ACC\_RDONLY=101)

parameter (HE5F\_ACC\_TRUNC=102)

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

```
fid = he5_gdopen("Grid.he5", HE5F_ACC_TRUNC)
```

**Note to users of the SDP Toolkit:** Please refer to the *Release 7 SDP Toolkit User Guide for the ECS Project (333-EMD-001 Revision 05)*, 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

---

### HE5\_GDorigininfo

herr\_t HE5\_GDorigininfo(hid\_t *gridID*, int \**origincode*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*origincode* IN: Origin code

Purpose Retrieve origin code.

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

Description This routine retrieves the origin code.

Example In this example, we retrieve the origin code defined in *HE5\_GDdeforigin*.

```
status = HE5_GDorigininfo(gridID, &origincode);
```

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

FORTRAN integer function he5\_gdorginfo(*gridid*,*origincode*)

integer *gridid*

integer(\*) *origincode*

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

```
status = he5_gdorginfo(gridid, origincode)
```



## Return Pixel Registration Information

---

### HE5\_GDpixreginfo

herr\_t HE5\_GDpixreginfo(hid\_t *gridID*, int \**pixregcode*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*pixregcode* IN: Pixel registration code

Purpose Retrieve pixel registration code.

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

Description This routine retrieves the pixel registration code.

Example In this example, we retrieve the pixel registration code defined in *HE5\_GDdefpixreg*.

```
status = HE5_GDpixreginfo(gridID, &pixregcode);
```

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

FORTRAN integer function he5\_gdpreginfo(*gridid*,*pixregcode*)

```
integer      gridid
```

```
integer(*)   pixregcode
```

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

```
status = he5_gdpreginfo(gridid, pixregcode)
```

# Retrieve Grid Projection Information

---

## HE5\_GDprojinfo

herr\_t HE5\_GDprojinfo(hid\_t *gridID*, int \**projcode*, int \**zonecode*, int \**spherecode*, double *projparm*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_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 or FAIL(-1) otherwise

Description Retrieves the GCTP projection code, zone code, spheroid code and the projection parameters of the grid

Example In this example, we are retrieving the projection information from a grid attached to with *HE5\_GDattached*:

```
status = HE5_GDprojinfo(gridID, &projcode, &zonecode,
&spherecode, projparm);
```

FORTRAN integer function he5\_gdprojinfo( *gridid*, *projcode*, *zonecode*, *spherecode*,  
*projparm*)

integer(\*) *gridid*

integer(\*) *projcode*

integer(\*) *zonecode*

integer(\*) *spherecode*

real\*8 *projparm*(\*)

The equivalent FORTRAN code for the example above is:

```
status = he5_gdprojinfo(gridid, projcode, zonecode,
spherecode, projparm)
```

# Read Grid Attribute

---

## HE5\_GDreadattr

herr\_t HE5\_GDreadattr(hid\_t *gridID*, const char \**attrname*, void \**datbuf*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads attribute from a specific grid. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.

Example In this example, we read a floating point attribute with the name "ScalarFloat":

```
status = HE5_GDreadattr(gridID, "ScalarFloat", &attr_val);
```

FORTRAN integer function he5\_gdrdatr(*gridid*, *attrname*, *datbuf*)

```
integer      gridid
```

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

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

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

```
status = he5_gdrdatr(gridid, "ScalarFloat", attrval)
```

# Read Data From a Grid Field

---

## HE5\_GDreadfield

herr\_t HE5\_GDreadfield(hid\_t *gridID*, const char \**fieldname*, const hssize\_t *start*[], const hsize\_t *stride*[], const hsize\_t *edge*[], void \**buffer*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*fieldname* IN: Name of field to read  
*start* IN: Array specifying the starting location within each dimension  
*stride* IN: Array specifying the number of values to skip along each dimension  
*edge* IN: Array specifying the number of values to write along each dimension

*buffer* OUT: Buffer to store the data read from the field

Purpose Reads data from a grid field.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are improper Grid ID or unknown fieldname.

Description The values within *start*, *stride*, and *edge* arrays refer to the grid field (input) dimensions. The output data in *buffer* is written to 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 to the size of the dimension. Note that to allocate a string buffer size for reading an array of strings, first using *HE5\_GDreadlocattr* to get the value of maximum string length in the local attribute *StringLengthAttribute*.

Example In this example, we read data from the 10th row (0-based) of the *Temperature* field.

```
float row[120];  
  
hssize_t start[2]={10,1}; hsize_t edge[2]={1,120};  
  
status = HE5_GDreadfield(gridID, "Temperature", start, NULL,  
edge, row);
```

FORTRAN integer function

```

he5_gdrdfld(gridid,fieldname,start,stride,edge,buffer)
he5_gdrdcharfld(gridid,fieldname,elemen,numelem,start,stride,edge,
buffer)
integer      gridid
character*(*) fieldname
integer      elemen (each element length in array of string)
integer      numelem (number of elements in declared buffer array)
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.

**Note: `he5_gdrdcharfld()` is only for reading an array of character string field. For reading an array of single character field, please use `he5_gdrdfld()`.**

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 = he5_gdrdfld(gridid, "Temperature", start, stride,
edge, row)

```

## Read Group Grid Attribute

---

### HE5\_GDreadgrpattr

herr\_t HE5\_GDreadgrpattr(hid\_t *gridID*, const char \**attrname*, void \**datbuf*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads group attribute from the “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.

Example In this example, we read a floating point attribute with the name "ScalarFloat":

```
status = HE5_GDreadgrpattr(gridID, "ScalarFloat",
&attr_val);
```

FORTTRAN integer function he5\_gdrdgattr(*gridid*, *attrname*, *datbuf*)

integer *gridid*

character\*(\*) *attrname*

<valid type> *attrval*(\*)

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

```
status = he5_gdrdgattr(gridid, "ScalarFloat", attrval)
```

# Read Local Grid Attribute

---

## HE5\_GDreadlocattr

herr\_t HE5\_GDreadlocattr(hid\_t *gridID*, const char \**fieldname*, const char \**attrname*, void \**datbuf*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*fieldname* IN: Field name  
*attrname* IN: Attribute name  
*datbuf* OUT: Buffer allocated to hold attribute values  
Purpose Reads local attribute from a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).  
Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper Grid ID or number type or incorrect attribute name.  
Description The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.

Example In this example, we read a floating point attribute with the name "ScalarFloat":

```
status = HE5_GDreadlocattr(gridID, "DataField",  
"ScalarFloat", &attr_val);
```

FORTRAN integer function he5\_gdrdlattr(*gridid*, *fieldname*, *attrname*, *datbuf*)

```
integer      gridid  
character*(*) fieldname  
character*(*) attrname  
<valid type> attrval(*)
```

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

```
status = he5_gdrdlattr(gridid, "DataField", "ScalarFloat",  
attrval)
```

# Return Information about a Region

---

## HE5\_GDregioninfo

herr\_t HE5\_GDregioninfo(hid\_t *gridID*, hid\_t *regionID*, const char \* *fieldname*, hid\_t \**ntype*, int \**rank*, hsize\_t *dims*[], long \**size*, double *upleftpt*[], double *lowrightpt*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*regionID* IN: Region (period) ID returned by HE5\_GDdefboxregion (HE5\_GDdeftimeperiod)

*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

*upleftpt* OUT: Upper left point of subset region

*lowrightpt* OUT: Lower right point of subset region

Purpose Retrieves information about the subsetted region.

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

Description This routine returns information about a subsetted region for a particular field. It is useful when allocating space for a data buffer for the region. Because of differences in number type and geolocation mapping, a given region will give different values for the dimensions and size for various fields. The *upleftpt* and *lowrightpt* 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 *HE5\_GDdefboxregion* for the *Temperature* field. We use this to allocate space for data in the subsetted region.

```
status = HE5_GDregioninfo(GDdid, regionID, "Temperature",
&ntype, &rank, dims, &size, upleft, lowright);
```



FORTRAN integer function `he5_gdreginfo`(*gridid*, *regionid*, *fieldname*, *ntype*, *rank*,  
*dims*, *size*, *upleftpt*, *lowrightpt*)

integer *gridid*

integer *gridid*

character\*(\*) *fieldname*

integer *ntype*

integer *rank*

integer\*4 *dims*(\*)

integer\*4 *size*

real\*8 *upleftpt*(2)

real\*8 *lowrightpt*(2)

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

```
status = he5_gdreginfo(gridid, regid, "Spectra", ntype,  
rank, dims, size, upleftpt, lowrightpt)
```

# Create an Alias for Grid Data Field

---

## HE5\_GDsetalias

herr\_t HE5\_GDsetalias(hid\_t *gridID*, char \**fieldname*, const char \**aliaslist*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*fieldname* IN: Field name

*aliaslist* IN: List of alias(es) to associate with the Data Field

Purpose Create an alias for Grid data field

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

Description Creates aliases that can be used to refer to a Grid data field in addition to the name of the field.

Example In this example, we create an alias for the data field *Temperature*.

```
strcpy(aliaslist, "temps 0 to 30");  
status = HE5_GDsetalias(gridID, "Temperature", aliaslist);
```

FORTRAN integer function he5\_gdsetalias (*gridid*, *fieldname*, *aliaslist*)  
integer *gridid*  
character\*(\*) *fieldname*  
character\*(\*) *aliaslist*

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

```
aliaslist = "temps 0 to 30"  
status = he5_gdsetalias(gridid, "Temperature", aliaslist)
```

## Set External Data File(s)

---

### HE5\_GDsetextdata

herr\_t HE5\_GDsetextdata(hid\_t *gridID*, const char \**filelist*, off\_t *offset[]*, hsize\_t *size[]*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*filelist* IN: List of external file names  
*offset[]* IN: Array of offsets (in byte) from the beginning of file to the location in file where the data starts  
*size[]* IN: Array of sizes (in bytes) reserved in the file for the data

Purpose Sets the external data file(s) associated with the data set.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper grid ID.

Example In this example, we set the *ExtData* field:

```
status = HE5_GDsetextdata(gridID, "ext-1.dat,ext-2.dat,ext-3.dat", offset, size);
```

FORTRAN integer function he5\_gdsetxdat(*gridid*,*flist*,*offset*, *size*)

```
integer      gridid  
integer      status  
integer*4    offset(*)  
integer*4    size(*)  
character*(*) flist
```

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

```
status = he5_gdsetxdat(gridid,flist,offset,size)
```

## Set Fill Value for a Specified Field

---

### HE5\_GDsetfillvalue

herr\_t HE5\_GDsetfillvalue(hid\_t *gridID*, const char \**fieldname*, hid\_t *ntype*, void \**fillvalue*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*fieldname* IN: Fieldname  
*ntype* IN: Number type of fill value (should match the number type of a specified field)  
*fillvalue* IN: Pointer to the fill value to be used  
Purpose Sets fill value for the specified field.

**NOTE: THIS FUNCTION MUST BE CALLED BEFORE THE FUNCTION CALL TO DEFINE THE FIELD IT IS TO BE APPLIED. SETS A FILL VALUE FOR A CHARACTER STRING FIELD IS NOT AVAILABLE IN THIS RELEASE.**

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

Description The fill value is placed in all elements of the field which have not been explicitly defined.

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

```
tempfill = -999.0;
status = HE5_GDsetfillvalue(gridID, "Temperature", ntype,
&tempfill);
```

FORTRAN integer function he5\_gdsetfill(*gridid*,*fieldname*,*ntype*,*fillvalue*)

```
integer*4    gridid
character*(*) fieldname
integer      ntype
<valid type> fillvalue(*)
```

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

```
fillvalue = -999.0
status = he5_gdsetfill(gridid, "Temperature", ntype, fillvalue)
```

# Retrieve Tiling Information about a Grid Field

---

## HE5\_GDtileinfo

herr\_t HE5\_GDtileinfo(hid\_t *gridID*, char \**fieldname*, int \**tilecode*, int \**tilerank*, hsize\_t *tiledims*[])

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach  
*fieldname* IN: Fieldname  
*tilecode* OUT: Tile code: HE5\_HDFE\_TILE (1), HE5\_HDFE\_NOTILE (0)  
*tilerank* OUT: The number of tile dimensions  
*tiledims* OUT: Tile dimensions  
Purpose Retrieve tiling information about a specific field in the grid.  
Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.  
Description This routine returns the tiling code, tiling rank, and tiling dimensions for a given field.

Example In this example, we retrieve the tiling information about the *Pressure* fields:

```
status = HE5_GDtileinfo(gridID, "Pressure", &tilecode,  
&tilerank, tiledims);
```

The returned parameters will have the following values:

*Tilecode = 1, tilerank=2, tiledims[2]={100,200}*

FORTRAN integer function he5\_gdtileinfo(*gridid*, *fieldname*, *tilecode*, *tilerank*, *tiledims*)

integer *gridid*  
character\*(\*) *fieldname*  
integer *tilecode*  
integer *tilerank*  
integer\*4 *tiledims*(\*)

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

```
status = he5_gdtileinfo(gridid, "Pressure", tilecode,  
tilerank, tiledims)
```

The return parameters will have the following values:

*tilecode=1, tilerank=2, tiledims[3]={200,100}*

Note that the dimensions array is in FORTRAN order.

# Write/Update Grid Attribute

---

## HE5\_GDwriteattr

herr\_t HE5\_GDwriteattr(hid\_t *gridID*, const char \**attrname*, hid\_t *ntype*, hsize\_t *count*[], void \**datbuf*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_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 an object attribute in a specific grid. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call.

Example In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14:

```
attr_val = 3.14;
```

```
count[0] = 1;
```

```
status=HE5_GDwriteattr(gridid, "ScalarFloat", H5T_NATIVE_FLOAT  
, count, &attr_val);
```

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

```
attr_val = 3.14159;
```

```
status=HE5_GDwriteattr(gridid, "ScalarFloat", H5T_NATIVE_FLOAT  
, count, &attr_val);
```

FORTRAN integer function `he5_gdwratr(gridid, attrname, ntype, count, datbuf)`

integer `gridid`  
character\*(\*) `attrname`  
integer `ntype`  
integer\*4 `count(*)`  
<valid type> `attrval(*)`

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

```
parameter (HE5T_NATIVE_FLOAT=10)

attrval = 3.14

count(1)= 1

status=he5_gdwratr(gridid,"ScalarFloat",HE5T_NATIVE_FLOAT,
count, attrval)
```



# Write Data to a Grid Field

---

## HE5\_GDwritefield

herr\_t HE5\_GDwritefield(hid\_t *gridID*, const char *fieldname*, const hssize\_t *start*[], const hsize\_t *stride*[], const hsize\_t *edge*[], void *data*)

<i>gridID</i>	IN:	Grid ID returned by HE5_GDcreate or HE5_GDattach
<i>fieldname</i>	IN:	Name of field to write
<i>start</i>	IN:	Array specifying the starting location within each dimension (0-based)
<i>stride</i>	IN:	Array specifying the number of values to skip along each dimension
<i>edge</i>	IN:	Array specifying the number of values to write along each dimension
<i>data</i>	IN:	Values to be written to the field
Purpose		Writes data to a grid field.
Return value		Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description		The values within <i>start</i> , <i>stride</i> , and <i>edge</i> arrays refer to the grid field (output) dimensions. The input data in the <i>data</i> buffer is read from contiguously. The default 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>HE5_GDdeftile</i> for further information.

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

```
float temperature [200][120];

/* Define elements of temperature array */

status = HE5_GDwritefield(gridID, "Temperature", NULL, NULL,
                          NULL, temperature);

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

float newrow[2000];

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

```

status = HE5_GDwritefield(gridID, "Temperature", start, NULL,
edge, newrow);

```

FORTRAN integer function

```

he5_gdwrfld(gridid,fieldname,start,stride,edge,data)
he5_gdwrcharfld(gridid,fieldname,elemen,numelem,start,stride,edge,data)
)
integer      gridid
character*(*) fieldname
integer      elemen (each element length in array of string)
integer      numelem (number of elements in declared buffer array)
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.

**Note: `he5_gdwrcharfld()` is only for writing an array of character string field. For writing an array of single character field, please use `he5_gdwrfld()`.**

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 = he5_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 = he5_gdwrfld(gridid, "Temperature", start, stride,
edge, newrow)
```

# Write Field Metadata for an Existing Field not Defined with the Grid API

---

## HE5\_GDwritefieldmeta

herr\_t HE5\_GDwritefieldmeta(hid\_t *gridID*, const char \**fieldname*, char \**dimlist*, int *ntype*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*fieldname* IN: Name of field that metadata information is to be written

*dimlist* IN: Dimension list of field

*ntype* 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 or FAIL(-1) otherwise

Description This routine writes the field metadata for a grid field not defined by the Grid API

Example 

```
status = HE5_GDwritefieldmeta(gridID, "ExternField",
"Ydim,Xdim", HE5_HDFE_NATIVE_FLOAT);
```

FORTRAN integer function he5\_gdwrmeta(gridid, fieldname, dimlist, ntype)

integer *gridid*

character\*(\*) *fieldname*

character\*(\*) *dimlist*

integer *ntype*

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

```
parameter (HE5T_NATIVE_FLOAT=10)
```

```
status = he5_gdwrmeta(gridid, "ExternField", "Xdim,Ydim",
HE5T_NATIVE_FLOAT)
```

# Write/Update Group Grid Attribute

---

## HE5\_GDwritegrpattr

herr\_t HE5\_GDwritegrpattr(hid\_t *gridID*, const char \**attrname*, hid\_t *ntype*,  
hsize\_t *count*[], void \**datbuf*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*attrname* IN: Attribute name

*ntype* IN: Data type of attribute

*count* IN: Number of values to store in attribute

*datbuf* IN: Attribute values

**Purpose** Writes/Updates group attribute in the “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

**Description** If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to the “Data Fields” group in the grid file.

**Example** In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14:

```
count[0] = 1;  
  
attr_val = 3.14;  
  
status = HE5_GDwritegrpattr(gridid, "ScalarFloat",  
H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```
attr_val = 3.14159;  
  
status = HE5_GDwritegrpattr(gridid, "ScalarFloat",  
H5T_NATIVE_FLOAT, count, &attr_val);
```

FORTRAN integer function `he5_gdwrgattr(gridid, attrname, ntype, count, datbuf)`  
integer `gridid`  
character\*(\*) `attrname`  
integer `ntype`  
integer\*4 `count(*)`  
<valid type> `attrval(*)`

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

```
parameter (HE5T_NATIVE_FLOAT=10)
attrval = 3.14
count(1) = 1
status = he5_gdwrgattr(gridid, "ScalarFloat",
HE5T_NATIVE_FLOAT, count, attrval)
```

## Write/Update Local Grid Attribute

---

### HE5\_GDwritelocattr

herr\_t HE5\_GDwritelocattr(hid\_t *gridID*, const char \**fieldname*, const char \**attrname*, hid\_t *ntype*, hsize\_t *count*[], void \**datbuf*)

*gridID* IN: Grid ID returned by HE5\_GDcreate or HE5\_GDattach

*fieldname* IN: Field name

*attrname* IN: Attribute name

*ntype* IN: Data type of attribute

*count* IN: Number of values to store in attribute

*datbuf* IN: Attribute values

**Purpose** Writes/Updates local attribute in a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

**Description** If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to a particular "Data Field" in the grid file.

**Example** In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14:

```
count[0] = 1;

attr_val = 3.14;

status = HE5_GDwritelocattr(gridid, "DataField",
"ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```
attr_val = 3.14159;

status = HE5_GDwritelocattr(gridid, "DataField",
"ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

FORTRAN integer function `he5_gdwrlattr(gridid, fieldname, attrname, ntype, count, datbuf)`

integer *gridid*

character\*(\*) *fieldname*

character\*(\*) *attrname*

integer\*4 *count*(\*)

<valid type> *attrval*(\*)

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

```
parameter (HE5T_NATIVE_FLOAT=10)
```

```
attrval = 3.14
```

```
count(1) = 1
```

```
status = he5_gdwrlattr(gridid, "DataField", "ScalarFloat",  
HE5T_NATIVE_FLOAT, count, attrval)
```



#### **2.1.4 HDF-EOS Utility Routines**

This section contains an alphabetical list of the utility functions. The functions are alphabetized on their C-language names.

# Convert Among Angular Units

---

## HE5\_EHconvAng

double HE5\_EHconvAng(double *inAngle*, int *code*)

*inAngle* IN: Input angle

*code* IN: Conversion code

Purpose Convert among various angular units.

Return value Returns angle in desired units if successful or 0 otherwise.

Description This routine converts angles between three units, decimal degrees, radians, and packed degrees-minutes-seconds. In the later unit, an angle is expressed as a integral number of degrees and minutes and a float point value of seconds packed as a single double number as follows: DDDMMMSS.SS. The six conversion codes are: HE5\_HDFE\_RAD\_DEG (0), HE5\_HDFE\_DEG\_RAD (1), HE5\_HDFE\_DMS\_DEG (2), HE5\_HDFE\_DEG\_DMS (3), HE5\_HDFE\_RAD\_DMS (0), and HE5\_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 = HE5_EHconvAng(inAng, HDFE_DEG_DMS);
```

“outAng” will contain the value: 27030000.00.

FORTRAN real\*8 function he5\_ehconvang(*inangle*,*code*)

```
real*8      inangle
```

```
integer     code
```

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

```
inangle = 27.5
```

```
code = 3
```

```
outangle = he5_ehconvang(inangle, code)
```

# Get HDF-EOS Version String

---

## HE5\_EHgetversion

herr\_t HE5\_EHgetversion(hid\_t *fid*, char \**version*)

*fid* IN: File ID returned by *HE5\_SWopen*, *HE5\_GDopen*, or *HE5\_PTopen*.

*version* OUT: HDF-EOS version string

Purpose Get HDF-EOS version string.

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

Description This routine returns the HDF-EOS version string of an HDF-EOS file. This designates the version of HFD-EOS that was used to create the file. This string is of the form: "HDFEOS\_V*maj*.*min*" where *maj* is the major version and *min* is the minor version.

Example To get the HDF-EOS version (assumed to be 5.1.2) used to create the HDF-EOS file: "Swath.he5":

```
char version[16];  
fid = HE5_SWopen("Swath.he5", H5F_ACC_RDONLY);  
status = HE5_EHgetversion(fid, version);  
"version" will contain the string: "HDFEOS_5.1.2".
```

FORTRAN integer function he5\_ehgetver(*fid*,*version*)

```
integer fid  
character*(*) version  
integer HE5F_ACC_RDONLY  
parameter (HE5F_ACC_RDONLY=101)
```

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

```
character*16 version  
  
fid = he5_swopen("Swath.he5", HE5F_ACC_RDONLY)  
  
status = he5_ehgetver(fid, version)
```

# Return Information about Global File Attribute

---

## HE5\_EHglbatrinfo

herr\_t HE5\_EHglbatrinfo(hid\_t *fileID*, const char \**attrname*, hid\_t \**ntype*,  
hsz\_t \**count*)

*fileID* IN: HDF-EOS file ID returned by  
HE5\_SWopen/HE5\_GDopen/HE5\_PTopen

*attrname* IN: Attribute name

*numbertype* OUT: Number type of attribute. See Appendix A for interpretation of  
number types.

*count* OUT: Number of attribute elements

Purpose Returns information about Global File attribute. See Section 3.6 of  
Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description This routine returns number type and number of elements (count) of  
Global File attribute.

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

```
status = HE5_EHglbatrinfo(fileID, "FloatAttr", &nt,  
&count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

FORTRAN integer function he5\_ehglattinf(*fileid*, *attrname*, *ntype*, *count*,)  
integer *fileid*  
character\*(\*) *attrname*  
integer *ntype*  
integer \*4 *count*

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

```
status = he5_ehglattinf(fileid, "FloatAttr",nt,count)
```

## Get HDF-EOS File IDs

---

### HE5\_EHidinfo

herr\_t HE5\_EHidinfo(hid\_t *fid*, hid\_t \**HDFfid*, hid\_t \**gid*)

*fid* IN: File ID returned by *HE5\_SWopen*, *HE5\_GDopen*, or *HE5\_PTopen*.

*HDFfid* OUT: HDF-EOS file ID (returned by *HE5\_EHopen*)

*gid* OUT: "HDFEOS" group ID

Purpose Get HDF-EOS file IDs.

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

Description This is a wrapper around *HE5\_EHchkfid* () and it returns the HDF file IDs to the HDF-EOS file ID returned by *HE5\_SWopen*, *HE5\_GDopen*, or *HE5\_PTopen*. These ids can then be used to create or access HDF5 structures such as groups, attributes, datasets within an HDF-EOS file.

# Retrieve Information about Global File Attributes

---

## HE5\_EHinqglbattr

long HE5\_EHinqglbattr(hid\_t *fileID*, char \**attrnames*, long \**strbufsize*)

*fileID* IN: HDF-EOS file ID returned by  
HE5\_SWopen/HE5\_GDopen/HE5\_PTopen

*attrnames* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

Purpose Retrieve information about Global attributes defined in file. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each group 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 Global attributes defined for the “swath.he5” file (with the file ID *fileID*). In the first call, set the parameter *attrnames* to NULL. We assume that there are two attributes stored, *GlobAttr\_1* and *GlobAttr\_2*:

```
nattr = HE5_EHinqglbattr(fileID, NULL, &strbufsize);
```

The parameter *nattr* will have the value 2 and *strbufsize* will have value 21.

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_EHinqglbattr(fileID, attrnames, &strbufsize);
```

The variable, *attrnames*, will be set to: "*GlobAttr\_1,GlobAttr\_2*".

FORTRAN integer\*4 function he5\_ehinqglatts(*fileid*, *attrnames*, *strbufsize*)

integer *fileid*

character\*(\*) *attrnames*

integer\*4 *strbufsize*

integer\*4 *nattr*

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

```
nattr = he5_ehinqglatts(fileid, attrnames, strbufsize)
```

# Return Data Type Information about Global File Attribute

---

## HE5\_EHinqglbdatatype

herr\_t HE5\_EHinqglbdatatype(hid\_t *fileID*, const char \**attrname*, hid\_t \**datatype*, H5T\_class\_t \**classid*, H5T\_order\_t \**order*, size\_t \**size*)

*fileID* IN: HDF-EOS file ID returned by HE5\_SWopen/HE5\_GDopen/HE5\_PTopen

*attrname* IN: Attribute name

*datatype* OUT: Data type ID

*classID* OUT: Data type class ID

*order* OUT: Data type byte order

*size* OUT: Data type size (in bytes)

Purpose Returns data type information about Global File attribute

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

Description This routine returns the data type information of Global File attribute.

Example In this example, we return the data type information about the *FloatAttr* attribute defined in the HE5\_EHwriteglbattr routine.

```
status = HE5_EHinqglbdatatype(fileID, "FloatAttr",
&datatype, &classid, &order, &size);
```

FORTRAN integer function he5\_ehinqglbtype(*fileid*, *attrname*, *datatype*, *classid*, *order*, *size*)

integer *fileid*

character\*(\*) *attrname*

integer *datatype*, *classid*, *order*

integer \*4 *size*

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

```
status = he5_ehinqglbtype(fileid, "FloatAttr", datatype,
classid, order, size)
```

# Read Global File Attribute

---

## HE5\_EHreadglbattr

herr\_t HE5\_EHreadglbattr(hid\_t *fileID*, const char \**attrname*, void \**datbuf*)

*fileID* IN: HDF-EOS file ID returned by  
HE5\_SWopen/HE5\_GDopen/HE5\_PTopen

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads global attribute from a file. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper file 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 floating point attribute with the name "FloatAttr":

```
status = HE5_EHreadglbattr(fileID, "FloatAttr", &data);
```

FORTRAN integer function he5\_ehrdglatt(*fileid*,*attrname*,*datbuf*)

```
integer      fileid
```

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

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

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

```
status = he5_ehrdglatt(fileid, "FloatAttr", datbuf)
```



# Set Flag for Suppressing HDF5 Error Messages

---

## HE5\_EHset\_error\_on

hsize\_t HE5\_EHset\_error\_on(int *flag*, int *err\_level*)

*flag* IN: Input flag for suppressing HDF5 error messages  
0: Print both HDF-EOS5 and HDF5 error messages  
1: Print only HDF-EOS5 error messages  
2: Suppress all error messages

*err\_level* IN: A dummy flag for future applications

Purpose Sets a global flag value.

Return value Returns a global flag value.

Description This routine sets a global flag to suppress HDF5 and/or HDF-EOS5 error messages. A flag set by user will be in effect until next call to this routine, where user may change the flag to another value.

Example To suppress HDF5 error messages only:

```
status = HE5_EHset_error_on(1, 0);
```

To suppress both HDF5 and HDF-EOS5 error messages:

```
status = HE5_EHset_error_on(2, 0);
```

FORTRAN integer\*4 function he5\_ehset\_error\_onf(*flag*,*err\_level*)

integer *flag*

integer *err\_level*

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

```
status = he5_ehset_error_onf(1, 0)
```

```
status = he5_ehset_error_onf(2, 0)
```

## Write/Update Global File Attribute

---

### HE5\_EHwriteglbattr

herr\_t HE5\_EHwriteglbattr(hid\_t *fileID*, const char \**attrname*, hid\_t *ntype*,  
hsize\_t *count*[], void \**datbuf*)

*fileID* IN: HDF-EOS file ID returned by  
HE5\_SWopen/HE5\_GDopen/HE5\_PTopen

*attrname* IN: Attribute name

*ntype* IN: Data type of attribute

*count* IN: Number of values to store in attribute

*datbuf* IN: Attribute values

Purpose Writes/Updates Global attribute in HDF-EOS file. See Section 3.6 of  
Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical  
reasons for failure are an improper file 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. The  
attribute is linked to the "ADDITIONAL/FILE ATTRIBUTES" group in  
the HDF-EOS file.

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

```
count[0] = 1;  
  
attr_val = 3.14;  
  
status = HE5_EHwriteglbattr(fileid, "FloatAttr",  
H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```
attr_val = 3.14159;  
  
status = HE5_EHwriteglbattr(fileid, "FloatAttr",  
H5T_NATIVE_FLOAT, count, &attr_val);
```

FORTRAN integer function he5\_ehwrglatt(*fid*,*attrname*,*ntype*,*count*,*buffer*)

integer        *fid,status,ntype*  
character \*(\*) *attrname*  
integer\*4     *count*  
<valid type> *buffer(\*)*  
integer        *HE5T\_NATIVE\_FLOAT*  
parameter     (*HE5T\_NATIVE\_FLOAT=10*)

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

```
count = 1  
status = he5_ehwrglatt(fid, "FloatAttr", HE5T_NATIVE_FLOAT,  
count, buffer)
```

### **2.1.5 Zonal Average Interface Functions**

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

# Return Information about an Alias

---

## HE5\_ZAaliasinfo

herr\_t HE5\_ZAaliasinfo(hid\_t *zaid*, int *fldgroup*, const char \**aliasname*, int \**length*, char \**buffer*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fldgroup* IN: Field group flag

*aliasname* IN: Name of alias to retrieve information about

*length* IN/OUT: Size of buffer in bytes

*buffer* OUT: Buffer with original field name

Purpose Return information about an alias

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

Description This routine returns a buffer size and the buffer with an original field name.

Example In this example, we return the buffer size and the original data field *Temperature*. In the first call, set *namebuffer* to NULL and *length* is an output parameter. In the second call, *length* is an input parameter.

```
status = HE5_ZAaliasinfo(zaid, HE5_HDFE_DATAGROUP,
aliasname, length, NULL);
```

```
namebuffer = (char *)calloc(length + 1, sizeof(char));
```

```
status = HE5_ZAaliasinfo(zaid, HE5_HDFE_DATAGROUP,
aliasname, length, namebuffer);
```

FORTRAN integer function he5\_zaaliasinfo (*zaid*, *fldgroup*, *aliasname*, *length*, *buffer*)

integer *zaid*, *status*

character\*(\*) *fldgroup*

character\*(\*) *aliasname*

integer\*(\*) *length*

character\*(\*) *buffer*

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

```
aliaslist = "temps 0 to 30"
```

```
status = he5_zaaliasinfo(zaid, HE5_HDFE_DATAGROUP,
aliaslist, length, buffer)
```

## Attach to an Existing ZA Structure

---

### HE5\_ZAattach

hid\_t HE5\_ZAattach(hid\_t *fid*, const char \**zaname*)

*fid* IN: ZA file ID returned by HE5\_ZAopen

*zaname* IN: Name of za to be attached

Purpose Attaches to an existing za within the file.

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

Description This routine attaches to the za using the *zaname* parameter as the identifier.

Example In this example, we attach to the previously created za, "ExampleZA", within the HDF-EOS file, ZA.he5, referred to by the handle, *fid*:

```
zaID = HE5_ZAattach(fid, "ExampleZA");
```

The za can then be referenced by subsequent routines using the handle, *zaID*.

FORTRAN integer function he5\_zattach(*fid*,*zaname*)

integer *fid*

character\*(\*) *zaname*

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

```
zaid = he5_zattach(fid, "ExampleZA")
```

## Return Information about a ZA Attribute

---

### HE5\_ZAttrinfo

herr\_t HE5\_ZAttrinfo(hid\_t *zaID*, const char \**attrname*, hid\_t \**ntype*, hsize\_t \**count*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*attrname* IN: Attribute name

*ntype* OUT: Number type of attribute

*count* OUT: Number of elements in attribute

Purpose Returns information about an attribute in a specific za. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

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

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

```
status = HE5_ZAttrinfo(zaID, "ScalarFloat",&nt,&count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

FORTRAN integer function he5\_zattrinfo(*zaid*, *attrname*, *ntype*, *count*)

integer *zaid*

character\*(\*) *attrname*

integer *ntype*

integer\*4 *count*

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

```
status = he5_zattrinfo(zaid, "ScalarFloat",ntype,count)
```

# Retrieve Chunking Information about a Zonal Average Field

---

## HE5\_ZAchunkinfo

herr\_t HE5\_ZAchunkinfo(hid\_t *zaID*, char \**fldname*, int \**chunk\_rank*, hsize\_t *chunk\_dims*[])

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fldname* IN: Field name

*chunk\_rank* OUT: The number of chunking dimensions

*chunk\_dims* OUT: Array containing the chunking dimension sizes of the field

Purpose Retrieve chunking information about a specific field.

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

Description This routine returns the chunking rank and chunking dimensions for a given field.

Example In this example, we retrieve the chunking information about the *Count* data fields:

```
status = HE5_ZAchunkinfo(zaID, "Count", &rank_rank,  
rank_dims);
```

The return parameters will have the following values:

```
chunk_rank=2, chunk_dims[2]={100,360}
```

FORTRAN integer function he5\_zachunkinfo(*zaid*, *fldname*, *chunk\_rank*,  
*chunk\_dims*)

integer *zaid*

character\*(\*) *fldname*

integer *chunk\_rank*

integer\*4 *chunk\_dims*(\*)

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

```
status = he5_zachunkinfo(zaid, "Count", chunk_rank,  
chunk_dims)
```

The return parameters will have the following values:



*chunk\_rank=2, chunk\_dims[2]={360,100}*

Note that the dimensions array are in FORTRAN order.

# Close an HDF-EOS File

---

## HE5\_ZAclose

herr\_t HE5\_ZAclose(hid\_t *fid*)

*fid* IN: ZA file ID returned by HE5\_ZAopen

Purpose Closes file.

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

Description This routine closes the HDF-EOS ZA file.

Example

```
status = HE5_ZAclose(fid);
```

FORTRAN integer function he5\_zaclose(*fid*)

integer *fid*

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

```
status = he5_zaclose(fid)
```

# Retrieve Compression Information for Field

---

## HE5\_ZAcompinfo

herr\_t HE5\_ZAcompinfo(hid\_t *zaID*, const char \**fieldname*, int \**compcode*, int *compparm*[])

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*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 or 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 *HE5\_ZAdefcomp* function:

```
status = HE5_ZAcompinfo(zaID, "Opacity", &compcode,
compparm);
```

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

FORTRAN integer function he5\_zacompinfo(*zaid*,*fieldname* *compcode*, *compparm*)

integer *zaid*

character\*(\*) *fieldname*

integer *compcode*

integer *compparm*(\*)

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

```
status = he5_zacompinfo(zaid, 'Opacity', compcode, compparm)
```

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

## Create a New ZA Structure

---

### HE5\_ZAcreate

hid\_t HE5\_ZAcreate(hid\_t *fid*, const char \**zaname*)

*fid* IN: ZA file ID returned by HE5\_ZAopen

*zaname* IN: Name of za to be created

Purpose Creates a za within the file.

Return value Returns the za handle (*zaID*) if successful or FAIL (-1) otherwise.

Description The za is created as a Group within the HDF-EOS file with the name *zaname*.

Example In this example, we create a new za structure, *ExampleZA*, in the previously created file, *ZA.he5*.

```
zaID = HE5_ZAcreate(fid, "ExampleZA");
```

The za structure is referenced by subsequent routines using the handle, *zaID*.

FORTRAN integer function he5\_zacreate(*fid*,*zaname*)

integer *fid*

character\*(\*) *zaname*

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

```
zaid = he5_zacreate(fid, "ExampleZA")
```

# Define Chunking Parameters

---

## HE5\_ZAdefchunk

herr\_t HE5\_ZAdefchunk(hid\_t *zaID*, int *chunk\_rank*, const hsize\_t \**chunk\_dims*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach  
*chunk\_rank* IN: The number of chunk dimensions (a number other than zero)  
*chunk\_dims* IN: Chunk dimensions (NULL cannot be used)  
Purpose Defines chunking for subsequent field definitions  
Return Value Returns SUCCEED(0) if successful or FAIL(-1) otherwise  
Description This routine defines the chunking dimensions for fields defined following this function call, analogous to the procedure for setting the field compression scheme using *HE5\_ZAdefcomp*. The number of chunk dimensions and subsequent field dimensions must be the same.

Example We will define chunking for a two-dimensional field of size 2400 x 3600.

```
chunk_dims[0] = 100;  
chunk_dims[1] = 360;  
status = HE5_ZAdefchunk(zaID, 2, chunk_dims);
```

FORTRAN integer function he5\_zadefchunk(*zaid*, *chunk\_rank*,*chunk\_dims*)

```
integer      zaid  
integer      chunk_rank  
integer*4    chunk_dims(*)
```

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

```
chunk_dims(1) = 360  
chunk_dims(2) = 100  
chunk_rank    = 2  
status = he5_zadefchunk(zaid, chunk_rank, chunk_dims)
```

# Define Compression with Data Chunking

---

## HE5\_ZAdefcomchunk

herr\_t HE5\_ZAdefcomchunk(hid\_t *zaid*, int *compcode*, int \**compparm*, int *ndims*,  
const hsize\_t \**dim*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*compcode* IN: Compression method flag

*compparm* IN: Array of compression parameters

*ndims* IN: Rank of a field to compress (a number other than zero)

*dim* IN: Array of sizes of chunk (NULL cannot be used)

Purpose Compress the data field

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

Description This function allows the user to set compression for a data field with automatic chunking

Example In this example, we set (DEFLATE) compression for a field that is defined right after this call

```
ndims      = 2
compcode   = 4;
compparm[0] = 6;
dim[0]     = 100;
dim[1]     = 200;

status = HE5_ZAdefcomchunk(zaid, compcode, compparm, ndims,
dim);
```

FORTRAN integer function he5\_zadefcomch(zaid,compcode, compparm, ndims,dim)

```
integer    zaid
integer    compcode
integer    compparm(*)
integer    ndims
integer*4  dim(*)
```

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

```
compcode      = 4
compparm(1)   = 6
ndims         = 2
dim(1)        = 200
dim(2)        = 100
status = he5_zadefcomch(zaid, compcode, compparm, ndims,
dim)
```

# Set ZA Field Compression

---

## HE5\_ZAdefcomp

herr\_t HE5\_ZAdefcomp(hid\_t zaID, int compcode, int \*compparm)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*compcode* IN: HDF compression code

*compparm* IN: Compression parameters (if applicable)

Note: Only deflate compression is available in this release.

Purpose Sets the field compression for all subsequent field definitions.

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

Description This routine sets the HDF field compression for subsequent za field definitions. The routine HE5\_ZAdefchunk() must be called first, otherwise HE5\_ZAdefcomp doesn't work. The compression does not apply to one-dimensional fields. The compression schemes currently supported is: deflate (gzip) (HE5\_HDFE\_COMP\_DEFLATE=4) and no compression (HE5\_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 *HE5\_ZAwrite* routine, however, the entire field must be written in a single call. Any portion of a compressed field can then be accessed with the *HE5\_ZAread* routine. 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 = HE5_ZAdefcomp(zaID, HE5_HDFE_COMP_DEFLATE, NULL);

status = HE5_ZAdefine(zaID, "Pressure", "Track,Xtrack",
NULL, H5T_NATIVE_FLOAT);

compparm[0] = 5;

status = HE5_ZAdefine(zaID, "Opacity", "Track,Xtrack", NULL,
H5T_NATIVE_FLOAT);

status = HE5_ZAdefine(zaID, "Spectra", "Bands,Track,Xtrack",
NULL, H5T_NATIVE_FLOAT);

status = HE5_ZAdefcomp(zaID, HE5_HDFE_COMP_NONE, NULL);
```



```
status = HE5_ZAdefine(zaid, "Temperature", "Track,Xtrack",  
NULL, H5T_NATIVE_FLOAT);
```

FORTRAN integer function he5\_zadefcomp(zaid, compcode, compparm)

integer        *zaid*

integer        *compcode*

integer        *compparm*(\*)

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

```
parameter (HE5T_NATIVE_FLOAT=10)
```

```
parameter (HE5_HDFE_COMP_NONE=0)
```

```
parameter (HE5_HDFE_COMP_DEFLATE=4)
```

```
integer        compparm(5)
```

```
status = he5_zadefcomp(zaid, HE5_HDFE_COMP_DEFLATE,  
compparm);
```

```
oompparm(1) = 5
```

```
status = he5_zadefine(zaid, "Pressure", "Xtrack,Track", " ",  
HE5T_NATIVE_FLOAT);
```

```
status = he5_zadefine(zaid, "Opacity", "Xtrack,Track", " ",  
HE5T_NATIVE_FLOAT);
```

```
status = he5_zadefine(zaid, "Spectra", "Xtrack,  
Track,Bands", " ", HE5T_NATIVE_FLOAT)
```

```
status = he5_zadefcomp(zaid, HE5_HDFE_COMP_NONE, compparm)
```

```
status = he5_zadefine(zaid, "Temperature", "Xtrack,Track",  
" ", HE5T_NATIVE_FLOAT)
```

# Define a New Dimension within a Zonal Average

---

## HE5\_ZAdefdim

herr\_t HE5\_ZAdefdim(hid\_t *zaID*, char \**dimname*, hsize\_t *dim*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*dimname* IN: Name of dimension to be defined

*dim* IN: The size of the dimension

**Note: There are three illegal characters for dimension names: “/”, “;”, “.”**

Purpose Defines a new dimension within the za.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is an improper ZA 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 track1 dimension, *MyTrack1*, of size 2000, a track2 dimension, *MyTrack2*, of size 1000, a track data dimension, *DataTrack*, of size of 4000, and a cross track data dimension, *DataXtrack*, of size 2000:

```
status = HE5_ZAdefdim(zaID, "MyTrack1", 2000);
status = HE5_ZAdefdim(zaID, "MyTrack2", 1000);
status = HE5_ZAdefdim(zaID, "DataTrack", 4000);
status = HE5_ZAdefdim(zaID, "DataXtrack", 2000);
status = HE5_ZAdefdim(zaID, "Bands", 5);
```

To specify an unlimited dimension which can be used to define an appendable array, the dimension value should be set to -1 or equivalently, *H5S\_UNLIMITED*:

```
status = HE5_ZAdefdim(zaID, "Unlim", H5S_UNLIMITED);
```

FORTTRAN integer function he5\_zadefdim(*zaid*,*dimname*,*dim*)

```
integer      zaid
character*(*) dimname
integer*4    dim
```

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

```
dim      = 4000
```

```
status = he5_zadefdim(zaid, "DataTrack", dim)
```

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

```
parameter (HE5S_UNLIMITED_F=-1)
```

```
status = he5_zadefdim(zaid, "Unlim", HE5S_UNLIMITED_F)
```

# Define a New Data Field within a ZA

---

## HE5\_ZAdefine

herr\_t HE5\_ZAdefine(hid\_t *zaid*, const char \**za\_name*, char \**dimlist*, char \**maxdimlist*, hid\_t *dtype*)

*zaid*: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*za\_name* IN: Name of field to be defined

*dimlist* IN: The list of data dimensions defining the field

*maxdimlist* IN: The list of maximum data dimensions defining the field

*dtype* IN: The data type of the data stored in the field

Purpose Defines a new data field within the Zonal Average.

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

Description This routine defines data fields to be stored in the zonal average. The dimensions are entered as a string consisting of data fields dimensions separated by commas. They are entered in *C* order, that is, the last dimension is incremented first.

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

```
status = HE5_ZAdefine(zaid, "Spectra",  
                    "Bands,DataTrack,DataXtrack", " ", H5T_NATIVE_FLOAT);
```

FORTRAN integer function he5\_zadefine(*zaid*, *za\_name*, *dimlist*, *maxdimlist*, *dtype*)

```
integer      zaid
```

```
character*(*) za_name
```

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

```
character*(*) maxdimlist
```

```
integer      dtype
```

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

```
parameter (HE5T_NATIVE_FLOAT=10)  
status = he5_zadefine(zaid, "Spectra", "DataXtrack,  
DataTrack, Bands", " ", HE5T_NATIVE_FLOAT)
```

# Detach from a Zonal Average Structure

---

## HE5\_ZAdetach

herr\_t HE5\_ZAdetach(hid\_t *zaID*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

Purpose Detaches from zonal average interface.

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

Description This routine should be run before exiting from the za file for every za opened by *HE5\_ZAcreate* or *HE5\_ZAattach*.

Example In this example, we detach the za structure, *ExampleZA*:

```
status = HE5_ZAdetach(zaID);
```

FORTTRAN integer function he5\_zadetach(zaid)

```
integer      zaid
```

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

```
status = he5_zadetach(zaid)
```

# Retrieve Size of Specified Dimension

---

## HE5\_ZAdiminfo

hsize\_t HE5\_ZAdiminfo(hid\_t *zaID*, char \**dimname*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*dimname* IN: Dimension name

Purpose Retrieve size of specified dimension.

Return value Size of dimension if successful or 0 otherwise. A typical reason for failure is an improper ZA ID or dimension name.

Description This routine retrieves the size of specified dimension.

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

```
dimsize = HE5_ZAdiminfo(zaID, "DataTrack");
```

The return value, *dimsize*, will be equal to 4000.

FORTRAN integer\*4 function he5\_zadiminfo(*zaid*,*dimname*)

integer *zaid*

character\*(\*) *dimname*

integer\*4 *dimsize*

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

```
dimsize = he5_zadiminfo(zaid, "DataTrack")
```

# Remove an Alias for Zonal Average Data Field

---

## HE5\_ZAdropalias

herr\_t HE5\_ZAdropalias(hid\_t *zaID*, int *fldgroup*, const char \**aliasname*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fldgroup* IN: Field group flag

*aliasname* IN: Name of alias to remove

Purpose Remove an alias for Zonal Average data field

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

Description Removes alias associated with a Zonal Average data field.

Example In this example, we create and alias for the data field *Temperature*.

```
strcpy(aliasname, "temps 0 to 30");
```

```
status = HE5_ZAdropalias(zaID, HE5_HDFE_DATAGROUP,  
aliasname);
```

FORTTRAN integer function he5\_zadropalias (*zaid*, *fldgroup*, *aliasname*)

integer *zaid*

character\*(\*) *fldgroup*

character\*(\*) *aliasname*

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

```
aliasname = "temps 0 to 30"
```

```
status = he5_zadropalias(zaid, HE5_HDFE_DATAGROUP,  
aliasname)
```

# Rename Zonal Average Data Field

---

## HE5\_ZAfldrename

herr\_t HE5\_ZAfldrename(hid\_t *zaID*, char \**oldfieldname*, const char \**newfieldname*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*oldfieldname* IN: Current name of field

*newfieldname* IN: New name of field

Purpose Rename zonal average data field

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

Description This function allows the user to change the name of a field. This is useful in case the user would want to update the data field to reflect a version change in the calibration of a data field and show that in the name of the field.

Example In this example, we give a new name for the data field *Temperature*.

```
strcpy(newfieldname, "temps 0 to 30");
```

```
status = HE5_ZAfldrename(zaID, "Temperature", newfieldname);
```

FORTRAN integer function he5\_zafldrename (*zaid*, *oldfieldname*, *newfieldname*)  
integer *zaid*  
character\*(\*) *oldfieldname*  
character\*(\*) *newfieldname*

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

```
newfieldname = "temps 0 to 30"
```

```
status = he5_zafldrename(zaid, "Temperature", newfieldname)
```



# Retrieve Alias List for a ZA Data Fields Group

---

## HE5\_ZAgetaliaslist

long HE5\_ZAgetaliaslist(hid\_t *zaID*, int *fldgroup*, char \**aliaslist*, long \**strbufsize*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach  
*fldgroup* IN: Field group flag for “Data Fields” group  
*aliaslist* OUT: List of alias(es) in the “Data Fields” group (comma separated list)  
*strbufsize* OUT: Length of aliases list  
Purpose To retrieve the number and list of aliases in a Zonal Average structure.  
Return value Returns number of aliases in "Data Fields" group if successful or returns FAIL (-1) otherwise.  
Description Retrieves list of aliases in the “Data Fields” group (comma separated list) of a ZA and returns their number. The Data group flag is **HE5\_HDFE\_DATAGROUP**.

Example In this example, we get the alias list for the “data fields” group of a ZA structure.

```
/* first get the size of the list in bytes */
```

```
nalias = HE5_ZAgetaliaslist(zaID, HE5_HDFE_DATAGROUP, NULL, strbufsize);
```

```
aliaslist = (char *)malloc(strbufsize *sizeof(char));
```

```
nalias = HE5_ZAgetaliaslist(zaID, HE5_HDFE_DATAGROUP, aliaslist, strbufsize);
```

FORTRAN integer function he5\_gdgetaliaslist (*zaid*, *fldgroup*, *aliaslist*, *strbufsize*)  
integer *zaid*  
integer *fldgroup*  
integer *strbufsize*  
character\*(\*) *aliaslist*

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

```
integer nalias
```

```
nalias = he5_zagetaliaslist(zaid, HE5_HDFE_DATAGROUP, aliaslist, strbufsize)
```

# Get External Data File Information

---

## HE5\_ZAgetextdata

int HE5\_ZAgetextdata(hid\_t *zaid*, char \**fieldname*, size\_t *namelength*, char \**filelist*, off\_t *offset[]*, hsize\_t *size[]*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach  
*fieldname* IN: External field name  
*namelength* OUT: Length of each name entry  
*filelist* OUT: List of file names  
*offset[]* OUT: Array of offsets (in byte) from the beginning of file to the location in file where the data starts  
*size[]* OUT: Array of sizes (in bytes) reserved in the file for the data  
Purpose Retrieves information about external data file(s) associated with the data set.  
Return value Returns number of external data files if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper ZA ID or field name.

Example In this example, we get information about the *ExtData* field:

```
nfiles = HE5_ZAgetextdata(zaid, "ExtData", nmlen,  
filenames, offset, size);
```

FORTRAN integer function he5\_zagetxdat(*zaid*,*fieldname*,*nlen*, *flist*,*offset*, *size*)

```
integer      zaid  
integer      nfiles  
integer*4    nlen  
integer*4    offset(*)  
integer*4    size(*)  
character*(*) fieldname  
character*(*) flist
```

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

```
nfiles = he5_zagetxdat(zaid, "ExtData", nlen,  
flist, offset, size)
```

# Get Fill Value for a Specified Field

---

## HE5\_ZAgetfillvalue

herr\_t HE5\_ZAgetfillvalue(hid\_t *zaID*, char \**fieldname*, void \**fillval*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fieldname* IN: Fieldname

*fillval* OUT: Space allocated to store the fill value

Purpose Retrieves fill value for the specified field.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper ZA 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 = HE5_ZAgetfillvalue(zaID, "Temperature", &tempfill);
```

FORTRAN integer function he5\_zagetfill(*zaid*,*fieldname*,*fillval*)

integer *zaid*

character\*(\*) *fieldname*

<valid type> *fillval*(\*)

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

```
status = he5_zagetfill(zaid, "Temperature", tempfill)
```

# Return Information about a Group Zonal Average Attribute

---

## HE5\_ZAgrpattrinfo

herr\_t HE5\_ZAgrpattrinfo(hid\_t *zaid*, const char \**attrname*, hid\_t \**ntype*, hsize\_t \**count*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach  
*attrname* IN: Attribute name  
*numbertype* OUT: Number type of attribute. See Appendix A for interpretation of number types.  
*count* OUT: Number of attribute elements  
Purpose Returns information about a group attribute in the “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

Description This routine returns number type and number of elements (count) of a zonal average group attribute.

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

```
status = HE5_ZAgrpattrinfo(zaid, "ScalarFloat", &nt, &count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

FORTTRAN integer function he5\_zagattrinfo(*zaid*, *attrname*, *ntype*, *count*,)  
integer *zaid*  
character\*(\*) *attrname*  
integer *ntype*  
integer \*4 *count*

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

```
status = he5_zagattrinfo(zaid, "ScalarFloat",nt,count)
```

# Retrieve Information about a Zonal Average Field

---

## HE5\_ZAinfo

herr\_t HE5\_ZAinfo(hid\_t *zaID*, char \**za\_name*, int \**rank*, hsize\_t *dims*[], hid\_t *dtype*[], char \**dimlist*, char \**maxdimlist*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*za\_name* IN: Field name

*rank* OUT: Rank of field

*dims* OUT: Array containing the dimension sizes of the field

*dtype* OUT: Array containing the data type of the field

*dimlist* OUT: List of dimensions in field

*maxdimlist* OUT: List of maximum dimensions in field

Purpose Retrieve information about a specific data field in the ZA.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. A typical reason for failure is the specified field does not exist.

Description This routine retrieves information on a specific data field.

Example In this example, we retrieve information about the *Spectra* data fields:

```
status = HE5_ZAinfo(zaID, "Spectra", &rank, dims, dtype,
dimlist, maxdimlist);
```

The return parameters will have the following values:

*rank=3, dtype=10, 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 `he5_zainfo(zaid, za_name, rank, dims, dtype, dimlist, maxdimlist)`

integer `zaid`

character\*(\*) `za_name`

integer `rank`

integer\*4 `dims(*)`

integer `dtype(*)`

character\*(\*) `dimlist`

character\*(\*) `maxdimlist`

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

```
status = he5_zainfo(zaid, "Spectra", rank, dims, dtype,  
dimlist, maxdimlist)
```

The return parameters will have the following values:

*rank=3, dtype=10, dims[3]={2000,4000,5} and dimlist="DataXtrack,  
DataTrack,Bands"*

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

# Retrieve Information Zonal Average Attributes

---

## HE5\_ZAinqattrs

long HE5\_ZAinqattrs(hid\_t *zaid*, char \**attrnames*, long \**strbufsize*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*attrnames* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

Purpose Retrieve information about attributes defined in a specific ZA. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each attribute name separated by commas. If *attrnames* 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 zonal average structure. In the first call, set the parameter *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_ZAinqattrs(zaid, NULL, &strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_ZAinqattrs(zaid, attrnames, &strbufsize);
```

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

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

FORTTRAN integer\*4 function he5\_zainqattrs(*zaid*,*attrnames*,*strbufsize*)

```
integer      zaid
```

```
character*(*) attrnames
```

```
integer*4    strbufsize
```

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

```
nattr = he5_zainqattrs(zaid, attrnames, strbufsize)
```





integer\*4     *size*

character \*(\*) *fieldname*

integer        *HE5\_HDFE\_DATAGROUP*

parameter     (*HE5\_HDFE\_DATAGROUP=1*)

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

```
status = he5_zaidtype(zaid, "Spectra", " ",  
HE5_HDFE_DATAGROUP, dtype, classid, order, size)
```

# Retrieve Information about Dimensions Defined in Zonal Average

---

## HE5\_ZAinqdims

long HE5\_ZAinqdims(hid\_t *zaID*, char \**dimlist*, hsize\_t *dims*[])

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*dimlist* 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 zonal average.

Return value Number of dimension entries found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper ZA 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 *ExampleZA* structure:

```
ndims = HE5_ZAinqdims(zaID, dimlist, dims);
```

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

```
"MyTrack1,MyTrack2,DataTrack,DataXtrack,Bands,Unlim"
```

```
with ndims = 6, dims[6]={2000,1000,4000,2000,5,-1}
```

FORTRAN integer\*4 function he5\_zainqdims(*zaid*,*dimlist*,*dims*)

```
integer      zaid
```

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

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

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

```
ndims = he5_zainqdims(zaid, dimlist, dims)
```

# Retrieve Information about Data Fields and Aliases Defined in Zonal Average

---

## HE5\_ZAinqfldalias

long HE5\_ZAinqfldalias(hid\_t *zaID*, char \**fldalias*, long \**strbufsize*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fldalias* OUT: List of data fields and aliases (entries separated by commas)

*strbufsize* OUT: String length of data fields and aliases list

Purpose Retrieve information about data fields & aliases defined in zonal average.

Return value Number of data fields and aliases found if successful or FAIL (-1) otherwise.

Description The list of data fields and aliases is returned as a string with each name separated by commas. If *fldalias* 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 data fields and aliases defined for the “Data Fields” group. In the first call, set the parameter *fldalias* to NULL. We assume that there are one data field and one alias stored, *Temperature* and *Temp*:

```
nfldalias = HE5_ZAinqfldalias(zaID, NULL, &strbufsize);
```

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

```
fldalias = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nfldalias = HE5_ZAinqfldalias(zaID, fldalias, &strbufsize);
```

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

*"Temperature,Temp"*.

FORTRAN integer\*4 function he5\_zainqfldalias(*zaid*, *fldalias*, *strbufsize*)

integer *zaid*

character\*(\*) *fldalias*

integer\*4 *strbufsize*

integer\*4 *nfldalias*

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

```
nfldalias = he5_zainqfldalias(zaid, fldalias, strbufsize)
```

# Retrieve Information about Zonal Average Group Attributes

---

## HE5\_ZAinqgrpattrs

long HE5\_ZAinqgrpattrs(hid\_t *zaid*, char \**attrnames*, long \**strbufsize*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*attrnames* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

Purpose Retrieve information about group attributes defined in the “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each group attribute name separated by commas. If *attrnames* 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 group attributes defined for the “Data Fields” group. In the first call, set *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_ZAinqgrpattrs(zaid, NULL, &strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_ZAinqgrpattrs(zaid, attrnames, &strbufsize);
```

The variable, *attrnames*, will be set to: "*attrOne,attr\_2*".

FORTTRAN integer\*4 function he5\_zainqgattrs(*zaid*, *attrnames*, *strbufsize*)

integer *zaid*

character\*(\*) *attrnames*

integer\*4 *strbufsize*

integer\*4 *nattr*

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

```
nattr = he5_zainqgattrs(zaid, attrnames, strbufsize)
```

# Retrieve Information Zonal Average Local Attributes

---

## HE5\_ZAinqlocattr

long HE5\_ZAinqlocattr(hid\_t *zaid*, const char \**fieldname*, char \**attrnames*, long \**strbufsize*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fieldname* IN: Fieldname to retrieve local attribute information

*attrnames* OUT: Attribute list (entries separated by commas)

*strbufsize* OUT: String length of attribute list

Purpose Retrieve information about local attributes defined for a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Number of attributes found if successful or FAIL (-1) otherwise.

Description The attribute list is returned as a string with each local attribute name separated by commas. If *attrnames* 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 local attributes defined for a field "DataField". In the first call, set the parameter *attrnames* to NULL. We assume that there are two attributes stored, *attrOne* and *attr\_2*:

```
nattr = HE5_ZAinqlocattr(zaid, "DataField", NULL,
&strbufsize);
```

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

```
attrnames = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nattr = HE5_zainqlattr(zaid, "DataField", attrnames,
&strbufsize);
```

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

"*attrOne,attr\_2*".

FORTRAN integer\*4 function he5\_zainqlattr(*zaid*, *fieldname*, *attrnames*, *strbufsize*)

integer *zaid*

character\*(\*) *fieldname*

character\*(\*) *attrnames*

integer\*4     *strbufsize*

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

```
nattr = he5_zainqlattrs(zaid, "DataField", attrnames,  
strbufsize)
```

# Retrieve Information Defined in Zonal Average

---

## HE5\_ZAinquire

long HE5\_ZAinquire(hid\_t *zaID*, char \**za\_name\_list*, int *rank*[], hid\_t *dtype*[])

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*za\_name\_list* OUT: Listing of data fields (entries separated by commas)

*rank* OUT: Array containing the rank of each data field

*dtype* OUT: Array containing the data type of each data field

Purpose Retrieve information about all of the data fields defined in zonal average.

Return value Number of data fields found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper ZA id.

Description The field list is returned as a string with each data field separated by commas. The *rank* and *dtype* 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 = HE5_ZAinquire(zaID, za_name_list, rank,
                    dtype);
```

The parameter, *za\_name\_list*, will have the value:

"Spectra" with *rank*[1]={3}, *dtype*[1]={10}

FORTRAN integer\*4 function he5\_zainquire(*zaid*, *za\_name\_list*, *rank*, *dtype*)

integer *zaid*

character\*(\*) *za\_name\_list*

integer *rank*(\*)

integer *dtype*(\*)

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

```
nflds = he5_zainquire(zaid, za_name_list, rank, dtype)
```



# Retrieve Zonal Average Data Structures Defined in HDF-EOS File

---

## HE5\_ZAinqza

long HE5\_ZAinqza(const char \* *filename*, char \**zalist*, long \**strbufsize*)

*filename* IN: The HDF-EOS file name

*zalist* OUT: ZA list (entries separated by commas)

*strbufsize* OUT: String length of ZA list

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

Return value Number of ZAs found if successful or FAIL (-1) otherwise.

Description The ZA list is returned as a string with each za name separated by commas. If *zalist* 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 ZAs. Note that *strbufsize* does not count the null string terminator.

Example In this example, we retrieve information about the ZAs defined in an HDF-EOS file, *ZA.he5*. In the first call, set the parameter *zalist* to NULL. We assume that there are two ZAs stored, *zaOne* and *za\_2*:

```
nza = HE5_ZAinqza("ZA.he5", NULL, &strbufsize);
```

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

```
zalist = (char *)calloc(strbufsize+1, sizeof(char));
```

```
nza = HE5_ZAinqza("ZA.he5", zalist, &strbufsize);
```

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

```
"zaOne,za_2".
```

FORTTRAN integer\*4 function he5\_zainqza(*filename*,*zalist*,*strbufsize*)

character\*(\*) *filename*

character\*(\*) *zalist*

integer\*4 *strbufsize*

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

```
nza = he5_zainqza('za.he5', zalist, strbufsize)
```

# Return Information about a Local Zonal Average Attribute

---

## HE5\_ZAlocattrinfo

herr\_t HE5\_ZAlocattrinfo(hid\_t *zaid*, const char \**fieldname*, const char \**attrname*, hid\_t \**ntype*, hsize\_t \**count*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fieldname* IN: Field name

*attrname* OUT: Attribute name

*ntype* OUT: Number type of attribute

*count* OUT: Number of attribute elements

**Purpose** Returns information about local attribute(s) in a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

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

**Description** This routine returns number type and number of elements (*count*) of a data field's local attribute.

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

```
status = HE5_ZAlocattrinfo(zaid, "DataField", attrname,
&ntype, &count);
```

The *nt* variable will have the value 10 and *count* will have the value 1.

**FORTTRAN**

```
integer function he5_zalattrinfo(zaid, fieldname, attrname, ntype, count)
integer      zaid
character(*) attrname
integer      ntype
integer *4   count
```

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

```
status = he5_zalattrinfo(zaid, "DataField", attrname, ntype,
count)
```

# Mount External Data File

---

## HE5\_ZAmountexternal

hid\_t HE5\_ZAmountexternal(hid\_t *zaID*, int *fldgroup*, const char \**extfilename*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fldgroup* IN: Field group flag

*extfilename* IN: External file name

Purpose Mount external data file

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

Description This function allows the user to store required data needed by multiple data files into a separate file so it is not repeated throughout the data files.

Example In this example, we mount a file that contains calibration information needed by the data fields in another file

```
strcpy(extfilename, "/home/user/data/calibration.hdf5");  
  
fileID = HE5_ZAmountexternal(zaID, HE5_HDFE_DATAGROUP,  
extfilename);
```

FORTRAN Not available with this release.

# Return Number of Specified Objects in a Zonal Average

---

## HE5\_ZAnentries

long HE5\_ZAnentries(hid\_t *zaid*, int *entrycode*, long *\*strbufsize*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*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 successful or FAIL (-1) otherwise. A typical reason for failure is an improper ZA 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:

HE5\_HDFE\_NENTDIM (0) - Dimensions

HE5\_HDFE\_NENTMAP (1) - Dimension Mappings

HE5\_HDFE\_NENTIMAP (2) - Indexed Dimension

Mappings

HE5\_HDFE\_NENTDFLD (4) - Data Fields

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

```
ndflds = HE5_ZAnentries(zaid, HE5_HDFE_NENTDFLD, &bufsize);
```

The return value, *ndflds*, will be equal to 4 and *bufsz* = 39

FORTRAN integer\*4 function he5\_zanentries(*zaid*, *entrycode*, *bufsize*)

integer *zaid*

integer *entrycode*

integer\*4 *bufsize*

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

```
parameter (HE5_HDFE_NENTDFLD=4)
```

```
ndflds = he5_zanentries(zaid, HE5_HDFE_NENTDFLD, bufsize)
```

# Open HDF-EOS File

---

## HE5\_ZAopen

hid\_t HE5\_ZAopen(const char \*filename, uintn access)

*filename* IN: Complete path and filename for the file to be opened  
*access* IN: H5F\_ACC\_RDONLY, H5F\_ACC\_RDWR or H5F\_ACC\_TRUNC  
Purpose Opens or creates HDF-EOS file in order to create, read, or write a ZA.  
Return value Returns the ZA file id handle (fid) if successful or FAIL (-1) otherwise.  
Description This routine creates a new file or opens an existing one, depending on the access parameter.

Access codes:

H5F\_ACC\_RDONLY Open for read only. If file does not exist, error  
H5F\_ACC\_RDWR Open for read/write. If file does not exist, error  
H5F\_ACC\_TRUNC If file exists, delete it, then open a new file for read/write

Example In this example, we create a new ZA file named, *ZA.he5*. It returns the file handle, *fid*.

```
fid = HE5_ZAopen("ZA.he5", H5F_ACC_TRUNC);
```

FORTRAN integer function he5\_zapopen(filename, access)

character\*(\*) filename

integer access

The access codes should be defined as parameters:

```
parameter (HE5F_ACC_RDWR = 100)
```

```
parameter (HE5F_ACC_RDONLY = 101)
```

```
parameter (HE5F_ACC_TRUNC = 102)
```

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

```
fid = he5_zapopen("za.he5", HE5F_ACC_TRUNC)
```

**Note to users of the SDP Toolkit:** Please refer to the *Release 7 SDP Toolkit User Guide for the ECS Project (333-EMD-001 revision 05)*, 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.

# Read Data from a Zonal Average Field

---

## HE5\_ZAread

herr\_t HE5\_ZAread(hid\_t *zaID*, char \**za\_name*, const hssize\_t *start*[], const hsize\_t *stride*[], const hsize\_t *count*[], void \**datbuf*)

<i>zaID</i>	IN: ZA ID returned by HE5_ZAcreate or HE5_ZAattach
<i>za_name</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>count</i>	IN: Array specifying the number of values to read along each dimension
<i>datbuf</i>	OUT: Buffer to store the data read from the field
Purpose	Reads data from a zonal average field.
Return value	Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are improper ZA id or unknown fieldname.
Description	The values within <i>start</i> , <i>stride</i> , and <i>count</i> arrays refer to the zonal average field (input) dimensions. The output data in <i>datbuf</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>count</i> are $(dim - start) / stride$ where <i>dim</i> refers is the size of the dimension. Note that to allocate a string buffer size for reading an array of strings, first using <i>HE5_ZAreadlocattr</i> to get the value of maximum string length in the local attribute <i>StringLengthAttribute</i> .

Example In this example, we read data from the *Spectra* field.

```
float      plane[15][40][20];
hssize_t   start[3] = {0, 0, 0};
hsize_t    count[3] = {15, 40, 20};

status = HE5_ZAread(zaID, "Spectra", start, NULL, count,
plane);
```

FORTRAN integer function

he5\_zaread(*zaid*, *za\_name*, *start*, *stride*, *count*, *datbuf*)

he5\_zareadchar(*zaid*, *za\_name*, *elemlen*, *numelem*, *start*, *stride*, *count*,  
*datbuf*)

integer *zaid*

character\*(\*) *za\_name*

integer *elemlen* (*each element length in array of string*)

integer *numelem* (*number of elements in declared buffer array*)

integer\*4 *start*(\*)

integer\*4 *stride*(\*)

integer\*4 *count*(\*)

<valid type> *datbuf*(\*)

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

**Note: he5\_zareadchar() is only for reading an array of character string field. For reading an array of single character field, please use he5\_zaread().**

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

```
real*4      plane(800)
integer*4   start(3), stride(3), count(3)

start(1) = 0
start(2) = 0
stride(1) = 1
stride(2) = 1
stride(3) = 1
count(1) = 20
count(2) = 40
count(3) = 1

status=he5_zaread(zaid, "Spectra", start, stride,
count, plane)
```

# Read Zonal Average Attribute

---

## HE5\_ZAreadattr

herr\_t HE5\_ZAreadattr(hid\_t *zaID*, const char \**attrname*, void \**datbuf*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads attribute from a specific ZA. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper ZA 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 floating point attribute with the name "ScalarFloat":

```
status = HE5_ZAreadattr(zaID, "ScalarFloat", &data);
```

FORTTRAN integer function he5\_zardattr(*zaid*,*attrname*,*datbuf*)

integer *zaid*

<valid type> *datbuf*(\*)

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

```
status = he5_zardattr(zaid, "ScalarFloat", datbuf)
```



# Read External Data Set

---

## HE5\_ZAreadexternal

herr\_t HE5\_ZAreadexternal(hid\_t *zaID*, int *fldgroup*, const char \**fieldname*, void \**buffer*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fldgroup* IN: Field group flag

*fieldname* IN: Name of field to read

*buffer* OUT: Output data buffer

Purpose Read external data set

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

Description This function allows the user to get the data required from the external data file.

Example In this example, the field “Cal data” is read from the external file:

```
strcpy(fieldname, "Cal data");  
  
status = HE5_ZAreadexternal(zaID, HE5_HDFE_DATAGROUP,  
fieldname, buffer);
```

FORTRAN Not available with this release.

## Read Group Zonal Average Attribute

---

### HE5\_ZAreadgrpattr

herr\_t HE5\_ZAreadgrpattr(hid\_t *zaID*, const char \**attrname*, void \**datbuf*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads group attribute from the “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper ZA 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 floating point attribute with the name "ScalarFloat":

```
status = HE5_ZAreadgrpattr(zaID, "ScalarFloat", &data);
```

FORTTRAN integer function he5\_zardgattr(*zaid*,*attrname*,*datbuf*)

```
integer      zaid
```

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

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

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

```
status = he5_zardgattr(zaid, "ScalarFloat", datbuf)
```

# Read Local Zonal Average Attribute

---

## HE5\_ZAreadlocattr

herr\_t HE5\_ZAreadlocattr(hid\_t *zaID*, const char \**fieldname*, const char \**attrname*, void \**datbuf*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fieldname* IN: Field name

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads local attribute from a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper ZA 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 = HE5_ZAreadlocattr(zaID, "DataField", "ScalarFloat",
&data);
```

FORTTRAN integer function he5\_zardlatr(*zaid*, *fieldname*, *attrname*, *datbuf*)

integer *zaid*

character\*(\*) *fieldname*

character\*(\*) *attrname*

<valid type> *datbuf*(\*)

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

```
status = he5_zardlatr(zaid, "DataField", "ScalarFloat",
datbuf)
```

# Create an Alias for Zonal Average Data Field

---

## HE5\_ZAsetalias

herr\_t HE5\_ZAsetalias(hid\_t *zaid*, char \**fieldname*, const char \**aliaslist*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fieldname* IN: Field name

*aliaslist* IN: List of alias(es) to associate with the Data Field

Purpose Create an alias for Zonal Average data field

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

Description Creates aliases that can be used to refer to a Zonal Average data field in addition to the name of the field.

Example In this example, we create an alias for the data field *Temperature*.

```
strcpy(aliaslist, "temps 0 to 30");  
status = HE5_ZAsetalias(zaid, "Temperature", aliaslist);
```

FORTRAN integer function he5\_zasetalias (*zaid*, *fieldname*, *aliaslist*)  
integer *zaid*  
character\*(\*) *fieldname*  
character\*(\*) *aliaslist*

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

```
aliaslist = "temps 0 to 30"  
status = he5_zasetalias(zaid, "Temperature", aliaslist)
```

## Set External Data File(s)

---

### HE5\_ZAsetextdata

herr\_t HE5\_ZAsetextdata(hid\_t *zaID*, const char \**filelist*, off\_t *offset[]*, hsize\_t *size[]*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*filelist* IN: List of external file names

*offset[]* IN: Array of offsets (in byte) from the beginning of file to the location in file where the data starts

*size[]* IN: Array of sizes (in bytes) reserved in the file for the data

Purpose Sets the external data file(s) associated with the data set.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper ZA ID.

Example In this example, we set the *ExtData* field:

```
status = HE5_ZAsetextdata(zaID, "ext-1.dat,ext-2.dat,ext-3.dat", offset, size);
```

FORTRAN integer function he5\_zasetxdat(*zaid*,*fllist*,*offset*, *size*)

integer *zaid*

integer *status*

integer\*4 *offset*(\*)

integer\*4 *size*(\*)

character\*(\*) *fllist*

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

```
status = he5_zasetxdat(zaid,fllist,offset,size)
```

## Set Fill Value for a Specified Field

---

### HE5\_ZAsetfillvalue

herr\_t HE5\_ZAsetfillvalue(hid\_t *zaid*, char \**fieldname*, hid\_t *ntype*, void \**fillvalue*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach  
*fieldname* IN: Field name  
*ntype* IN: Number type of fill value (should match the number type of a specified field)  
*fillvalue* IN: Pointer to the fill value to be used

**NOTE: THIS FUNCTION MUST BE CALLED BEFORE THE FUNCTION CALL TO DEFINE THE FIELD IT IS TO BE APPLIED. SETS A FILL VALUE FOR A CHARACTER STRING FIELD IS NOT AVAILABLE IN THIS RELEASE.**

Purpose Sets fill value for the specified field.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper ZA 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 = HE5_ZAsetfillvalue(zaid, "Temperature", ntype,  
&tempfill);
```

FORTRAN integer function he5\_zasetfill(*zaid*, *fieldname*, *ntype*, *fillvalue*)

```
integer      zaid  
character*(*) fieldname  
integer      ntype  
<valid type> fillvalue(*)
```

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

```
fillvalue = -999.0  
  
status = he5_zasetfill(zaid, "Temperature", ntype, fillvalue)
```

## Dismount External Data File

---

### HE5\_ZAunmount

herr\_t HE5\_ZAunmount(hid\_t *zaID*, int *fldgroup*, hid\_t *fileID*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fldgroup* IN: Field group flag

*fileID* IN: ID of file returned by HE5\_ZAmountexternal

Purpose Dismount external data file

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

Description This function dismounts from the external file once the user has completed using the data in the file.

Example In this example, we dismount from the file used in the previous function

```
status = HE5_ZAunmount(zaID, HE5_HDFE_DATAGROUP, fileID);
```

FORTRAN Not available with this release.

# Write Data to a Zonal Average Field

---

## HE5\_ZAwrite

herr\_t HE5\_ZAwrite(hid\_t *zaID*, char \**za\_name*, const hssize\_t *start*[], const hsize\_t *stride*[], const hsize\_t *count*[], void \**datbuf*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach  
*za\_name* IN: Name of field to write  
*start* IN: Array specifying the starting location within each dimension (0-based)  
*stride* IN: Array specifying the number of values to skip along each dimension  
*count* IN: Array specifying the number of values to write along each dimension  
*datbuf* IN: Values to be written to the field

Purpose Writes data to a zonal average field.

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

Description The values within *start*, *stride*, and *count* arrays refer to the zonal average field (output) dimensions. The input data in the *datbuf* 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 *count* 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 *Spectra* field.

```
float plane [15][40][20];  
  
/* Define elements of plane array */  
  
hssize_t start[3]={0,0,0}; hsize_t count[3]={15,40,20};  
  
status = HE5_ZAwrite(zaID, "Spectra", start, NULL, count,  
plane);
```

FORTRAN integer function

```
he5_zawrite(zaid,za_name,start,stride,count,datbuf)
```



he5\_zawritechar(zaid,za\_name,elemLen,numelem,start,stride,count,datbuf)

integer        zaid

character\*(\*) za\_name

integer        elemLen (each element length in array of string)

integer        numelem (number of elements in declared buffer array)

integer\*4     start(\*)

integer\*4     stride(\*)

integer\*4     count(\*)

<valid type> datbuf(\*)

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

**Note: he5\_zawritechar() is only for writing an array of character string field. For writing an array of single character field, please use he5\_zawrite().**

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

```
real*4     plane(800)

integer*4 start(3), stride(3), count(3)

start(1) = 0

start(2) = 0

stride(1) = 1

stride(2) = 1

stride(3) = 1

count(1) = 20

count(2) = 40

count(3) = 1

status = he5_zawrite(zaid, "Spectra", start, stride, count,
plane)
```

# Write/Update Zonal Average Attribute

---

## HE5\_ZAwriteattr

herr\_t HE5\_ZAwriteattr(hid\_t *zaid*, const char \**attrname*, hid\_t *ntype*, hsize\_t *count*[], void \**datbuf*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*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 specific ZA. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper ZA 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 floating point number with the name "ScalarFloat" and the value 3.14:

```
attr_val = 3.14;
```

```
status = HE5_ZAwriteattr(zaid, "ScalarFloat",  
H5T_NATIVE_FLOAT, 1, &attr_val);
```

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

```
attr_val = 3.14159;
```

```
status = HE5_ZAwriteattr(zaid, "ScalarFloat",  
H5T_NATIVE_FLOAT, 1, &attr_val);
```

FORTRAN integer function `he5_zawratr(zaid, attrname, ntype, count, datbuf)`  
integer `zaid`  
character\*(\*) `attrname`  
integer\*4 `count(*)`  
<valid type> `datbuf(*)`

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

```
parameter (HE5T_NATIVE_FLOAT = 10)
datbuf = 3.14
count = 1
status = he5_zawratr(zaid, "ScalarFloat",
HE5T_NATIVE_FLOAT, count, datbuf)
```

# Write Field Metadata for an Existing Zonal Average Data Field

---

## HE5\_ZAwritedatameta

herr\_t HE5\_ZAwritedatameta(hid\_t *zaID*, const char \**fieldname*, char \**dimlist*, int *mvalue*)

*zaID* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fieldname* IN: Name of field

*dimlist* IN: The list of data dimensions defining the field

*mvalue* IN: The number type of the data stored in the field

Purpose Writes field metadata for an existing zonal average data field.

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

Description This routine writes field metadata for an existing data field. This is useful when the data field was defined without using the zonal average API. Note that any entries in the dimension list must be defined through the *HE5\_ZAdefdim* routine before this routine is called.

Example In this example we write the metadata for the “Band\_1” data field used in the zonal average.

```
status = HE5_ZAwritedatameta(zaID, "Band_1", "DataTrack,  
DataXtrack", H5T_NATIVE_FLOAT);
```

FORTRAN integer function

```
he5_zawrdmeta(zaid,fieldname,dimlist,mvalue)
```

```
integer zaid
```

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

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

```
integer mvalue
```

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

```
parameter (HE5T_NATIVE_FLOAT = 10)  
status = he5_zawrdmeta(zaID, "Band_1", "DataTrack,  
DataXtrack", HE5T_NATIVE_FLOAT)
```

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

# Write/Update Group Zonal Average Attribute

---

## HE5\_ZAwritegrpattr

herr\_t HE5\_ZAwritegrpattr(hid\_t *zaid*, const char \**attrname*, hid\_t *ntype*,  
hsize\_t *count*[], void \**datbuf*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*attrname* IN: Attribute name

*ntype* IN: Data type of attribute

*count* IN: Number of values to store in attribute

*datbuf* IN: Attribute values

**Purpose** Writes/Updates group attribute in the “Data Fields” group. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

**Return value** Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper ZA 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. The attribute is linked to the “Data Fields” group in the zonal average file.

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

```
count[0] = 1;  
attr_val = 3.14;  
  
status = HE5_ZAwritegrpattr(zaid, "ScalarFloat",  
H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```
attr_val = 3.14159;  
  
status = HE5_ZAwritegrpattr(zaid, "ScalarFloat",  
H5T_NATIVE_FLOAT, count, &attr_val);
```

FORTRAN integer function he5\_zawrgattr(*zaid*, *attrname*, *ntype*, *count*, *datbuf*)

integer        *zaid*  
character\*(\*) *attrname*  
integer        *ntype*  
integer\*4     *count*(\*)  
<valid type> *datbuf*(\*)

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

```
parameter (HE5T_NATIVE_FLOAT=10)
datbuf = 3.14
count = 1
status = he5_zawrgattr(zaid, "ScalarFloat",
HE5T_NATIVE_FLOAT, count, datbuf)
```

# Write/Update Local Zonal Average Attribute

---

## HE5\_ZAwritelocatrr

herr\_t HE5\_ZAwritelocatrr(hid\_t *zaid*, const char \**fieldname*, const char \**attrname*, hid\_t *ntype*, hsize\_t *count*[], void \**datbuf*)

*zaid* IN: ZA ID returned by HE5\_ZAcreate or HE5\_ZAattach

*fieldname* IN: Field name

*attrname* IN: Attribute name

*ntype* IN: Data type of attribute

*count* IN: Number of values to store in attribute

*datbuf* IN: Attribute values

Purpose Writes/Updates local attribute in a specific field. See Section 3.6 of Volume 1 (Different Types of Attributes in HDF-EOS5).

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper ZA 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. The attribute is linked to a particular "Data Field" in the zonal average file.

Example In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14:

```
count[0] = 1;
attr_val = 3.14;

status = HE5_ZAwritelocatrr(zaid, "DataField",
"ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```
attr_val = 3.14159;

status = HE5_ZAwritelocatrr(zaid, "DataField",
"ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

FORTRAN integer function `he5_zawrlattr(zaid, fieldname, attrname, ntype, count, datbuf)`

integer `zaid`

character\*(\*) `fieldname`

character\*(\*) `attrname`

integer `ntype`

integer\*4 `count(*)`

<valid type> `datbuf(*)`

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

```
parameter (HE5T_NATIVE_FLOAT=10)

datbuf = 3.14

count = 1

status = he5_zawrlattr(zaid, "DataField", "ScalarFloat",
HE5T_NATIVE_FLOAT, count, datbuf)
```



# Convert Grid Coordinates (i,j) to (Longitude, Latitude)

---

## HE5\_GDij2ll

intn HE5\_GDij2ll(int *projcode*, int *zonecode*, float64 *projparm*[], int *spherecode*, long *xdimsize*, long *ydimsize*, float64 *upleft*[], float64 *lowright*[], long *npnts*, long *row*[], long *col*[], float64 *longitude*[], float64 *latitude*[], int *pixcen*, int *pixcnr*)

<i>projcode</i>	IN:	GCTP projection code
<i>zonecode</i>	IN:	GCTP zone code used by UTM projection
<i>projparm</i>	IN:	Projection parameters
<i>spherecode</i>	IN:	GCTP spherecode
<i>xdimsize</i>	IN:	xdimsize from HE5_GDgridinfo()
<i>ydimsize</i>	IN:	ydimsize from HE5_GDgridinfo()
<i>upleft</i>	IN:	Upper left corner of the grid in meter (all projections except Geographic) or DMS degree (Geographic projection), values from HE5_GDgridinfo()
<i>lowright</i>	IN:	Lower right corner of the grid in meter or DMS degree, Geographic) or DMS degree (Geographic projection), values from HE5_GDgridinfo()
<i>npnts</i>	IN:	number of lon-lat points
<i>row</i>	IN:	row numbers of the pixels (zero based)
<i>col</i>	IN:	column numbers of the pixels (zero based)
<i>pixcen</i>	IN:	Code from HE5_GDpixreginfo
<i>pixcnr</i>	IN:	Code from HE5_GDoriginfo
<i>longitude</i>	OUT:	longitude array (decimal degrees)
<i>latitude</i>	OUT:	latitude array (decimal degrees)
Purpose		Converts a grid's (i,j) coordinates to longitude and latitude.
Return value		Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description		This routine converts any grid's (i,j) coordinates to longitude and latitude in decimal degrees.

Example

```

int          gridid, npnts = 2;
int          projcode, origincode, pixregcode, zonecode, spherecode;
float64      upleft[2], lowright[2];
float64      projparm[13];
long         cols[2], rows[2] ;
float64      lon[2], lat[2];
long         xdimsize, ydimsize;
cols[0]= 10;
rows[0]= 14;
cols[1]= 17;
rows[1]= 9;

status = HE5_GDprojinfo(gridid, &projcode, &zonecode, &spherecode,
projparm);

status = HE5_GDgridinfo(gridid, &xdimsize, &ydimsize, upleft,
lowright);

status = HE5_GDpixreginfo(gridid, &pixregcode);

status = HE5_GDorigininfo(gridid, &origincode);

status = HE5_GDij2ll(projcode, zonecode, projparm, spherecode,
xdimsize, ydimsize, upleft, lowright, npnts, rows, cols, lon, lat,
pixregcode, origincode);

```

FORTRAN integer function he5\_gdij2ll( projcode, zonecode, projparm, spherecode, xdimsize, ydimsize, upleft, lowright, npnts, rows, cols, longitude, latitude, pixregcode, origincode)

```

integer      projcode, pixregcode, origincode, zonecode, spherecode
real*8      projparm(*)
integer      xdimsize, ydimsize, npnts
integer      cols(*), rows(*)
real*8      longitude(*), latitude(*)
real*8      upleft(2), lowright(2)

```

The Equivalent FORTRAN code for the example above is:

```

npnts = 2
cols(1)= 10

```

rows(1)= 14

cols(2)= 17

rows(2)= 9

status = he5\_gdprojinfo(gridid, projcode, zonecode, spherecode,  
projparm)

status = he5\_gdgridinfo(gridid, xdimsize, ydimsize, upleft, lowright)

status = he5\_gdpixreginfo(gridid, pixregcode)

status = he5\_gdorigininfo(gridid, origincode)

status = he5\_gdij2ll(projcode, zonecode, projparm, spherecode, xdimsize,

& ydimsize, upleft, lowright, npnts, rows, cols, longitude, latitude,  
pixregcode, origincode)

# Convert Grid Coordinates (Longitude, Latitude) to (i,j)

---

## HE5\_GDl12ij

intn HE5\_GDl12ij(int *projcode*, int *zonecode*, float64 *projparm*[], int *spherecode*, long *xdimsize*, long *ydimsize*, float64 *upleft*[], float64 *lowright*[], int *npnts*, float64 *longitude*[], float64 *latitude*[], long *row*[], long *col*[], float64 *xval*[], float64 *yval*[])

<i>projcode</i>	IN:	GCTP projection code
<i>zonecode</i>	IN:	GCTP zone code used by UTM projection
<i>projparm</i>	IN:	Projection parameters
<i>spherecode</i>	IN:	GCTP spherecode
<i>xdimsize</i>	IN:	xdimsize from HE5_GDgridinfo()
<i>ydimsize</i>	IN:	ydimsize from HE5_GDgridinfo()
<i>upleft</i>	IN:	Upper left corner of the grid in meter (all projections except Geographic) or DMS degree (Geographic projection), values from HE5_GDgridinfo()
<i>lowright</i>	IN:	Lower right corner of the grid in meter or DMS degree, Geographic) or DMS degree (Geographic projection), values from HE5_GDgridinfo()
<i>npnts</i>	IN:	number of lon-lat points
<i>longitude</i>	IN:	longitude array (decimal degrees)
<i>latitude</i>	IN:	latitude array (decimal degrees)
<i>row</i>	OUT:	row numbers of the pixels (zero based)
<i>col</i>	OUT:	column numbers of the pixels (zero based)
<i>xval</i>	OUT:	x array
<i>yval</i>	OUT:	y array
Purpose		Converts pixel's longitude and latitude to its (i,j) coordinates
Return value		Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description		This routine converts longitude and latitude pair (in decimal degrees) of any pixel in grid to its (i,j) coordinates. In addition it outputs the x, y position (scaled distances) of the point in the grid.

Example

```

int          gridid, npnts = 2;
int          projcode, origincode, pixregcode, zonecode, spherecode ;
float64     upleft[2], lowright[2];
float64     projparm[13];
long        xcord[2], ycord[2];
float64     cols[2], rows[2], lon[2], lat[2];
long        xdimsize, ydimsize;
lat[0]= 48.0;
lon[0]= -120.0;
lat[1]= 34.0;
lon[1]= -110.0;

status = HE5_GDprojinfo(gridid, &projcode, &zonecode, &spherecode,
projparm);

status = HE5_GDgridinfo(gridid, &xdimsize, &ydimsize, upleft,
lowright);

status = HE5_GDpixreginfo(gridid, &pixregcode);

status = HE5_GDorigininfo(gridid, &origincode);

status = HE5_GDl12ij(projcode, zonecode, projparm, spherecode,
xdimsize, ydimsize, upleft, lowright, npnts, lon, lat, , rows, cols, xcord,
ycord);

```

FORTRAN integer function he5\_gdll2ij( projcode, zonecode, projparm, spherecode, xdimsize, ydimsize, upleft, lowright, npnts, longitude, latitude, row, col, xcord, ycord )

```

integer     projcode, pixregcode, origincode, zonecode, spherecode
real*8     projparm(*)
integer     xdimsize, ydimsize, npnts
integer     row(*), col(*)
real*8     longitude(*), latitude(*), xcord(*), ycord(*)
real*8     upleft(2), lowright(2)

```

The Equivalent FORTRAN code for the example above is:

```

npnts = 2
lat(1)= 48.0

```

lon(1)= -120.0

lat(2)= 34.0

lon(2)= -110.0

status = he5\_gdprojinfo(gridid, projcode, zonecode, spherecode,  
projparm)

status = he5\_gdgridinfo(gridid, xdimsize, ydimsize, upleft, lowright)

status = he5\_gdpixreginfo(gridid, pixregcode)

status = he5\_gdorigininfo(gridid, origincode)

status = he5\_gdll2ij(projcode, zonecode, projparm, spherecode, xdimsize,

& ydimsize, upleft, lowright, npnts, lon, lat, row, col, xcord, ycord)

# Convert EASE Grid Coordinates (r,s) to (longitude, latitude)

---

## HE5\_GDrs2ll

herr\_t HE5\_GDrs2ll(int *projcode*, double *projparm*[], long *xdimsize*, long *ydimsize*, double *upleft*[], double *lowright*[], int *npnts*, double *r*[], double *s*[], double *longitude*[], double *latitude*[], int *pixcen*, int *pixcnr*)

<i>projcode</i>	IN:	GCTP projection code (HE5_GCTP_BCEA)
<i>projparm</i>	IN:	Projection parameters array
<i>xdimsize</i>	IN:	xdimsize from HE5_GDgridinfo()
<i>ydimsize</i>	IN:	ydimsize from HE5_GDgridinfo()
<i>upleft</i>	IN:	Upper left corner lon/lat of the grid in DMS format, value from HE5_GDgridinfo()
<i>lowright</i>	IN:	Lower right corner lon/lat of the grid in DMS format, value from HE5_GDgridinfo()
<i>npnts</i>	IN:	Number of lon-lat points
<i>r</i>	IN:	Array of EASE grid's r coordinate
<i>s</i>	IN:	Array of EASE grid's s coordinate
<i>pixcen</i>	IN:	Code from HE5_GDpixreginfo()
<i>pixcnr</i>	IN:	Code from HE5_GDorigininfo()
<i>longitude</i>	OUT:	longitude array (decimal degrees)
<i>latitude</i>	OUT:	latitude array (decimal degrees)
Purpose		Converts EASE grid's (r,s) coordinates to longitude and latitude.
Return Value		Returns SUCCEED(0) if successful or FAIL(-1) otherwise.
Description		This routine converts EASE grid's (r,s) coordinates to longitude and latitude in decimal degrees.

### Example

```
hid_t gridID;  
  
int projcode, origincode, pixregcode, npnts = 2;
```

```

double upleft[2], lowright[2], projparm[13];
double rcord[2], scord[2], lon[2], lat[2];
long  xdimsize, ydimsize;
rcord[0] = 0.;
scord[0] = 0.;
rcord[1] = 691.5;
scord[1] = 293.;
status = HE5_GDprojinfo(gridID, HE5_GCTP_BCEA, 0, 0,
projparm);
status = HE5_GDgridinfo(gridID, xdimsize, ydimsize, upleft,
lowright);
status = HE5_GDpixreginfo(gridID, &pixregcode);
status = HE5_GDorigininfo(gridID, &origincode);
status = HE5_GDrs2ll(HE5_GCTP_BCEA, projparm, xdimsize,
ydimsize, upleft, lowright, npnts, rcord, scord, lon, lat,
pixregcode, origincode);

```

FORTRAN integer function *he5\_gdrs2ll*(*HE5\_GCTP\_BCEA*, *projparm*, *xdimsize*,  
*ydimsize*, *upleft*, *lowright*, *npnts*, *r*, *s*, *longitude*, *latitude*, *pixregcode*,  
*origincode*)

```

integer      gridid
integer      projcode, pixregcode, origincode, npnts
real*8      projparm(*)
integer*4    xdimsize, ydimsize
real*8      r(*), s(*), longitude(*), latitude(*)
real*8      upleft(2), lowright(2)

```

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

```

parameter (HE5_GCTP_BCEA=98)
npnts = 2
rcord(1) = 0.
scord(1) = 0.
rcord(2) = 691.5
scord(2) = 293.

```



```
status = he5_gdprojinfo(gridid, HE5_GCTP_BCEA, 0, 0,
projparm)

status = he5_gdgridinfo(gridid, xdimsize, ydimsize, upleft,
lowright)

status = he5_gdpreginfo(gridid, pixregcode)

status = he5_gdorginfo(gridid, origincode)

status = he5_gdrs2ll(HE5_GCTP_BCEA, projparm, xdimsize,
ydimsize, upleft, lowright, npnts, rcord, scord, longitude,
latitude, pixregcode, origincode)
```

This page intentionally left blank.

## Appendix A. Numbertype Codes

---

The HDF-EOS5 library predefines a number of commonly used datatypes with names that resemble their equivalent in HDF5. The numbertype codes as defined in `HE5_HdfEosDef.h` are shown in Table A1. These types have standard symbolic names of the form `HE5T_arch_base` where *arch* is an architecture name and *base* is a programming type name (Table A2). The base name of most types consists of a letter to indicate the class (Table A3), a precision in bits, and an indication of the byte order (Table A4). Table A5 shows examples of predefined datatypes.

<b>Table A1</b>			
HE5T_NATIVE_INT	0	HE5T_STD_I8LE	29
HE5T_NATIVE_UINT	1	HE5T_STD_I16BE	30
HE5T_NATIVE_SHORT	2	HE5T_STD_I16LE	31
HE5T_NATIVE_USHORT	3	HE5T_STD_I32BE	32
HE5T_NATIVE_SCHAR	4	HE5T_STD_I32LE	33
HE5T_NATIVE_UCHAR	5	HE5T_STD_I64BE	34
HE5T_NATIVE_LONG	6	HE5T_STD_I64LE	35
HE5T_NATIVE_ULONG	7	HE5T_STD_U8BE	36
HE5T_NATIVE_LLONG	8	HE5T_STD_U8LE	37
HE5T_NATIVE_ULLONG	9	HE5T_STD_U16BE	38
HE5T_NATIVE_FLOAT	10	HE5T_STD_U16LE	39
HE5T_NATIVE_REAL	10	HE5T_STD_U32BE	40
HE5T_NATIVE_DOUBLE	11	HE5T_STD_U32LE	41
HE5T_NATIVE_LDOUBLE	12	HE5T_STD_U64BE	42
HE5T_NATIVE_INT8	13	HE5T_STD_U64LE	43
HE5T_NATIVE_UINT8	14	HE5T_STD_B8BE	44
HE5T_NATIVE_INT16	15	HE5T_STD_B8LE	45
HE5T_NATIVE_UINT16	16	HE5T_STD_B16BE	46
HE5T_NATIVE_INT32	17	HE5T_STD_B16LE	47

<b>Table B1 (continued)</b>			
HE5T_NATIVE_UINT32	18	HE5T_STD_B32BE	48
HE5T_NATIVE_INT64	19	HE5T_STD_B32LE	49
HE5T_NATIVE_UINT64	20	HE5T_STD_B64BE	50
HE5T_NATIVE_B8	21	HE5T_STD_B64LE	51
HE5T_NATIVE_B16	22	HE5T_IEEE_F32BE	52
HE5T_NATIVE_B32	23	HE5T_IEEE_F32LE	53
HE5T_NATIVE_B64	24	HE5T_IEEE_F64BE	54
HE5T_NATIVE_HSIZE	25	HE5T_IEEE_F64LE	55
HE5T_NATIVE_HERR	26	HE5T_NATIVE_CHAR	56
HE5T_NATIVE_HBOOL	27	HE5T_CHARSTRING	57
HE5T_STD_I8BE	28		

<b>Table A2</b>	
<b>Architecture Name</b>	<b>Description</b>
IEEE	IEEE-754 standard floating point types in various byte orders.
STD	This is an architecture that contains semi-standard datatypes like signed two's complement integers, unsigned integers, and bitfields in various byte orders.
NATIVE	This architecture contains C-like datatypes for the machine on which the library was compiled.

<b>Table A3</b>	
B	Bitfield
F	Floating point
I	Signed integer
S	Character string
U	Unsigned integer

<b>Table A4</b>	
BE	Big endian
LE	Little endian
VX	Vax order

<b>Table A5</b>	
<b>Example</b>	<b>Description</b>
HE5T_IEEE_F64LE	Eight-byte, little-endian, IEEE floating-point
HE5T_IEEE_F32BE	Four-byte, big-endian, IEEE floating point
HE5T_STD_I32LE	Four-byte, little-endian, signed two's complement integer
HE5T_STD_U16BE	Two-byte, big-endian, unsigned integer
HE5T_NATIVE_B64	Native Eight-byte bit field

This page intentionally left blank.

# Abbreviations and Acronyms

---

AI&T	Algorithm Integration & Test
AIRS	Atmospheric Infrared Sounder
API	application program interface
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer
CCSDS	Consultative Committee on Space Data Systems
CDRL	Contract Data Requirements List
CDS	CCSDS day segmented time code
CERES	Clouds and Earth Radiant Energy System
CM	configuration management
COTS	commercial off-the-shelf software
CUC	constant and unit conversions
CUC	CCSDS unsegmented time code
DAAC	distributed active archive center
DBMS	database management system
DCE	distributed computing environment
DCW	Digital Chart of the World
DEM	digital elevation model
DTM	digital terrain model
ECR	Earth centered rotating
ECS	EOSDIS Core System
EDC	Earth Resources Observation Systems (EROS) Data Center
EDHS	ECS Data Handling System
EDOS	EOSDIS Data and Operations System
EOS	Earth Observing System
EOSAM	EOS AM Project (morning spacecraft series)
EOSDIS	Earth Observing System Data and Information System
EOSPM	EOS PM Project (afternoon spacecraft series)

ESDIS	Earth Science Data and Information System (GSFC Code 505)
FDF	flight dynamics facility
FOV	field of view
ftp	file transfer protocol
GCT	geo-coordinate transformation
GCTP	general cartographic transformation package
GD	grid
GPS	Global Positioning System
GSFC	Goddard Space Flight Center
HDF	hierarchical data format
HITC	Hughes Information Technology Corporation
http	hypertext transport protocol
I&T	integration & test
ICD	interface control document
IDL	interactive data language
IP	Internet protocol
IWG	Investigator Working Group
JPL	Jet Propulsion Laboratory
LaRC	Langley Research Center
LIS	Lightening Imaging Sensor
M&O	maintenance and operations
MCF	metadata configuration file
MET	metadata
MODIS	Moderate-Resolution Imaging Spectroradiometer
MSFC	Marshall Space Flight Center
NASA	National Aeronautics and Space Administration
NCSA	National Center for Supercomputer Applications
netCDF	network common data format
NGDC	National Geophysical Data Center
NMC	National Meteorological Center (NOAA)



ODL	object description language
PC	process control
PCF	process control file
PDPS	planning & data production system
PGE	product generation executive (formerly product generation executable)
POSIX	Portable Operating System Interface for Computer Environments
PT	point
QA	quality assurance
RDBMS	relational data base management system
RPC	remote procedure call
RRDB	recommended requirements database
SCF	Science Computing Facility
SDP	science data production
SDPF	science data processing facility
SGI	Silicon Graphics Incorporated
SMF	status message file
SMP	Symmetric Multi-Processing
SOM	Space Oblique Mercator
SPSO	Science Processing Support Office
SSM/I	Special Sensor for Microwave/Imaging
SW	swath
TAI	International Atomic Time
TBD	to be determined
TDRSS	Tracking and Data Relay Satellite System
TRMM	Tropical Rainfall Measuring Mission (joint US – Japan)
UARS	Upper Atmosphere Research Satellite
UCAR	University Corporation for Atmospheric Research
URL	universal reference locator
USNO	United States Naval Observatory
UT	universal time

UTC	Coordinated Universal Time
UTCf	universal time correlation factor
UTM	universal transverse mercator
VPF	vector product format
WWW	World Wide Web
ZA	Zonal Average