EED2-175-002

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

Technical Paper

December 2017

Prepared Under Contract #NNG15HZ39C

RESPONSIBLE ENGINEER

[Signature] 12/21/17

Abe Taaheri, Toolkit Task Lead  Date
EOSDIS Evolution and Development-2 (EED-2) Contract

SUBMITTED BY

[Signature] 12/21/17

Kristene Rodrigues, Task 32 Lead  Date
EOSDIS Evolution and Development-2 (EED-2) Contract

Raytheon Company
Riverdale, Maryland
This page intentionally left blank.
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 5.1.16. The software is based on HDF5, a new version of HDF provided by by The HDF Group. HDF5 is a complete rewrite of the earlier HDF4 version, containing a different data model and user interface. HDF-EOS V5.1.16 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 5.1.16 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 5.1.16 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 EOSDIS Evolution and Development-2 (EED-2) Contract, December 2017, 333-EED2-001, Revision 01).

HDF-EOS5 is an extension of The HDF Group (THG) HDF5 and uses HDF5 library calls as an underlying basis. Version 5-1.8.19 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. The HDF-EOS plug-in for HDFView, the HDF-EOS viewing tools, will be 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:

RVL_PGSTLKIT@raytheon.com

Any questions should be addressed to:

Data Management Office
EOSDIS Evolution and Development-2 (EED-2) Contract
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
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 ................................................................................................................. 1-7
   1.5.1 The Grid Data Interface .............................................................................. 1-7
   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 ............................................................................. 1-10
   1.6.4 GCTP Projection Parameters .................................................................... 1-12
1.7 Zonal Average Data .................................................................................................. 1-17
   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 ................................................................. 2-179
2.1.4 HDF-EOS Utility Routines ............................................................... 2-279
2.1.5 Zonal Average Interface Functions .................................................. 2-290

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 ........................................ 1-18

Appendix A. Numertype Codes

Abbreviations and Acronyms
1. Introduction

1.1 Purpose

The HDF-EOS Interface Based on HDF5, Volume 2: Function Reference Guide was prepared under the EOSDIS Evolution and Development-2 Contract (NNG15HZ39C).

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>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>FORTRAN</th>
<th>Description</th>
<th>Pg. Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>HE5_PTopen</td>
<td>he5_ptopen</td>
<td>Creates a new file or opens an existing one</td>
<td>2-32</td>
</tr>
<tr>
<td></td>
<td>HE5_PTcreate</td>
<td>he5_ptcreate</td>
<td>Creates a new point data set and returns a handle</td>
<td>2-06</td>
</tr>
<tr>
<td></td>
<td>HE5_PTattach</td>
<td>he5_ptattach</td>
<td>Attaches to an existing point data set</td>
<td>2-02</td>
</tr>
<tr>
<td></td>
<td>HE5_PTdetach</td>
<td>he5_ptdetach</td>
<td>Releases a point data set and frees memory</td>
<td>2-14</td>
</tr>
<tr>
<td></td>
<td>HE5_PTclose</td>
<td>he5_ptclose</td>
<td>Closes the HDF-EOS file and deactivates the point interface</td>
<td>2-05</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_PTdeflevel</td>
<td>he5_ptdeflevel</td>
<td>Defines a level within the point data set</td>
<td>2-07</td>
</tr>
<tr>
<td></td>
<td>HE5_PTdeflinkage</td>
<td>he5_ptdeflinkage</td>
<td>Defines link field to use between two levels</td>
<td>2-13</td>
</tr>
<tr>
<td>Basic I/O</td>
<td>HE5_PTwritenlevel</td>
<td>he5_ptwritenlevel</td>
<td>Writes (appends) full records to a level</td>
<td>2-42</td>
</tr>
<tr>
<td></td>
<td>HE5_PTreaddlevel</td>
<td>he5_ptreaddlevel</td>
<td>Reads data from the specified fields and records of a level</td>
<td>2-36</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwriteñafield</td>
<td>he5_ptwriteñafield</td>
<td>Updates the specified fields and records of a level</td>
<td>2-37</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwritegrpatr</td>
<td>he5_ptwritegrpatr</td>
<td>Creates or updates an attribute of the point data set</td>
<td>2-38</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwritegrpatr</td>
<td>he5_ptwritegrpatr</td>
<td>Writes/updates group attribute in a point</td>
<td>2-40</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwritelocalattr</td>
<td>he5_ptwritelocalattr</td>
<td>Write/updates local attribute in a point</td>
<td>2-43</td>
</tr>
<tr>
<td></td>
<td>HE5_PTreaddirattr</td>
<td>he5_ptreaddirattr</td>
<td>Reads existing attribute of point data set</td>
<td>2-33</td>
</tr>
<tr>
<td></td>
<td>HE5_PTreaddirattr</td>
<td>he5_ptreaddirattr</td>
<td>Reads group attribute from a point</td>
<td>2-34</td>
</tr>
<tr>
<td></td>
<td>HE5_PTreaddlocalattr</td>
<td>he5_ptreaddlocalattr</td>
<td>Reads local attribute from a point</td>
<td>2-35</td>
</tr>
<tr>
<td>Inquiry</td>
<td>HE5_PTwritelnlevels</td>
<td>he5_ptwritelnlevels</td>
<td>Returns the number of levels in a point data set</td>
<td>2-30</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwritenrecs</td>
<td>he5_ptwritenrecs</td>
<td>Returns the number of records in a level</td>
<td>2-31</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwritelnfields</td>
<td>he5_ptwritelnfields</td>
<td>Returns number of fields defined in a level</td>
<td>2-29</td>
</tr>
<tr>
<td></td>
<td>HE5_PTlevelinfo</td>
<td>he5_PTwritelninfo</td>
<td>Returns information about a given level</td>
<td>2-27</td>
</tr>
<tr>
<td></td>
<td>HE5_PTlevelindx</td>
<td>he5_PTwritelnindx</td>
<td>Returns index number for a named level</td>
<td>2-26</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwbcklinkinfo</td>
<td>he5_PTwbcklinkinfo</td>
<td>Returns link field to previous level</td>
<td>2-04</td>
</tr>
<tr>
<td></td>
<td>HE5_PTfwdlinkinfo</td>
<td>he5_PTwfwdlinkinfo</td>
<td>Returns link field to following level</td>
<td>2-15</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwgetlevelname</td>
<td>he5_PTwgetlevelname</td>
<td>Returns level name given level number</td>
<td>2-16</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwgetrecnums</td>
<td>None</td>
<td>Retrieves number of records in one level corresponding to a group of records in a different level</td>
<td>2-17</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwattrinfo</td>
<td>he5_PTwattrinfo</td>
<td>Returns information about point attributes</td>
<td>2-03</td>
</tr>
<tr>
<td></td>
<td>HE5_PTwattrinfo2</td>
<td>he5_PTwattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element.</td>
<td>2-03</td>
</tr>
</tbody>
</table>
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 1-1. Summary of the Point Interface (2 of 2)**

<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name C</th>
<th>Routine Name FORTRAN</th>
<th>Description</th>
<th>Pg. Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inquiry</td>
<td>HE5_PTgrpattrinfo</td>
<td>he5_ptgrpattrinfo</td>
<td>Returns information about point group attributes</td>
<td>2-18</td>
</tr>
<tr>
<td></td>
<td>HE5_PTgrpattrinfo2</td>
<td>he5_ptgrpattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element.</td>
<td>2-18</td>
</tr>
<tr>
<td></td>
<td>HE5_PLocattrinfo</td>
<td>he5_plocattrinfo</td>
<td>Returns information about point local attributes</td>
<td>2-28</td>
</tr>
<tr>
<td></td>
<td>HE5_PLocattrinfo2</td>
<td>he5_plocattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element.</td>
<td>2-28</td>
</tr>
<tr>
<td></td>
<td>HE5_PInqattrs</td>
<td>he5_pinqattrs</td>
<td>Retrieves number and names of point attributes</td>
<td>2-19</td>
</tr>
<tr>
<td></td>
<td>HE5_PInqgrattrs</td>
<td>he5_pinqgrattrs</td>
<td>Retrieves number and names of group attributes</td>
<td>2-22</td>
</tr>
<tr>
<td></td>
<td>HE5_PInqlocattrs</td>
<td>he5_pinqlocattrs</td>
<td>Retrieves number and names of local attributes defined</td>
<td>2-23</td>
</tr>
<tr>
<td></td>
<td>HE5_PInqpoint</td>
<td>he5_pinqpoint</td>
<td>Retrieves number and names of points in file</td>
<td>2-25</td>
</tr>
<tr>
<td></td>
<td>HE5_PInqdatatype</td>
<td>he5_pinqdatatype</td>
<td>Returns data type information about specified level in point</td>
<td>2-20</td>
</tr>
<tr>
<td>Category</td>
<td>Routine Name</td>
<td>Description</td>
<td>Page Nos.</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td><strong>Access</strong></td>
<td>HE5_SWopen</td>
<td>he5_swopenOpens or creates HDF file in order to create, read, or write a swath</td>
<td>2-123</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWcreate</td>
<td>he5_swcreateCreates a swath within the file</td>
<td>2-53</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWattach</td>
<td>he5_swattachAttaches to an existing swath within file</td>
<td>2-47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWdetach</td>
<td>he5_swdetachDetaches from swath interface</td>
<td>2-76</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWclose</td>
<td>he5_swcloseCloses file</td>
<td>2-51</td>
<td></td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>HE5_SWdefdim</td>
<td>he5_swdefdimDefines a new dimension within the swath</td>
<td>2-64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefidxmap</td>
<td>he5_swdefidxmapDefines the mapping between the geolocation and data dimensions</td>
<td>2-66</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefgeofield</td>
<td>he5_swdefgfield Defines a new geolocation field within the swath</td>
<td>2-68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefdatafield</td>
<td>he5_swdefdatafield Defines a new data field within the swath</td>
<td>2-62</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefcomp</td>
<td>he5_swdefcompDefines a field compression scheme</td>
<td>2-59</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefchunk</td>
<td>he5_swdefchunkDefine chunking parameters</td>
<td>2-56</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefcomchunk</td>
<td>he5_swdefcomchunkDefines compression with automatic chunking</td>
<td>2-57</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWsetalias</td>
<td>he5_swsetaliasDefines alias for data field</td>
<td>2-140</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWdropalias</td>
<td>he5_swdropaliasRemoves alias from the list of field aliases</td>
<td>2-78</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWfldrename</td>
<td>he5_swfldrenameChanges the field name</td>
<td>2-87</td>
<td></td>
</tr>
<tr>
<td><strong>Basic I/O</strong></td>
<td>HE5_SWwritefield</td>
<td>he5_swwritefield Writes data to a swath field</td>
<td>2-155</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritegeometa</td>
<td>he5_swwritegeometa Writes field metadata for an existing swath geolocation field</td>
<td>2-160</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritedatameta</td>
<td>he5_swwritedatameta Writes field metadata for an existing swath data field</td>
<td>2-151</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadfield</td>
<td>he5_swreadfield Reads data from a swath field</td>
<td>2-131</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWwriteattr</td>
<td>he5_swwriteattr Writes/update attribute in a swath</td>
<td>2-149</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadattr</td>
<td>he5_swreadattr Reads attribute from a swath</td>
<td>2-126</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritegeograttr</td>
<td>he5_swwritegeograttr WritesUpdates group Geolocation Fields attribute in a swath</td>
<td>2-158</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritelocattr</td>
<td>he5_swwritelocattrWriteUpdates local attribute in a swath</td>
<td>2-163</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadgeograttr</td>
<td>he5_swreadgeograttr Reads attribute in group Geolocation Fields from a swath</td>
<td>2-133</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadgrpattr</td>
<td>he5_swreadgrpattr Reads attribute in Data Fields from a swath</td>
<td>2-134</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWsetfillvalue</td>
<td>he5_swsetfillvalue Sets fill value for the specified field</td>
<td>2-144</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWgetfillvalue</td>
<td>he5_swgetfillvalue Retrieves fill value for the specified field</td>
<td>2-94</td>
<td></td>
</tr>
<tr>
<td><strong>Inquiry</strong></td>
<td>HE5_SWaliasinfo</td>
<td>he5_swaliasinfoRetrieves information about field aliases</td>
<td>2-46</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWgetaliaslist</td>
<td>he5_swgetaliaslistRetrieves list and number of aliases in a geo or data group</td>
<td>2-90</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqdims</td>
<td>he5_swinqdimsRetrieves information about dimensions defined in swath</td>
<td>2-105</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqmaps</td>
<td>he5_swinqmapsRetrieves information about the geolocation relations defined</td>
<td>2-117</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqidxmaps</td>
<td>he5_swinqidxmapsRetrieves information about the indexed geolocation/data mappings defined</td>
<td>2-114</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_SWinggeofields</td>
<td>he5_swinggeofieldsRetrieves information about the geolocation fields defined</td>
<td>2-108</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Routine Name</td>
<td>Description</td>
<td>Page Nos.</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>-------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>Inquiry</td>
<td>HE5_SWinqdatafields</td>
<td>he5_swinqdflds</td>
<td>Retrieves information about the data fields defined</td>
<td>2-100</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqattrs</td>
<td>he5_swinqattrs</td>
<td>Retrieves number and names of attributes defined</td>
<td>2-99</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqdatatype</td>
<td>he5_swidtype</td>
<td>Returns data type information about specified fields in swath</td>
<td>2-101</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqdfldalias</td>
<td>he5_swinqdfldalias</td>
<td>Returns information about data fields &amp; aliases defined in swath</td>
<td>2-103</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqgflldalias</td>
<td>he5_swinqgflldalias</td>
<td>Returns information about geolocation fields &amp; aliases defined in swath</td>
<td>2-111</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqgeograttrs</td>
<td>he5_swinqgeograttrs</td>
<td>Retrieve information about group Geolocation Fields attributes defined in swath</td>
<td>2-109</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqgrpattrs</td>
<td>he5_swinqgrpattrs</td>
<td>Retrieve information about group Data Fields attributes defined in swath</td>
<td>2-113</td>
</tr>
<tr>
<td></td>
<td>HE5_SWlocattrinfo</td>
<td>he5_swlocattrinfo</td>
<td>Retrieve information about local attributes defined in swath</td>
<td>2-115</td>
</tr>
<tr>
<td></td>
<td>HE5_SWlocattrinfo2</td>
<td>he5_swlocattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element</td>
<td>2-119</td>
</tr>
<tr>
<td></td>
<td>HE5_SWhnentries</td>
<td>he5_swnentries</td>
<td>Returns number of entries and descriptive string buffer size for a specified entity</td>
<td>2-122</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdiminfo</td>
<td>he5_swdiminfo</td>
<td>Retrieve size of specified dimension</td>
<td>2-77</td>
</tr>
<tr>
<td></td>
<td>HE5_SWchunkinfo</td>
<td>he5_swchunkinfo</td>
<td>Retrieve chunking information</td>
<td>2-49</td>
</tr>
<tr>
<td></td>
<td>HE5_SWmapinfo</td>
<td>he5_swmapinfo</td>
<td>Retrieve offset and increment of specified geolocation mapping</td>
<td>2-120</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdxmapinfo</td>
<td>he5_swmapinfo</td>
<td>Retrieve offset and increment of specified geolocation mapping</td>
<td>2-96</td>
</tr>
<tr>
<td></td>
<td>HE5_SWatattrinfo</td>
<td>he5_swatattrinfo</td>
<td>Returns information about swath attribute</td>
<td>2-48</td>
</tr>
<tr>
<td></td>
<td>HE5_SWatattrinfo2</td>
<td>he5_swatattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element</td>
<td>2-48</td>
</tr>
<tr>
<td></td>
<td>HE5_SWgeogrpatattrinfo</td>
<td>he5_swgeogrpatattrinfo</td>
<td>Returns information about group Geolocation Fields attribute</td>
<td>2-88</td>
</tr>
<tr>
<td></td>
<td>HE5_SWgeogrpatattrinfo2</td>
<td>he5_swgeogrpatattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element</td>
<td>2-88</td>
</tr>
<tr>
<td></td>
<td>HE5_SWgrpattrinfo</td>
<td>he5_swgrpattrinfo</td>
<td>Returns information about group Data Fields attribute</td>
<td>2-95</td>
</tr>
<tr>
<td></td>
<td>HE5_SWgrpattrinfo2</td>
<td>he5_swgrpattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element</td>
<td>2-95</td>
</tr>
<tr>
<td></td>
<td>HE5_SWfieldinfo</td>
<td>he5_swfieldinfo</td>
<td>Retrieve information about a specific geolocation or data field</td>
<td>2-84</td>
</tr>
<tr>
<td></td>
<td>HE5_SWcompinfo</td>
<td>he5_swcompinfo</td>
<td>Retrieve compression information about a field</td>
<td>2-52</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqswath</td>
<td>he5_swinqswath</td>
<td>Retrieves number and names of swaths in file</td>
<td>2-118</td>
</tr>
<tr>
<td></td>
<td>HE5_SWregionindex</td>
<td>he5_swregidx</td>
<td>Returns information about the swath region ID</td>
<td>2-136</td>
</tr>
<tr>
<td></td>
<td>HE5_SWupdateidxmap</td>
<td>he5_swupdateidxmap</td>
<td>Update map index for a specified region</td>
<td>2-147</td>
</tr>
<tr>
<td></td>
<td>HE5_SWgeomapinfo</td>
<td>he5_swgeomapinfo</td>
<td>Retrieve type of dimension mapping for a dimension</td>
<td>2-89</td>
</tr>
<tr>
<td>Category</td>
<td>Routine Name</td>
<td>C FORTRAN</td>
<td>Description</td>
<td>Page Nos.</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------</td>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Subset</td>
<td>HE5_SWdefboxregion</td>
<td>he5_swdefboxreg</td>
<td>Define region of interest by latitude / longitude</td>
<td>2-54</td>
</tr>
<tr>
<td></td>
<td>HE5_SWregioninfo</td>
<td>he5_swregioninfo</td>
<td>Returns information about defined region</td>
<td>2-138</td>
</tr>
<tr>
<td></td>
<td>HE5_SWextractregion</td>
<td>he5_swextractreg</td>
<td>Read a region of interest from a field</td>
<td>2-84</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdeftimeperiod</td>
<td>he5_swdeftimeper</td>
<td>Define a time period of interest</td>
<td>2-71</td>
</tr>
<tr>
<td></td>
<td>HE5_SWperiodinfo</td>
<td>he5_swperiodinfo</td>
<td>Returns information about a defined time period</td>
<td>2-124</td>
</tr>
<tr>
<td></td>
<td>HE5_SWextracttimeperiod</td>
<td>he5_swextracttimeperiod</td>
<td>Extract a defined time period</td>
<td>2-82</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefvrtregion</td>
<td>he5_swdefvrtreg</td>
<td>Define a region of interest by vertical field</td>
<td>2-73</td>
</tr>
<tr>
<td></td>
<td>HE5_SWindexinfo</td>
<td>he5_swindexinfo</td>
<td>Returns the indices about a subsetted region</td>
<td>2-96</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdupregion</td>
<td>he5_swdupreg</td>
<td>Duplicate a region or time period</td>
<td>2-81</td>
</tr>
<tr>
<td></td>
<td>HE5_PRdefine</td>
<td>he5_prdefine</td>
<td>Defines profile data structure</td>
<td>2-165</td>
</tr>
<tr>
<td></td>
<td>HE5_PRread</td>
<td>he5_prread</td>
<td>Reads profile data</td>
<td>2-171</td>
</tr>
<tr>
<td></td>
<td>HE5_PRwrite</td>
<td>he5_prwrite</td>
<td>Writes profile data</td>
<td>2-175</td>
</tr>
<tr>
<td></td>
<td>HE5_PRinquire</td>
<td>he5_prinquire</td>
<td>Retrieves information about profiles</td>
<td>2-170</td>
</tr>
<tr>
<td></td>
<td>HE5_Prinfor</td>
<td>he5_prinfo</td>
<td>Return information about profile</td>
<td>2-167</td>
</tr>
<tr>
<td></td>
<td>HE5_PRwritegrpattr</td>
<td>he5_prwrgattr</td>
<td>Writes/updates group Profile Fields attribute in a swath</td>
<td>2-177</td>
</tr>
<tr>
<td></td>
<td>HE5_PRreadgrpattr</td>
<td>he5_prrdgattr</td>
<td>Reads attribute in group ProfileFields from a swath</td>
<td>2-173</td>
</tr>
<tr>
<td></td>
<td>HE5_PRinqgrpattrs</td>
<td>he5_prinqgattrs</td>
<td>Retrieve information about group Profile Fields attributes defined in swath</td>
<td>2-169</td>
</tr>
<tr>
<td></td>
<td>HE5_PRgrpattrinfo</td>
<td>he5_prgrpattrinfo</td>
<td>Returns information about a group Profile Fields attribute</td>
<td>2-166</td>
</tr>
<tr>
<td></td>
<td>HE5_PRreclaimspace</td>
<td>Not available</td>
<td>Reclaims memory used by data buffer in HE5_PRread()call</td>
<td>2-174</td>
</tr>
<tr>
<td></td>
<td>HE5_SWmountexternal</td>
<td>Not available</td>
<td>Mount external data file</td>
<td>2-121</td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadexternal</td>
<td>Not available</td>
<td>Read external data set</td>
<td>2-130</td>
</tr>
<tr>
<td></td>
<td>HE5_SWunmount</td>
<td>Not available</td>
<td>Dismount external data file</td>
<td>2-146</td>
</tr>
<tr>
<td></td>
<td>HE5_SWsetextdata</td>
<td>he5_swsetexdat</td>
<td>Set external data set</td>
<td>2-142</td>
</tr>
<tr>
<td></td>
<td>HE5_SWsetdimscale</td>
<td>he5_swsetdimscale</td>
<td>Sets dimension scale for a field dimension within the swath</td>
<td>2-141</td>
</tr>
<tr>
<td></td>
<td>HE5_SWedf dimscale</td>
<td>he5_sWedf dimscale</td>
<td>Sets dimension scale for a dimension for all fields within the swath</td>
<td>2-141</td>
</tr>
<tr>
<td></td>
<td>HE5_SWgetdimscale</td>
<td>he5_sWgetdimscale</td>
<td>Gets dimension scale for a field dimension within the swath</td>
<td>2-91</td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritedscaleattr</td>
<td>he5_sWwritedscaleattr</td>
<td>Writes/Updates a dimension scale attribute in a specific swath</td>
<td>2-152</td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadattscaleattr</td>
<td>he5_sWreadattscaleattr</td>
<td>Reads a dimension scale attribute from a specific dimension</td>
<td>2-127</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqdscaleatts</td>
<td>he5_sWinqdscaleatts</td>
<td>Retrieve information about the attributes defined for a specific dimension scale</td>
<td>2-106</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdscaleattrinfo</td>
<td>he5_sWdscaleattrinfo</td>
<td>Returns information about attribute(s) in a specific dimension scale</td>
<td>2-79</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdscaleattrinfo2</td>
<td>he5_sWdscaleattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element.</td>
<td>2-79</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>HE5_GDopen</td>
<td>he5_gdopen Creates a new file or opens an existing one</td>
<td>2-244</td>
</tr>
<tr>
<td>Access</td>
<td>HE5_GDcreate</td>
<td>he5_gdcreate Creates a new grid in the file</td>
<td>2-187</td>
</tr>
<tr>
<td>Access</td>
<td>HE5_GDattach</td>
<td>he5_gdattach Attaches to a grid</td>
<td>2-181</td>
</tr>
<tr>
<td>Access</td>
<td>HE5_GDdetach</td>
<td>he5_gddetach Detaches from grid interface</td>
<td>2-209</td>
</tr>
<tr>
<td>Access</td>
<td>HE5_GDclose</td>
<td>he5_gdclose Closes file</td>
<td>2-185</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDdeforigin</td>
<td>he5_gddeforigin Defines origin of grid pixel</td>
<td>2-198</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDdefdim</td>
<td>he5_gddefdim Defines dimensions for a grid</td>
<td>2-195</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDdefproj</td>
<td>he5_gddefproj Defines projection of grid</td>
<td>2-200</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDdefpixreg</td>
<td>he5_gddefpixreg Defines pixel registration within grid cell</td>
<td>2-199</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDdeffield</td>
<td>he5_gddeffield Defines data fields to be stored in a grid</td>
<td>2-196</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDdefcomp</td>
<td>he5_gddefcomp Defines a field compression scheme</td>
<td>2-219</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDblkSOMoffset</td>
<td>None This is a special function for SOM MISR data. Write block SOM offset values.</td>
<td>2-183</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDdefcomtile</td>
<td>he5_gddefcomtile Defines compression with automatic tiling</td>
<td>2-194</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDsetalias</td>
<td>he5_gdsetalias Defines alias for data field</td>
<td>2-259</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDdropalias</td>
<td>he5_gddrpalias Removes alias from a list of field alias</td>
<td>2-211</td>
</tr>
<tr>
<td>Category</td>
<td>Routine Name</td>
<td>C Routine Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Basic I/O</td>
<td>HE5_GDwritefieldmeta</td>
<td>he5_gdwrmeta</td>
<td>Writes metadata for field already existing in file</td>
</tr>
<tr>
<td></td>
<td>HE5_GDwritefield</td>
<td>he5_gdwrflid</td>
<td>Writes data to a grid field</td>
</tr>
<tr>
<td></td>
<td>HE5_GDreadfield</td>
<td>he5_gdrdflid</td>
<td>Reads data from a grid field</td>
</tr>
<tr>
<td></td>
<td>HE5_GDwriteattr</td>
<td>he5_gdwattr</td>
<td>Writes/updates attribute in a grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDwriteilocattr</td>
<td>he5_gdwriattr</td>
<td>Writes/updates local attribute in a grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDwritegrpattr</td>
<td>he5_gdwrgattr</td>
<td>Writes/updates group attribute in a grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDreadattr</td>
<td>he5_gdrdaattr</td>
<td>Reads attribute from a grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDreadgrpattr</td>
<td>he5_gdrgattr</td>
<td>Reads group attribute from a grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDreadlocattr</td>
<td>he5_gdrlattr</td>
<td>Reads local attribute from a grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDsetfillvalue</td>
<td>he5_gdsetfill</td>
<td>Sets fill value for the specified field</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgetfillvalue</td>
<td>he5_gdgetfill</td>
<td>Retrieves fill value for the specified field</td>
</tr>
<tr>
<td>Inquiry</td>
<td>HE5_GDgetaliaslist</td>
<td>He5_gdgetaliaslist</td>
<td>Retrieves list and number of aliases in a data group</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqdims</td>
<td>he5_gdinqdims</td>
<td>Retrieves information about dimensions defined in grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqfields</td>
<td>he5_gdinqflds</td>
<td>Retrieves information about the data fields defined in grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqattrs</td>
<td>he5_gdinqattrs</td>
<td>Retrieves number and names of attributes defined</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqdatatype</td>
<td>he5_gdinqdatatype</td>
<td>Returns data type information about specified fields in grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDingrgrpattrs</td>
<td>he5_gdinqgrpattrs</td>
<td>Retrieve information about group attributes defined in grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqlocattrs</td>
<td>he5_gdinqlocattrs</td>
<td>Retrieve information about local attributes defined for a field</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqfldaliases</td>
<td>he5_gdinqfldaliases</td>
<td>Returns information about data fields &amp; aliases defined in grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDnentries</td>
<td>he5_gdnentries</td>
<td>Returns number of entries and descriptive string buffer size for a specified entity</td>
</tr>
<tr>
<td></td>
<td>HE5_GDaliasinfo</td>
<td>he5_gdaliasinfo</td>
<td>Retrieves information about aliases</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgridinfo</td>
<td>he5_gdgridinfo</td>
<td>Returns dimensions of grid and X-Y coordinates of corners</td>
</tr>
<tr>
<td></td>
<td>HE5_GDprojinfo</td>
<td>he5_gdprojinfo</td>
<td>Returns all GCTP projection information</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdiminfo</td>
<td>he5_gddiminfo</td>
<td>Retrieves size of specified dimension</td>
</tr>
<tr>
<td></td>
<td>HE5_GDcompinfo</td>
<td>he5_gdcompinfo</td>
<td>Retrieve compression information about a field</td>
</tr>
<tr>
<td></td>
<td>HE5_GDfieldinfo</td>
<td>he5_gdfieldinfo</td>
<td>Retrieves information about a specific field in the grid</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinggrid</td>
<td>he5_gdinggrid</td>
<td>Retrieves number and names of grids in file</td>
</tr>
<tr>
<td></td>
<td>HE5_GDattrinfo</td>
<td>he5_gdatrinfo</td>
<td>Returns information about grid attributes</td>
</tr>
<tr>
<td></td>
<td>HE5_GDattrinfo2</td>
<td>he5_gdatrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element.</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgrpattrinfo</td>
<td>he5_gdgrpattrinfo</td>
<td>Returns information about a grid group attribute</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgrpattrinfo2</td>
<td>he5_gdgrpattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element.</td>
</tr>
<tr>
<td></td>
<td>HE5_GDlocattrinfo</td>
<td>he5_gdlattrinfo</td>
<td>Returns information about a Data Field’s local attribute(s)</td>
</tr>
<tr>
<td></td>
<td>HE5_GDlocattrinfo2</td>
<td>he5_gdlattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element.</td>
</tr>
<tr>
<td></td>
<td>HE5_GDorigininfo</td>
<td>he5_gdorginfo</td>
<td>Return information about grid pixel origin</td>
</tr>
</tbody>
</table>
Table 1-3. Summary of the Grid Interface (3 of 3)

<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>FORTRAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_GDpixreginfo</td>
<td>he5_gdpixreginfo</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Return pixel registration information for given grid</td>
<td>2-247</td>
</tr>
<tr>
<td>Subset</td>
<td>HE5_GDdefboxregion</td>
<td>he5_gddboxregion</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define region of interest by latitude/longitude</td>
<td>2-190</td>
</tr>
<tr>
<td></td>
<td>HE5_GDregioninfo</td>
<td>he5_gdreginfo</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Returns information about a defined region</td>
<td>2-257</td>
</tr>
<tr>
<td></td>
<td>HE5_GDextractregion</td>
<td>he5_gdxtractregion</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read a region of interest from a field</td>
<td>2-214</td>
</tr>
<tr>
<td>Subset</td>
<td>HE5_GDdeftimeperiod</td>
<td>he5_gdtypeperiod</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define a time period of interest</td>
<td>2-205</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdeftimeperiod</td>
<td>he5_gdtypeperiod</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define a region of interest by vertical field</td>
<td>2-207</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgetpixels</td>
<td>he5_gdgetpix</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>get row/columns for lon/lat pairs</td>
<td>2-222</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgetpixvalues</td>
<td>he5_gdgetpixval</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>get field values for specified pixels</td>
<td>2-224</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinterpolute</td>
<td>he5_gdinterpolute</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Perform bilinear interpolation on a grid field</td>
<td>2-240</td>
</tr>
<tr>
<td>Tiling</td>
<td>HE5_GDdeftile</td>
<td>he5_gddeftile</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define a tiling scheme</td>
<td>2-202</td>
</tr>
<tr>
<td></td>
<td>HE5_GDtileinfo</td>
<td>he5_gdtileinfo</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Retrieve tiling information</td>
<td>2-265</td>
</tr>
<tr>
<td>Utility</td>
<td>HE5_GDij2ll</td>
<td>he5_gdi2ll</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>convert (i,j) coordinates to (lon,lat) for a grid</td>
<td>2-364</td>
</tr>
<tr>
<td></td>
<td>HE5_GDI2lj</td>
<td>he5_gdi2lj</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>convert (lon,lat) coordinates to (i,j) for a grid</td>
<td>2-367</td>
</tr>
<tr>
<td></td>
<td>HE5_GDrs2ll</td>
<td>he5_gdrs2ll</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Convert (r,s) coordinates to (lon,lat) for EASE grid</td>
<td>2-370</td>
</tr>
<tr>
<td>External</td>
<td>HE5_GDsetextdata</td>
<td>he5_gdsetextdata</td>
<td></td>
</tr>
<tr>
<td>Data Sets</td>
<td></td>
<td>Set external data set</td>
<td>2-263</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgetextdata</td>
<td>he5_gdgetextdata</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Get external data set</td>
<td>2-220</td>
</tr>
<tr>
<td>Dimension</td>
<td>HE5_GDsetdimscale</td>
<td>he5_gdsetdimscale</td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td></td>
<td>Sets dimension scale for a field dimension within the grid</td>
<td>2-260</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdefdimscale</td>
<td>he5_gddefdimscale</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sets dimension scale for a dimension of all fields within the grid</td>
<td>2-260</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgetdimscale</td>
<td>he5_gdgetdimscale</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gets dimension scale for a field dimension within the grid</td>
<td>2-218</td>
</tr>
<tr>
<td></td>
<td>HE5_GDwritedscaleattr</td>
<td>he5_gwritedscaleattr</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Writes/Updates a dimension scale attribute in a specific grid</td>
<td>2-269</td>
</tr>
<tr>
<td></td>
<td>HE5_GDreadscaleattr</td>
<td>he5_gscaleattr</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reads a dimension scale attribute from a specific dimension</td>
<td>2-250</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqscalescaleattr</td>
<td>he5_ginqscalescaleattr</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Retrieve information about the attributes defined for a specific dimension</td>
<td>2-232</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdscaleattrinfo</td>
<td>he5_gdscaleattrinfo</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Returns information about attribute(s) in a specific dimension scale</td>
<td>2-212</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdscaleattrinfo2</td>
<td>he5_gdscaleattrinfo2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Same as above, but also returns buffer size of attribute element.</td>
<td>2-212</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HE5_GCTP_GEO</td>
<td>Geographic</td>
</tr>
<tr>
<td>HE5_GCTP_UTM</td>
<td>Universal Transverse Mercator</td>
</tr>
<tr>
<td>HE5_GCTP_SPCS</td>
<td>State Plane Coordinate System</td>
</tr>
<tr>
<td>HE5_GCTP_ALBERS</td>
<td>Albers Conical Equal-Area Projection</td>
</tr>
<tr>
<td>HE5_GCTP_LAMCC</td>
<td>Lambert Conformal Conic</td>
</tr>
<tr>
<td>HE5_GCTP_MERCAT</td>
<td>Mercator Projection</td>
</tr>
<tr>
<td>HE5_GCTP_PS</td>
<td>Polar Stereographic</td>
</tr>
<tr>
<td>HE5_GCTP_POLYC</td>
<td>Polyconic</td>
</tr>
<tr>
<td>HE5_GCTP_TM</td>
<td>Transverse Mercator</td>
</tr>
<tr>
<td>HE5_GCTP_LAMAZ</td>
<td>Lambert Azimuthal Equal Area</td>
</tr>
<tr>
<td>GCTP_SNSOID</td>
<td>Sinusoidal</td>
</tr>
<tr>
<td>HE5_GCTP_HOM</td>
<td>Hotine Oblique Mercator</td>
</tr>
<tr>
<td>HE5_GCTP_SOM</td>
<td>Space Oblique Mercator</td>
</tr>
<tr>
<td>HE5_GCTP_GOOD</td>
<td>Interrupted Goode Homolosine</td>
</tr>
<tr>
<td>HE5_GCTP_ISINUS</td>
<td>Integerized Sinusoidal Projection*</td>
</tr>
<tr>
<td>GCTP_CEA</td>
<td>Cylindrical Equal-Area (for EASE grid with</td>
</tr>
<tr>
<td></td>
<td>Corners in meters)**</td>
</tr>
<tr>
<td>GCTP_BCEA</td>
<td>Cylindrical Equal-Area (for EASE grid with grid</td>
</tr>
<tr>
<td></td>
<td>corners in packed degrees, DMS)**</td>
</tr>
</tbody>
</table>

* 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 31st 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 EOSDIS Evolution and Development Project, December 2017, (333-EED2-001, Revision 01.)

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

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

Starting with HDFEOS5 version 1.15 the Lambert Azimuthal Equal area projection was generalized to support WGS84 ellipsoidal Earth model in addition to the spherical model that was supported before. This generalization was needed to support EASE GRID 2.0 used for SMP products. Starting with version 1.16 we also applied the same for Sinusoidal projection.

<table>
<thead>
<tr>
<th>Code &amp; Projection Id</th>
<th>Array Element</th>
<th>Code &amp; Projection Id</th>
<th>Array Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Geographic</td>
<td>Lon/Z Lat/Z</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>1 U T M</td>
<td>Spheroid Zone</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>2 PGSd_SPCS</td>
<td></td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3 Albers Conical Equal_Area</td>
<td>Smajor Sminor STDPR1 STDPR2 CentMer OriginLat FE FN</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>4 Lambert Conformal C</td>
<td>Smajor Sminor STDPR1 STDPR2 CentMer OriginLat FE FN</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>5 Mercator</td>
<td>Smajor Sminor CentMer TrueScale FE FN</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>6 Polar Stereographic</td>
<td>Smajor Sminor LongPol TrueScale FE FN</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>7 Polyconic</td>
<td>Smajor Sminor CentMer TrueScale FE FN</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>9 Transverse Mercator</td>
<td>Smajor Sminor Factor CentMer OriginLat FE FN</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>11 Lambert Azimuthal**</td>
<td>Smajor Sminor CentLon CenterLat FE FN</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>Lambert Azimuthal</td>
<td>Sphere CentLon CenterLat FE FN</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>16 PGSd_SNSOID**</td>
<td>Smajor Sminor CentMer</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>PGSd_SNSOID</td>
<td>Sphere CentMer</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>20 Hotin Oblique Merc A</td>
<td>Smajor Sminor Factor OriginLat FE FN</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>20 Hotin Oblique Merc B</td>
<td>Smajor Sminor Factor AziAng AzmthPt OriginLat FE FN</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>22 Space Oblique Merc A</td>
<td>Smajor Sminor IncAng AscLong FE FN</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>22 Space Oblique Merc B</td>
<td>Smajor Sminor Satnum Path FE FN</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>24 Interrupted Goode</td>
<td>Sphere</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>97 CEA utilized by EASE grid (see Notes)</td>
<td>Smajor Sminor CentMer TrueScale FE FN</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>98 BCEA utilized by EASE grid (see Notes)</td>
<td>Smajor Sminor CentMer TrueScale FE FN</td>
<td>37</td>
<td></td>
</tr>
</tbody>
</table>

** Lambert Azimuthal and Sinusoidal support both spherical and WGS84 ellipsoidal Earth model
Table 1-4. Projection Transformation Package Projection Parameters (2 of 2)

<table>
<thead>
<tr>
<th>Code &amp; Projection Id</th>
<th>Array Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Geographic</td>
<td>9 10 11 12 13</td>
</tr>
<tr>
<td>1 U T M</td>
<td></td>
</tr>
<tr>
<td>2 PGSd_SPCS</td>
<td></td>
</tr>
<tr>
<td>3 Albers Conical Equal_Area</td>
<td></td>
</tr>
<tr>
<td>4 Lambert Conformal C</td>
<td></td>
</tr>
<tr>
<td>5 Mercator</td>
<td></td>
</tr>
<tr>
<td>6 Polar Stereographic</td>
<td></td>
</tr>
<tr>
<td>7 Polyconic</td>
<td></td>
</tr>
<tr>
<td>9 Transverse Mercator</td>
<td></td>
</tr>
<tr>
<td>11 Lambert Azimuthal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>16 PGSd_SNSOID</td>
<td></td>
</tr>
<tr>
<td>20 Hotin Oblique Merc A</td>
<td>Long1 Lat1 Long2 Lat2 zero</td>
</tr>
<tr>
<td>20 Hotin Oblique Merc B</td>
<td></td>
</tr>
<tr>
<td>22 Space Oblique Merc A</td>
<td>PSRev S Rev</td>
</tr>
<tr>
<td>22 Space Oblique Merc B</td>
<td></td>
</tr>
<tr>
<td>24 Interrupted Goode</td>
<td></td>
</tr>
<tr>
<td>31 &amp; 99 Integerized Sinusoidal</td>
<td>NZone RFlag</td>
</tr>
<tr>
<td>97 CEA utilized by EASE grid (see Notes)</td>
<td></td>
</tr>
<tr>
<td>98 BCEA utilized by EASE grid (see Notes)</td>
<td></td>
</tr>
</tbody>
</table>

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, format 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, format 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, format 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_GDbkomsomoffset.
- 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 (DDDMMMSSSS.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:
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) \( \frac{6371.2280}{(25.067525/2)} = 508.325253 \)
2) \( \frac{6371.2280}{(25.067525/2)} = 508.325253 \)
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 3)

<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>FORTRAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access</td>
<td>HE5_ZAopen</td>
<td>he5_zaoopen  opens or creates HDF file in order to create, read, or write a zonal average</td>
<td>2-336</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAcreate</td>
<td>he5_zacreate creates a zonal average within the file</td>
<td>2-297</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAttach</td>
<td>he5_zaattach attaches to an existing zonal average within the file</td>
<td>2-292</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdetach</td>
<td>he5_zadetach detaches from zonal average interface</td>
<td>2-306</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAClose</td>
<td>he5_zaclose closes file</td>
<td>2-295</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_ZADefDim</td>
<td>he5_zadefdim defines a new dimension within the zonal average</td>
<td>2-303</td>
</tr>
<tr>
<td></td>
<td>HE5_ZADefine</td>
<td>he5_zadefine defines a new data field within the zonal average</td>
<td>2-305</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdefcomp</td>
<td>he5_zadefcomp defines a field compression scheme</td>
<td>2-301</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdefchunk</td>
<td>he5_zadefchunk defines chunking parameters</td>
<td>2-298</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdefchunk</td>
<td>he5_zadefchunk defines compression with automatic chunking</td>
<td>2-299</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAsetalias</td>
<td>he5_zasetalias defines alias for data field</td>
<td>2-346</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdropalias</td>
<td>he5_zadropalias removes alias from the list of field aliases</td>
<td>2-308</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAFldrename</td>
<td>he5_zafldrename changes the field name</td>
<td>2-311</td>
</tr>
<tr>
<td>Basic I/O</td>
<td>HE5_ZAwrite</td>
<td>he5_zawrite writes data to a zonal average field</td>
<td>2-352</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAread</td>
<td>he5_zaread reads data from a zonal average field</td>
<td>2-337</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwriteattr</td>
<td>he5_zawriteattr writes/updates attribute in a zonal average</td>
<td>2-354</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAreadattr</td>
<td>he5_zreadattr reads attribute from a zonal average</td>
<td>2-339</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwritegrpattr</td>
<td>he5_zawritegrattr writes/updates group attribute in a zonal average</td>
<td>2-360</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwritegrpattr</td>
<td>he5_zawritegrattr writes/updates group attribute in a zonal average</td>
<td>2-360</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwritegrpattr</td>
<td>he5_zawritegrattr writes/updates group attribute in a zonal average</td>
<td>2-360</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwritegrpattr</td>
<td>he5_zawritegrattr writes/updates group attribute in a zonal average</td>
<td>2-360</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwritegrpattr</td>
<td>he5_zawritegrattr writes/updates group attribute in a zonal average</td>
<td>2-360</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwritegrpattr</td>
<td>he5_zawritegrattr writes/updates group attribute in a zonal average</td>
<td>2-360</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwritegrpattr</td>
<td>he5_zawritegrattr writes/updates group attribute in a zonal average</td>
<td>2-360</td>
</tr>
<tr>
<td>Category</td>
<td>Routine Name</td>
<td>C</td>
<td>FORTRAN</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------</td>
<td>------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>readlocattr</td>
</tr>
<tr>
<td>Inquiry</td>
<td></td>
<td>HE5_Z</td>
<td>setfillvalue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>getfillvalue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>getaliaslist</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>aliasinfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>inqdims</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>inquire</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>inqattrs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>inqdatatype</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>inqfldalias</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>chunkinfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>inqgrpattrs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>inqlocattrs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>locattrinfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>nentries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>diminfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>attrinfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>attrinfo2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>attrinfo2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>entries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>diminfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>attrinfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>attrinfo2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>grpattrinfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>grpattrinfo2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>info</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>compinfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>mountexternal</td>
</tr>
<tr>
<td>External Files</td>
<td></td>
<td>HE5_Z</td>
<td>readexternal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>unmount</td>
</tr>
<tr>
<td>External Data Sets</td>
<td></td>
<td>HE5_Z</td>
<td>setextdata</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HE5_Z</td>
<td>getextdata</td>
</tr>
<tr>
<td>Category</td>
<td>Routine Name</td>
<td>Routine Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------</td>
<td>--------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>FORTRAN</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAsetdimscale</td>
<td>he5_zasetdimscale</td>
<td>Sets dimension scale for a field dimension within the ZA</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdefdimscale</td>
<td>he5_zadefdimscale</td>
<td>Sets dimension scale for a dimension of all fields within the ZA</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAgetdimscale</td>
<td>he5_zagetdimscale</td>
<td>Gets dimension scale for a field dimension within the ZA</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwritedscaleattr</td>
<td>he5_zawritedscaleattr</td>
<td>Writes/Updates a dimension scale attribute in a specific ZA</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAreaddscaleattr</td>
<td>he5_zareaddscaleattr</td>
<td>Reads a dimension scale attribute from a specific dimension</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAinqdscaleattrs</td>
<td>he5_zainqdscaleattrs</td>
<td>Retrieve information about the attributes defined for a specific dimension scale</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdscaleattrinfo</td>
<td>he5_zadscaleattrinfo</td>
<td>Returns information about attribute(s) in a specific dimension scale</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdscaleattrinfo2</td>
<td>he5_zadscaleattrinfo2</td>
<td>Same as above, but also returns buffer size of attribute element.</td>
</tr>
</tbody>
</table>
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.

Note: The hsize_t typedef uses the largest type of integer available on a machine (typically a 64-bit integer). So when compiling a FORTRAN code in a 64-bit structure one must declare integers as integer*8 (rather than integer *4) for integers whose C equivalent is declared as hsize_t, since underlying C code expects “long” type integer. For 32-bit compilation on a 64-bit machine “integer *4” should work fine.
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)

```fortran
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, HE5_PTattrinfo2**

\[
\text{herr}_t \text{ HE5_PTattrinfo(hid}_t \text{ pointID, const char }^* \text{ attrname, hid}_t^* \text{ numbertype, hsize}_t^* \text{ count)}
\]

\[
\text{herr}_t \text{ HE5_PTattrinfo2(hid}_t \text{ pointID, const char }^* \text{ attrname, hid}_t^* \text{ numbertype, hsize}_t^* \text{ count, hsize}_t^* \text{ size)}
\]

- **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
- **size** OUT: Buffer size of attribute element

**Purpose** Returns information about an object attribute in a specific point object. 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.

\[
\text{status} = \text{HE5_PTattrinfo(pointID, "ScalarFloat", &nt, &count)};
\]

The \( nt \) variable will have the value 10 and \( count \) will have the value 1.

**FORTRAN**

\[
\text{integer function he5_ptattrinfo(pointid,attrname,ntype,count)}
\]

\[
\text{integer} \quad \text{pointid}
\]

\[
\text{character} \quad \text{*attrname}
\]

\[
\text{integer} \quad \text{ntype}
\]

\[
\text{integer}^*4 \quad \text{count}
\]

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

\[
\text{pointid} = \text{he5_ptattrinfo(pointid, "ScalarFloat", ntype, count)}
\]
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)

*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*

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

*FORTRAN*

integer function he5_ptclose(fid)

integer fidd

The equivalent FORTRAN code for the example above is:

```fortran
status = he5_ptclose(fid)
```
Create a New Point Structure

**HE5_PTcreate**

```c
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`.

**FORTRAN**

```fortran
integer function he5_ptcreate(fid, pointname)
   integer pointid, fid
   character *(* pointname

The equivalent FORTRAN code for the example above is:

```
pointid = he5_ptcreate(fid, "ExamplePoint")
```
Define a New Level Within a Point

**HE5_PTdeflevel**

```c
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.

```c
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);
lev1_info.offset[1] = HOFFSET(Lev1_Data, rain);
lev1_info.offset[2] = HOFFSET(Lev1_Data, temp);
lev1_info.offset[3] = HOFFSET(Lev1_Data, id);
lev1_info.dtype[0] = H5T_NATIVE_DOUBLE;
lev1_info.dtype[1] = H5T_NATIVE_FLOAT;
lev1_info.dtype[2] = H5T_NATIVE_FLOAT;
lev1_info.dtype[3] = H5T_NATIVE_CHAR;
lev1_info.dims[0][0] = 1;
lev1_info.dims[1][0] = 1;
lev1_info.dims[2][0] = 1;
lev1_info.dims[3][0] = 1;
lev1_info.datasize = sizeof(Lev1_Data);

status = HE5_PTdeflevel(pointID2, "Observations",
&lev1_info);

for (i = 0; i < 4; i++)
  free(lev1_info.fieldname[i]);

FORTRAN 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.

```c
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:

```fortran
status = he5_ptdeflinkage(pointid, "Desc-Loc", "Observations", "ID")
```
Detach from Point Structure

**HE5_PTdetach**

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

**FORTRAN**

```fortran
integer function he5_ptdetach(pointid)
    integer    pointid, status
    The equivalent FORTRAN code for the example above is:
    ```fortran
    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.)

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

The linkfield will contain the string: ID.

**FORTRAN**

integer function he5_ptfwdlinkinfo(pointid,level,linkfield)

integer pointid,status
character *(*) linkfield
integer level

The equivalent FORTRAN code for the example above is:

```fortran
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 0th 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.

FORTRAN integer function he5_ptgetlevelname(pointid, level, levelname, strbufsz)

```
integer pointid,status,level
character *(* levelname
integer*4 strbufsz
```

The equivalent FORTRAN 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, hsize_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(pointID, inLevel, outLevel, nrec, recs, &outNrec, outRecs);

FORTRAN Not available with this release.
Return Information About Group Attribute

HE5_PTgrpattrinfo, HE5_PTgrpattrinfo2

herr_t HE5_PTgrpattrinfo(hid_t pointID, const char *attrname, hid_t *numbertype, hsize_t *count)

herr_t HE5_PTgrpattrinfo2(hid_t pointID, const char *attrname, hid_t *numbertype, hsize_t *count, hsize_t *size)

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
size OUT: Buffer size of attribute element

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

```c
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 object. 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`:

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

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

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

The variable, `attrlist`, will be set to: "attrOne,attr_2".

### FORTRAN
```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:

```fortran
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.

```c
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 FORTRAN code for the example above is:

```fortran
status = he5_ptinqdatatype(pointid1, "Observations", " ",
HE5_HDFE_DATAGROUP, dtype, classid, order, size)
```
Retrieve Information About Group Attributes

**HE5_PTinqgrpattrs**

```c
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`:

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

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

```c
attrlist = (char *)malloc((strbufsize+1) * sizeof(char));
nattr = HE5_PTinqgrpattrs(pointID, attrlist, strbufsize);
```

The variable, `attrlist`, will be set to: "GrpAttrOne,GrpAttr_2".

**FORTRAN**

```fortran
integer*4 function he5_ptinqgrpattrs(pointid,attrlist,strbufsz)
integer pointid
character *(* attrlist
integer*4 nattr,strbufsz
```

The equivalent FORTRAN code for the example above is:

```fortran
nattr = he5_ptinqgrpattrs(pointid, attrlist, strbufsz)
```
Retrieve Information About Level Attributes

**HE5_PTinqlocattrs**

```c
long HE5_PTinqlocattrs(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`:

```c
nattr = HE5_PTinqlocattrs(pointID, "Observations", NULL, strbufsize);
```

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

```c
attrlist = (char *)malloc((strbufsize+1) * sizeof(char));
nattr = HE5_PTinqlocattrs(pointID, levelname, attrlist, strbufsize);
```

The variable, `attrlist`, will be set to:
"LocAttrOne,LocAttrTwo".

**FORTRAN**

```fortran
integer*4 function he5_ptinqlocattrs(pointid,levelname,attrlist,strbufsz)
integer pointid
```

2-23  EED2-175-002
character *(*)  *levelname*, *attrlist*

integer*4    *nattr,strbufsz*

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

```
nattr = he5_ptinqlocattrs(pointid, *levelname*, *attrlist*,
                          strbufsz)
```
HE5_PTinqpoint

int HE5_PTinqpoint(const char * filename, char *pointlist, long *strbufsize)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>IN: HDF-EOS filename</td>
</tr>
<tr>
<td>pointlist</td>
<td>OUT: Point list (entries separated by commas)</td>
</tr>
<tr>
<td>strbufsize</td>
<td>OUT: String length of point list</td>
</tr>
</tbody>
</table>

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:

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

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

```c
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)

```fortran
integer npoint
character (*) pointlist
integer*4 strbufsz
```

The equivalent FORTRAN code for the example above is:

```fortran
npoint = he5_ptinqpoint("Point.he5", pointlist, strbufsz)
```
Return Index Number of a Named Level

**HE5_PTlevelindx**

```c
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 given 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`.

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

The `levindx` variable will have the value 1.

**FORTRAN**

```fortran
integer function he5_ptlevelindx(pointid,levelname)
    integer       pointid,levindx
    character (*)(*)  levelname
```

The equivalent FORTRAN code for the example above is:

```fortran
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.

```c
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".

**FORTRAN**
See Example 4 from Section 7.1.1.2 of Volume 1 (Overview of Examples).
Return Information About Level Attribute

HE5_PTlocattrinfo, HE5_PTlocattrinfo2

herr_t HE5_PTlocattrinfo(hid_t pointID, const char *levelname, const char *attrname, hid_t *numbertype, hsize_t *count)

herr_t HE5_PTlocattrinfo2(hid_t pointID, const char *levelname, const char *attrname, hid_t *numbertype, hsize_t *count, hsize_t *size)

pointID IN: Point ID returned by HE5_PTcreate or HE5_Ptattach
levelname IN: Level name
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
size OUT: Buffer size of attribute element

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 LocalFloat attribute associated with the level Observations.

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:

```c
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:

```fortran
level = 0
nflds = he5_ptnfields(pointid2, level, fieldlist, strbufsz)
```
Return Number of Levels in a Point Structure

**HE5_PTNlevels**

```c
int HE5_PTNlevels(hid_t pointID)
```

**pointID** 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:

```c
nlevels = HE5_PTNlevels(pointID2);
```

The `nlevels` variable will be 2.

**FORTRAN**

```fortran
integer function he5_ptnlevels(pointid)
integer pointid, nlevels
```

The equivalent FORTRAN code for the example above is:

```fortran
nlevels = he5_ptnlevels(pointid)
```
Return Number of Records in a Given Level

HE5_PTnrecs

hszie_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)
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 object attribute from a specific point object. 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);

FORTRAN integer function he5_ptreadatt(pointid,attrname,buffer)

integer pointid,status
character *(* attrname
<valid type> buffer(*)

The equivalent FORTRAN 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)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointID</td>
<td>Point ID returned by HE5_PTcreate or HE5_PTattach</td>
</tr>
<tr>
<td>attrname</td>
<td>Attribute name</td>
</tr>
<tr>
<td>datbuf</td>
<td>Buffer allocated to hold attribute values</td>
</tr>
</tbody>
</table>

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":

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

FORTRAN

```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)

pointID IN: Point ID returned by HE5_PTcreate or HE5_PTattach
levelname IN: Level name
attrname IN: Attribute name
datbuf 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 In this example, we read floating point attribute with the name "LocalFloat" defined in the Observations level:

```c
status = HE5_PTreadlocattr(pointID, "Observations", "LocalFloat", &attr_val);
```

FORTRAN integer function he5_ptreadlocattr(pointid,levelname,attrname,buffer)

```fortran
integer pointid,status
character (*) levelname,attrname
<valid type> buffer(*)
```

The equivalent FORTRAN code for the example above is:

```fortran
status = he5_ptreadlocattr(pointid, "Observations", "LocalFloat", buffer)
```
Read Records From a Point Level

**HE5_PTreadlevel**

```c
herr_t HE5_PTreadlevel(hid_t pointID, int level, HE5_CmpDTSinfo *inStruct, size_t *size, void *buffer)
```

- **pointID** IN: Point ID returned by HE5_PTcreate or HE5_PTattach
- **level** IN: Level to read (0-based)
- **inStruct** IN: C-data structure containing information about specified level.
- **size** IN: Size (in bytes) of data structure to read data to.
- **buffer** 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, *pointID1*. 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”.

```c
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, hsize_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 refered to by the point ID, pointID1.

```c
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.

FORTRAN 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 object attribute in a specific point object. 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:

```c
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:

```c
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_ATIVE_FLOAT, count, buffer)
# Write/Update Point Group Attribute

**HE5_PTwritegrpattr**

```c
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:

```c
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:

```c
attr_val = 3.14159;
status = HE5_PTwritegrpattr(pointid, "GroupFloat", H5T_NATIVE_FLOAT, 1, &attr_val);
```

**FORTRAN**

```fortran
integer function he5_ptwritegrpattr(pointid, attrname, ntype, count, buffer)

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

---

2-40 | EED2-175-002
<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_ptwritegrpatr(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);

FORTRAN See Example 3 from Section 7.1.1.2 of Volume 1 (Overview of Examples).
Write/Update Point Level Attribute

HE5_PTwritelocattr

herr_t HE5_PTwritelocattr(hid_t pointID, const char *levelname, const char *attrname, int ntype, hsize_t count, void *datbuf)

pointID       IN: Point ID returned by HE5_PTCreat 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_PTwritelocattr(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_PTwritelocattr(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.

Note: The hsize_t typedef uses the largest type of integer available on a machine (typically a 64-bit integer). So when compiling a FORTRAN code in a 64-bit structure one must declare integers as integer*8 (rather than integer *4) for integers whose C equivalent is declared as hsize_t, since underlying C code expects “long” type integer. For 32-bit compilation on a 64-bit machine “integer *4” should work fine.
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)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fid</td>
<td>Swath file ID returned by HE5_SWopen</td>
</tr>
<tr>
<td>swathname</td>
<td>Name of swath to be attached</td>
</tr>
</tbody>
</table>

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:

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

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

FORTRAN

integer function he5_swattach(fid,swathname)

integer         fid
character(*) swathname

The equivalent FORTRAN code for the example above is:

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

Note: If unlike the above example user defines a swathname string and then copies the actual name into that string, then it is suggested that user initialize every single character in the swathname string in their code to "'0'", before copying swathname into this string [before passing the string into HE5_SWattach() ]. If user is getting the swath name from another call, then user must initialize the swathname string before that call. Failing to do this may result in having some random characters in the swathname and, therefore, failing of HE5_SWattach().
Return Information About a Swath Attribute

HE5_SWattrinfo, HE5_SWattrinfo2

herr_t HE5_SWattrinfo(hid_t swathID, const char *attrname, hid_t *ntype, hsize_t *count)

herr_t HE5_SWattrinfo2(hid_t swathID, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

- **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
- **size**: OUT: Buffer size of attribute element

**Purpose**: Returns information about an object attribute in a specific swath object. 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.

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

The variable will have the value 10 and `count` will have the value 1.

**FORTRAN**: integer function he5_swattrinfo(swathid, attrname, ntype, count)

```fortran
tinteger swathid
ntcharacter(*) attrname
integer ntype
ntinteger count
```

The equivalent FORTRAN code for the example above is:

```fortran
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}

FORTRAN integer function he5_swchunkinfo(swathid, fieldname,chunk_rank, chunk_dims)

integer swathid
character*(*) fieldname
integer chunk_rank
integer*4 chunk_dims(*)

The equivalent FORTRAN 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, chunkDims[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)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>fid</em></td>
<td>IN: Swath file ID returned by HE5_SWopen</td>
</tr>
</tbody>
</table>

**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)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>fid</em></td>
<td></td>
</tr>
</tbody>
</table>

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:

```c
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)

```fortran
integer swathid
character(*) filename
integer compcode
integer compparm(*)
```

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

```fortran
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*.

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

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

**FORTRAN**

```fortran
integer function he5_swcreate(fid, swathname)

integer    fid
character(*) swathname

The equivalent FORTRAN code for the example above is:

```fortran
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);
FORTRAN

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

integer swathid
real*8 cornerlon(*)
real*8 cornerlat(*)
integer mode

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_swdefboxreg(swathid, cornerlon, cornerlat, HE5_HDFE_MIDPOINT)
Define Chunking Parameters

**HE5_SWdefchunk**

```c
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.
```c
chunk_dims[0] = 100;
chunk_dims[1] = 360;
status = HE5_SWdefchunk(swathID, 2, chunk_dims);
```

**FORTRAN**

```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:
```fortran
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
comcode  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**

```c
herr_t HE5_SWdefcomp(hid_t swathID, int compcode, int *compparm)
```

**Parameters**

- `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.

```fortran
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)

- **swathID** IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
- **fieldname** 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
- **ntype** IN: The number type of the data stored in the field
- **merge** 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 C order, that is, the last dimension is incremented first. The API will attempt to merge into a single object those fields that share dimensions and in case of multidimensional fields, numbertype.

**Note:** One should define both chunking and compression before every 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:

```c
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**

```c
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:

```c
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:

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

**FORTRAN**

```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:

```fortran
parameter (HE5S_UNLIMITED_F=-1)
status = he5_swdefdim(swathid, "Unlim", HE5S_UNLIMITED_F)
```
Define Mapping between Geolocation and Data Dimensions

**HE5_SWdefdimmap**

```c
herr_t HE5_SWdefdimmap(hid_t swathID, char *geodim, char *datadim, hsize_t offset, hsize_t increment)
```

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

```c
status = HE5_SWdefdimmap(swathID, "GeoTrack", "DataTrack", 0, 2);
status = HE5_SWdefdimmap(swathID, "GeoXtrack", "DataXtrack", 1, 2);
```

**FORTRAN**

```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:

```fortran
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)

swathID IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
fieldname IN: Name of field to be defined
dimlist IN: The list of geolocation dimensions defining the field
maxdimlist IN: The maximum dimension list of geolocation dimensions defining the field
ntype IN: The number type of the data stored in the field
merge 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 C order, that is, the last dimension is incremented first. The API will attempt to merge into a single object those fields that share dimensions and in case of multidimensional fields, ntype. Two and three dimensional fields will be merged into a single three-dimensional object if the last two dimensions (in C order are equal). If the merge code for a field is set to 0, the API will not attempt to merge it with other fields. Fields using the unlimited dimension will not be merged.

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

status = 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**

```c
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.

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

```c
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.

**FORTRAN**

```fortran
integer function he5_swdefimap(swathid, geodim, datadim, index)
```

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

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

```fortran
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.

```c
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**

```c
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);
FORTRAN integer function he5_swdefvrtreg(swathid, regionid, vertobj, range)
  integer swathid
  integer regionid
  character(*) vertobj
  real*8 range(*)

The equivalent FORTRAN 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**

\[
\text{herr}_t \text{HE5_SWdetach}(\text{hid}_t \text{swathID})
\]

**swathID**  \hspace{1cm} \text{IN:}  \hspace{1cm} \text{Swath ID returned by HE5_SWcreate or HE5_SWattach}

**Purpose**  \hspace{1cm} \text{Detaches from swath interface.}

**Return value**  \hspace{1cm} \text{Returns SUCCEED (0) if successful or FAIL (-1) otherwise.}

**Description**  \hspace{1cm} \text{This routine should be run before exiting from the swath file for every swath opened by HE5_SWcreate or HE5_SWattach.}

**Example**  \hspace{1cm} \text{In this example, we detach the swath structure, ExampleSwath:}

\[
\text{status} = \text{HE5_SWdetach} (\text{swathID});
\]

**FORTRAN**

\[
\text{integer function he5_swdetach(swathid)}
\]

\[
\text{integer} \hspace{1cm} \text{swathid}
\]

\[
\text{The equivalent FORTRAN code for the example above is:}
\]

\[
\text{status} = \text{he5_swdetach} (\text{swathid})
\]
HE5_SWdiminfo

hsiz_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":

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

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

**FORTRAN**

```fortran
    integer*4 function he5_swdiminfo(swathid, dimname)
    integer swathid
    character(*) dimname
    integer*4 dimsize
    The equivalent FORTRAN code for the example above is:
    dimsize = he5_swdiminfo(swathid, "GeoTrack")
```
Remove an Alias for Swath Data Field

HE5_SWdropalias

```c
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*.

```c
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:

```fortran
aliasname = "temps 0 to 30"
status = he5_swdropalias(swathid, HE5_HDFE_DATAGROUP, aliasname)
```
Return Information about a Swath Dimension Scale Attribute

HE5_SWdscaleattrinfo, HE5_SWdscaleattrinfo2

herr_t HE5_SWdscaleattrinfo(hid_t swathID, const char *dimname, const char *attrname, hid_t *ntype, hsize_t *count)

herr_t HE5_SWdscaleattrinfo2(hid_t swathID, const char *dimname, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

swathID  IN:  Swath ID returned by HE5_SWcreate or HE5_SWattach
dimname  IN:  Dimension scale name
attrname  IN:  Attribute name
ntype  OUT:  Number type of attribute
count  OUT:  Number of attribute elements
size  OUT:  Buffer size of attribute element

Purpose  Returns information about attribute(s) in a specific dimension scale.
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 dimension scale attribute.
Example  In this example, we return information about the IntValues attribute of Bands dimension scale.

status = HE5_SWdscaleattrinfo(swathID, "Bands", "IntValues", &ntype, &count);

The ntype variable will have the value 0 and count will have the value of 3.

FORTRAN  integer function he5_swdscaleattrinfo(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_swdscaleattrinfo(swathid, "Bands", "IntValues", ntype, count)
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:

```c
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**

```c
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_SWattatch
- **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.

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

**FORTRAN**: integer function he5_swextper(swathID, periodid, fieldname, externalflag, buffer)

```fortran
integer swathID
integer periodid
character(*) fieldname
integer externalflag
<valid type> buffer(*)
```
The equivalent FORTRAN code for the example above is:

```fortran
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)

- **swathID** IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
- **fieldname** IN: Fieldname
- **rank** OUT: Rank of field
- **dims** OUT: Array containing the dimension sizes of the field
- **ntype** OUT: Array containing the numbertype of the field. See Appendix A for interpretation of number types.
- **dimlist** OUT: List of dimensions in field
- **maxdimlist** 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, HE5_SWgeogrpatrinfo2**

herr_t HE5_SWgeogrpatrinfo(hid_t swathID, const char *attrname,
        hid_t *ntype, hsize_t *count)

herr_t HE5_SWgeogrpatrinfo2(hid_t swathID, const char *attrname,
        hid_t *ntype, hsize_t *count, hsize_t *size)

- **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
- **size** OUT: Buffer size of attribute element

**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.

```c
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:

```fortran
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.

```fortran
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.

**FORTRAN**

```fortran
integer function he5_swgmapinfo(swathid, geodim)

integer swathid
character(*) geodim

The equivalent FORTRAN code for the example above is:

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

HE5_SWgetaliaslist

long HE5_SWgetaliaslist(hid_t swathID, int fldgroup, char *aliaslist, long *strbufsize)

  IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
  IN: Field group flag (geo or data)
  OUT: List of alias(es) in the “Data Fields” or “Geo Fields” group (comma separated list)
  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 Dimension Scale for a Dimension of a Field within a Swath

HE5_SWgetdimscale

long HE5_SWgetdimscale(hid_t swathID, char *fieldname, char *dimname, hsize_t *dimsize, hid_t *numbertype, void *data)

swathID IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
fieldname IN: Name of the field whose dimname dimension scale is read
dimname IN: The dimension for which scale values are read
dimsize OUT: The size of the dimension to be read
numbertype OUT: The number type of the data stored in the scale. See Appendix A for number types.
data OUT: Values to be read for the dimension scale

Purpose Gets dimension scale for a field dimension within the swath.

Return value Returns data buffer size if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list or nonexistent field.

Description This routine gets dimension scale for a field dimension within the swath. The dimension scales attributes label, unit, format and others can be read using HE5_SWreaddscaleattr().

Example In this example, we get dimension scale for the Bands dimension in the Spectra field, defined using HE5_SWsetdimscale() or HE5_SWdefdimscale():

long buffsize;

hsize_t nbands
hid_t ntype;
int *bands;

/* First call, with NULL for data buffer, returns */
/* buffsize needed for the data buffer */
buffsize = HE5_SWgetdimscale(swathID, "Spectra",
    "Bands", &nbands, &ntype, NULL);

/* allocate enough buffer for the data */
bands = (int *)malloc(buffsize);

buffsize = HE5_SWgetdimscale(swathID, "Spectra",
    "Bands", &nbands, &ntype, (void *)bands);

FORTRAN  integer function he5_swgetdimscale(swathid, fieldname, dimname, dimsize, numbertype, data)
              integer*4     swathid  
              character*(*)  fieldname 
              character*(*)  dimname 
              integer*4     dimsize  
              integer*4     numbertype 
              <valid type>    data(*) 

The equivalent FORTRAN code for the example above is:

integer*4  bands(5)
integer*4  nbands, ntype, buffsize

buffsize = he5_swgetdimscale(swathid, "Spectra",
    "Bands", nbands, ntype, buffsize);
Get External Data File Information

**HE5_SWgetextdata**

```c
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:

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

**FORTRAN**

```fortran
integer function he5_swgetxdat(swathid,fieldname,nlen,flist,offset,size)
  integer swathid
  integer nfiles
  integer*4 nlen
  integer*4 offset(*)
  integer*4 size(*)
  character(*) filename
  character(*) flist
```

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

```fortran
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:

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

**FORTRAN**
integer function he5_swgetfill(swathid,fieldname,fillval)

```fortran
integer swathid
character(*) fieldname
<valid type> fillval(*)
```

The equivalent FORTRAN code for the example above is:

```fortran
status = he5_swgetfill(swathid, "Temperature", tempfill)
```
Return Information about a Swath Attribute in Group “Data Fields”

HE5_SWgrpatrinfo, HE5_SWgrpatrinfo2

herr_t HE5_SWgrpatrinfo(hid_t swathID, const char *attrname, hid_t *ntype, hsize_t *count)

herr_t HE5_SWgrpatrinfo2(hid_t swathID, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

*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*

*size OUT: Buffer size of attribute element*

**Purpose** Returns information about a group attribute in “Data Fields” group. See Section 3.6 of Volume I (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.

```c
status = HE5_SWgrpatrinfo(swathID, "ScalarFloat", &nt, &count);
```

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

**FORTRAN**

```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:

```fortran
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:

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

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

**FORTRAN**

```fortran
integer*4 function he5_swimapinfo(swathid, geodim, datadim, index)
integer swathid
character(*) geodim
character(*) datadim
integer*4 index(*)

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

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

```fortran
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:

```c
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`

**FORTRAN**

```fortran
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:

\[
\text{status} = \text{he5_swindexinfo(regionid, "Longitude", rank, dimlist, indices)}
\]

The return parameters will have the following values:

\[
\text{rank}=2, \ dimlist="DataXtrack,DataTrack", \text{ and } indices(1,1)=0, \indices(1,2)=10, \indices(2,1)=4, \indices(2,2)=11
\]

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 object. 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:

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

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

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

```c
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:

```fortran
nattr = he5_swinqattrs(swathid, attrnames, strbufsize)
```
Retrieve Information about Data Fields Defined in Swath

**HE5_SWinqdatafields**

```c
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 number type 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:

```c
nflds = HE5_SWinqdatafields(swathID, fieldlist, rank, ntype);
```

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


**FORTRAN**

```fortran
integer*4 function he5_swinqdflds(swathid, fieldlist, rank, ntype)

integer swathid
character(*) fieldlist
integer rank(*)
integer ntype(*)
```

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

```fortran
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)
```

- **swathID** IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
- **fieldname** IN: Field name
- **attrname** IN: Attribute name
- **fieldgroup** 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
- **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 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 *Spectra* field in the swath defined in the *HE5_SWdefdatafield* routine.
```
status = HE5_SWinqdatatype(swathID, "Spectra", NULL, fieldgroup, &datatype, &classid, &order, &size);
```
FORTRAN

integer function he5_swidtype(swathid,fieldname,attrname,fgdgrp,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**

```c
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:

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

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

```c
fldalias = (char *)calloc(strbufsize+1, sizeof(char));
nfldalias = HE5_SWinqdfldalias(swathID, fldalias, &strbufsize);
```

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

"Temperature,Temp".

**FORTRAN**

```fortran
integer*4 function he5_swinqdfldalias(swathid, fldalias, strbufsize)
integer swathid
character(*) fldalias
integer*4 strbufsize
integer*4 nfldalias
```

2-103 EED2-175-002
The equivalent *FORTRAN* code for the example above is:

\[
nfldalias = \text{he5}\_\text{swinqdfldalias}(\text{swathid, fldalias, strbufsize})
\]
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 for Swath Dimension Scale Attributes

HE5_SWinqdscaleattrs

long HE5_SWinqdscaleattrs(hid_t swathID, const char *dimname, char *attrnames, long *strbufsize)

swathID    IN:    Swath ID returned by HE5_SWcreate or HE5_SWattach

dimname    IN:    Dimension scale name to retrieve attribute information

attrnames  OUT:    Attribute list (entries separated by commas)

strbufsize OUT:    String length of attribute list

Purpose Retrieve information about the attributes defined for a specific dimension scale.

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 dimension scale attributes defined for a field “Bands”. In the first call, set the parameter attrnames to NULL. We assume that there are five attributes stored, label, unit, format, MissingValue, and IntValues:

nattr = HE5_SWinqlocattrs(swathID, “Bands”, NULL, &strbufsize);

The parameter, nattr, will have the value 5 and strbufsize will have value 40.

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

nattr = HE5_SWinqlocattrs(swathID, “Bands”, attrnames, &strbufsize);

The variable, attrlist, will be set to:

"label,unit,format,MissingValue,IntValues ".

FORTRAN integer*4 function he5_swinqdscaleattrs(swathid, dimname, attrnames, strbufsize)

integer swathid
character(*) \textit{dimname}

class character(*) \textit{attrnames}

integer*4 \textit{strbufsize}

The equivalent \textit{FORTRAN} code for the example above is:

\begin{verbatim}
nattr = he5_swinqlattrs(swathid, "Bands", attrnames, strbufsize)
\end{verbatim}
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:

```c
nflds = HE5_SWinqgeofields(swathID, fieldlist, rank, ntype);
```

The parameter, **fieldlist**, will have the value: "Longitude,Latitude" with


**FORTRAN** integer*4 function he5_swinqgflds(swathid, fieldlist, rank, ntype)

```fortran
integer swathid
character(*) fieldlist
integer rank(*)
integer ntype(*)
```

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

```fortran
nflds = he5_swinqgflds(swathid, fieldlist, rank, ntype)
```
Retrieve Information about Group “Geolocation Fields” Attributes

HE5_SWinqgeograttrs
long HE5_SWinqgeograttrs(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_SWinqgeograttrs(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_SWinqgeograttrs(swathID, attrnames, &strbufsize);

The variable, attrnames, will be set to:
"attrOne,attr_2".

FORTRAN integer*4 function he5_swinqgeogattrs(swathid,attrnames, strbufsize)
integer swathid
character(*) attrnames
integer*4 strbufsize
integer*4 nattr
The equivalent *FORTRAN* code for the example above is:

\[ \text{nattr} = \text{he5_swinqgeogattrs}(\text{swathid}, \text{attrnames}, \text{strbufsize}) \]
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_SWinqgfldalialias(swathID, fldalias, &strbufsize);

The variable, fldalias, will be set to:

"Latitude,Lat".

FORTRAN integer*4 function he5_swinqgfldalialias(swathid, fldalias, strbufsize)

integer swathid
character*($) fldalias
integer*4 strbufsize
integer*4 nfldalias
The equivalent *FORTRAN* code for the example above is:

\[ \text{nfldalias} = \text{he5/swing/fldalias(swathid, fldalias, strbufsize)} \]
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`:

```c
nattr = HE5_SWinqgrpattrs(swathID, NULL, &strbufsize);
```

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

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

```c
nattr = HE5_SWinqgrpattrs(swathID, attrnames, &strbufsize);
```

The variable, `attrnames`, will be set to: "attrOne,attr_2".

**FORTRAN**

integer*4 function he5_swinqgattrs(swathid, attrnames, strbufsize)

```fortran
integer     swathid
character(*) attrnames
integer*4   strbufsize
integer*4   nattr
```

The equivalent FORTRAN code for the example above is:

```fortran
nattr = he5_swinqgattrs(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}.

FORTRAN integer*4 function he5_swinqimaps(swathid, dimmap, idxsizes)

integer swathid
character(*) dimmap
integer*4 idxsizes(*)

The equivalent FORTRAN code for the example above is:

nidxmaps = he5_swinqimaps(swathid, dimmap, idxsizes)
Retrieve Information Swath Local Attributes

**HE5_SWinqlattrs**

```c
long HE5_SWinqlattrs(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`:

```c
nattr = HE5_SWinqlattrs(swathID, "DataField", NULL, &strbufsize);
```

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

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

```c
nattr = HE5_SWinqlattrs(swathID, "DataField", attrnames, &strbufsize);
```

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

"attrOne,attr_2".

**FORTRAN**

```fortran```
integer*4 function he5_swinqlattrs(swathid , fieldname, attrnames, strbufsize)
integer swathid
character(*) fieldname
character(*) attrnames
```

---

2-115 EED2-175-002
integer*4  \textit{strbufsize}

The equivalent \textit{FORTRAN} code for the example above is:

\begin{verbatim}
nattr = he5_swinqlattrs(swathid, "DataField", attrnames, strbufsize)
\end{verbatim}
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:

```c
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}.

**FORTRAN**  
integer*4 function

```fortran
he5_swinqmaps(swathid,dimmap,offset,increment)
integer swathid
character(*) dimmap
integer(*) offset(*)
integer(*) offset(*)
```

The equivalent FORTRAN code for the example above is:

```fortran
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”.

FORTRAN integer*4 function he5_swinqswath(filename,swathlist,strbufsize)
character(*) filename
colorcharacter(*) swathlist
integer*4 strbufsize

The equivalent FORTRAN code for the example above is:

nswath = he5_swinqswath('Swath.he5', swathlist, strbufsize)
Return Information about a Local Swath Attribute

**HE5_SWlocattrinfo, HE5_SWlocattrinfo2**

herr_t HE5_SWlocattrinfo(hid_t swathID, const char *fieldname, const char *attrname, hid_t *ntype, hsize_t *count)

herr_t HE5_SWlocattrinfo2(hid_t swathID, const char *fieldname, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

- **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
- **size** OUT: Buffer size of attribute element

**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.

```c
status = HE5_SWlocattrinfo(swathID, "DataField", attrname, &ntype, &count);
```

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

**FORTRAN**

```fortran
integer function he5_swattrinfo(swathid, filename, attrname, ntype, count)
  integer swathid
  character(*) attrname
  integer ntype
  integer *4 count
```

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

```fortran
status = he5_swattrinfo(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:

```c
status = HE5_SWmapinfo(swathID, "GeoTrack", "DataTrack", 
                        &offset, &increment);
```

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

FORTRAN
integer function he5_swmapinfo(swathid, geodim, datadim, offset, increment)

```fortran
integer swathid
character(*) geodim
character(*) datadim
integer*4 offset
integer*4 increment
```

The equivalent FORTRAN code for the example above is:

```fortran
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 thoughtout 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 \( \text{bufsz} = 39 \)

**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.

```c
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:

```c
parameter (HE5F_ACC_RDWR = 100)
parameter (HE5F_ACC_RDONLY = 101)
parameter (HE5F_ACC_TRUNC = 102)
```

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

```c
fid = he5_swopen("Swath.he5", HE5F_ACC_TRUNC)
```

**Note to users of the SDP Toolkit:** Please refer to the SDP Toolkit User Guide for the EOSDIS Evolution and Development Project (333-EED2-001, Revision 01), 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**

```c
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.

```c
/* 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 swathid
  integer periodid
  character(*) fieldname
  integer ntype
  integer rank
  integer*4 dims(*)
  integer*4 size

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 object. 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":

```c
status = HE5_SWreadattr(swathID, "ScalarFloat", &data);
```

**FORTRAN**: integer function he5_swrdattr(swathid, attrname, datbuf)

```fortran
integer swathid
<valid type> datbuf(*)
```

The equivalent FORTRAN code for the example above is:

```fortran
status = he5_swrdattr(swathid, "ScalarFloat", datbuf)
```
Read Attribute for a Dimension scale within a Swath

**HE5_SWreaddscaleattr**

```c
herr_t HE5_SWreaddscaleattr(hid_t swathID, const char *dimname, const char *attrname, void *datbuf)
```

- **swathID** IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
- **dimname** IN: Dimension scale name for which attribute is written
- **attrname** IN: Attribute name
- **datbuf** OUT: Buffer allocated to hold attribute values

**Purpose** Reads a dimension scale attribute from a specific dimension.

**Return value** Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id 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 attributes of the Bands dimension scale:

```c
herr_t status = FAIL;
hid_t SWid1 = FAIL;
int i;
long nattr;
long strbufsize;
char *attrlist;
size_t fldnmlen[HE5_HDFE_NAMBUFSIZE];
char *fldnm[HE5_HDFE_NAMBUFSIZE];
char *attrname = (char *)NULL;
hid_t *ntype;
hsize_t count = 0;
void *attr;
int *attr_int;
float *attr_flt;
double *attr_dbl;
char *attr_char;
nattr = HE5_SWinqdscaleattrs(SWid1, "Bands", NULL, &strbufsize);
```
attrlist = (char *) calloc(strbufsize + 2, sizeof(char));
nattr = HE5_SWinqdscaleattrs(SWid1, "Bands",
    attrlist, &strbufsize);
nattr = HE5_EHparsestr(attrlist, ',', fldnm, fldnmlen);
for( i = 0; i < nattr; i++)
{
    attrname = (char *)calloc(fldnmlen[i] + 1, sizeof(char));
    memmove(attrname,fldnm[i],fldnmlen[i]);
    ntype = (hid_t *)calloc(1, sizeof(hid_t));
    if(strcmp(attrname, "REFERENCE_LIST") == 0 )
    {
        continue;
    }
    status = HE5_SWdscaleattrinfo(SWid1,"Bands",
        attrname, ntype, &count);
    if( (int)*ntype == 0 ) {
        attr_int = (int *)malloc(count * sizeof(int));
        attr = (void *) attr_int;
    } else if( (int)*ntype == 10 ) {
        attr_flt = (float *)malloc(count * sizeof(float));
        attr = (void *) attr_flt;
    } else if( (int)*ntype == 11 ) {
        attr_dbl = (double *)malloc(count * sizeof(double));
        attr = (void *) attr_dbl;
    } else if( (int)*ntype == 57 ) {
        attr_char = (char *)malloc((count+1) * sizeof(char));
        attr = (void *) attr_char;
    }
    status = HE5_SWreaddscaleattr(SWid1,"Bands",attrname, attr);
}

FORTAN

integer function he5_swreaddscaleattr (swathid, dimname, attrname, datbuf)
integer*4 swathid
character(*) dimname
character(*) attrname
<valid type> datbuf(*)

The equivalent FORTAN code for the example above is:

integer j, ntype
integer swid1
integer attr_int(25)
real*4 attr_flt(25)
real*8 attr_dbl(25)
character attr_char(25)
integer nattr
character*100 attrlist
character*100 strbufsize
character*15 attrname(10)
nattr = HE5_SWingdscaleattrs(SWid1, "Bands", attrlist, strbufsize)
attrname(1) = 'label'
attrname(2) = 'unit'
attrname(3) = 'format'
attrname(4) = 'MissingValue'
attrname(5) = 'IntValues'
do j = 1,5
  attr_char = ''
count(1)= 0
count(2)= 0
status = HE5_SWdscaleattrinfo(SWid1,"Bands",
attrname(j), ntype, count)
if( ntype .eq. 0) then
  status = HE5_SWreaddscaletattr(SWid1,"Bands",
  attrname(j), attr_int)
endif
if( ntype .eq. 10) then
  status = HE5_SWreaddscaletattr(SWid1,"Bands",
  attrname(j), attr_flt)
endif
if( ntype .eq. 11) then
  status = HE5_SWreaddscaletattr(SWid1,"Bands",
  attrname(j), attr_dbl)
endif
if( ntype .eq. 57) then
  status = HE5_SWreaddscaletattr(SWid1,"Bands",
  attrname(j), attr_char)
endif
enddo
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);  
```

FORTRAN  Not available with this release.
Read Data from a Swath Field

**HE5_SWreadfield**

```c
herr_t HE5_SWreadfield(hid_t swathID, char *fieldname, const hssize_t start[],
const hsize_t stride[], const hsize_t edge[], void *buffer)
```

- **swathID** IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
- **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 read along each dimension
- **buffer** 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 `start`, `stride`, and `edge` arrays refer to the swath 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 

\[
\left(\text{dim} - \text{start}\right) / \text{stride} \quad \text{where dim refers is the size of the dimension.}
\]

Note that to allocate a string buffer size for reading an array of strings, first using `HE5_SWreadlocattr` to get the value of maximum string length in the local attribute `StringLengthAttribute`.

**Example** In this example, we read data from the 10th track (0-based) of the `Longitude` field.

```c
float     track[1000];
hssize_t  start[2] = {9,1};
hsize_t   edge[2]  = {1,1000};
status = HE5_SWreadfield(swathID, "Longitude", start, NULL,
edge, track);
```
The start, stride, and edge arrays must be defined explicitly, with the start array being 0-based.

**Note:** `he5_swrdcharfld()` 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:

```fortran
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_SWreadgeogrpattr**

```c
herr_t HE5_SWreadgeogrpattr(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":

```c
classic
status = HE5_SWreadgeogrpattr(swathID, "ScalarFloat", &data);
```

**FORTRAN**

```fortran
integer function he5_swrdgeogattr(swathid,attrname,datbuf)
integer swathid
character(*) attrname
<valid type> datbuf(*)
```

The equivalent FORTRAN code for the example above is:

```fortran
status = he5_swrdgeogattr(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".

```c
status = HE5_SWreadgrpattr(swathID, "ScalarFloat", &data);
```

**FORTRAN** integer function he5_swrdgattr(swathid, attrname, datbuf)

- **swathid**: integer
- **attrname**: character(*)
- **datbuf**: <valid type>*

The equivalent FORTRAN code for the example above is:

```fortran
status = he5_swrdgattr(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":

```c
status = HE5_SWreadlocattr(swathID, "DataField", "ScalarFloat", &data);
```

**FORTRAN**

```fortran
integer function he5_swrdlattr(swathid, pathname, attrname, datbuf)

integer swathid
character(*) pathname
character(*) attrname
<valid type> datbuf(*)
```

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

```fortran
status = he5_swrdlattr(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_SWdefboxregion is the geolocation track dimension name and the range of that dimension are returned in addition to a regionID. Other than that difference they are the same function and this function is used just like HE5_SWdefboxregion. 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.

```c
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, mode,
    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 *)malloc(size, sizeof(double));
FORTRAN

integer function he5_swreginfo(swathid, regionid, fieldname, ntype, rank, dims, size)

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

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**

```c
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*.

```c
strcpy(aliaslist, "temps 0 to 30");
status = HE5_SWsetalias(swathID, "Temperature", aliaslist);
```

**FORTRAN**

```fortran
integer function he5_swsetalias (swathid, fieldname, aliaslist)
integer swathid
character(*) fieldname
character(*) aliaslist
```

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

```fortran
aliaslist = "temps 0 to 30"
status = he5_swsetalias(swathid, "Temperature", aliaslist)
```
Set Dimension Scale for a Dimension of a Field or Fields within a Swath

**HE5_SWsetdimscale**

herr_t HE5_SWsetdimscale(hid_t swathID, char *fieldname, char *dimname, 
                       const hsize_t dimsize, hid_t numbertype, void *data)

**HE5_SWdefdimscale**

herr_t HE5_SWdefdimscale(hid_t swathID, char *dimname, 
                         const hsize_t dimsize, hid_t numbertype, void *data)

- **swathID** IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
- **fieldname** IN: Name of the field whose **dimname** dimension scale is set
- **dimname** IN: The dimension for which scale is set in the field
- **dimsize** IN: The size of the dimension for which dimension is set
- **numbertype** IN: The number type of the data stored in the scale. See Appendix A for number types.
- **data** IN: Values to be written to the dimension scale

**Purpose**

HE5_SWsetdimscale() Sets dimension scale for a field dimension within the swath.

HE5_SWdefdimscale() Sets dimension scale for a dimension for all fields created before this call 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, non-existing field, or having the same dimension set before (for HE5_SWsetdimscale).

**Description**

These routines set dimension scale for a field (or fields) dimension within the swath. Once the dimension scale is set user can write label, unit, format and other attributes to it using HE5_SWwritedscaleattr().

**Example 1**

In this example, we set dimension scale for the “Bands” dimension in the Spectra field, defined by:

```c
status = HE5_SWdefdatafield(swathID, "Spectra",
                           "Bands,DataTrack,DataXtrack", H5T_NATIVE_FLOAT, 0);
int bands[5] = {1,3,6,7,9};
```
hsizet nbands = 5;

status = HE5_SWsetdimscale(swathID, "Spectra", "Bands",
                           nbands, H5T_NATIVE_INT, bands);

FORTRAN

integer function he5_swsetdimscale(swathid, fieldname, dimname, dimsize, numbertype, data)
integer*4 swathid
character(*) fieldname
character(*) dimname
integer*4 dimsize
integer*4 numbertype
<valid type> data(*)

The equivalent FORTRAN code for the example above is:
integer*4 bands(5)
integer*4 nbands
nbands = 5
bands(1) = 1
bands(2) = 3
bands(3) = 6
bands(4) = 7
bands(5) = 9
status = he5_swsetdimscale(swathid, "Spectra", "Bands",
                           nbands, H5T_NATIVE_INT, bands);

Example 2

In this example, we set dimension scale for the “Bands” dimension in all fields, defined by HE5_SWdefdatafield() in the swath:

int bands[5] = {1,3,6,7,9};
hsizet nbands = 5;
status = HE5_SWdefdimscale(swathID, "Bands",
                           nbands, H5T_NATIVE_INT, bands);

FORTRAN

integer function he5_swdefdimscale(swathid, dimname, dimsize, numbertype, data)
integer*4 swathid
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 (currently not used in the function. For future use)
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.

The fillvalue setting will affect all fields defined after calling HE5_SWsetfillvalue.

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);

FORTRAN integer function he5_swsetfill(swathid, fieldname, ntype, fillvalue)

integer swathid
character(*) fieldname
integer ntype
<valid type> fillvalue(*)

The equivalent FORTRAN 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)

- \textit{swathID} (IN): Swath ID returned by HE5_SWcreate or HE5_SWattach
- \textit{fldgroup} (IN): Field group flag
- \textit{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

\begin{verbatim}
status = HE5_SWunmount(swathID, HE5_HDFE_DATAGROUP, fileID);
\end{verbatim}

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[])

- **swathID** IN: Swath ID returned by HE5_SWcreate or Swattach.
- **regionID** IN: Region ID returned by HE5_SWdefboxregion.
- **indexin** IN: The array containing the indices of the data dimension to which each geolocation element corresponds.
- **indexout** OUT: The array containing the indices of the data dimension to which each geolocation corresponds in the subsetted region. The indexout set to NULL, will not be returned.
- **indices** 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 HE5_SWdefboxregion.

In the first call, set index_region to NULL:

```c
/* 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);
```

**FORTRAN**

integer*4 function he5_swupimap(swathid, regionid, indexin, indexout, indicies)
integer swathid
integer regionid
integer*4 indexin(*)
integer*4 indexout(*)
integer*4 indices(2)

The equivalent *FORTRAN* 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 object. 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**

```c
herr_t HE5_SWwritedatameta(hid_t swathID, const char *fieldname, char *dimlist, int mvalue)
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>swathID</code></td>
<td>Swath ID returned by HE5_SWcreate or HE5_SWattach</td>
</tr>
<tr>
<td><code>fieldname</code></td>
<td>Name of field</td>
</tr>
<tr>
<td><code>dimlist</code></td>
<td>The list of data dimensions defining the field</td>
</tr>
<tr>
<td><code>mvalue</code></td>
<td>The number type of the data stored in the field</td>
</tr>
</tbody>
</table>

**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.

```c
status = HE5_SWwritedatameta(swathID, "Band_1", "GeoTrack, GeoXtrack", H5T_NATIVE_FLOAT);
```

**FORTRAN**

```fortran
integer function he5_swwrdmeta(swathid,fieldname,dimlist,mvalue)
integer swathid
character(*) fieldname
character(*) dimlist
integer mvalue
```

The equivalent FORTRAN code for the example above is:

```fortran
parameter (HE5T_NATIVE_FLOAT = 10)
status = he5_swwrdmeta(swathid, "Band_1", "GeoXtrack, GeoTrack", HE5T_NATIVE_FLOAT)
```

The dimensions are entered in FORTRAN order with the first dimension being incremented first.
Write/Update Attribute for a Dimension scale within a Swath

HE5_SWwritedscaleattr

herr_t HE5_SWwritedscaleattr(hid_t swathID, const char *dimname,
    const char *attrname, hid_t ntype, hsize_t count[], void *datbuf)

swathID IN:  Swath ID returned by HE5_SWcreate or HE5_SWattach
dimname IN:  Dimension scale name for which attribute is written
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 a dimension scale attribute in a specific swath

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

Description If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call.

Example In this example, we write attributes label, unit, format, MissingValues, and IntValues for the Bands dimension scale:

```c
strcpy(label, "Bands Dim");
strcpy(unit, "None");
strcpy(format, "I2");

count[0]= 12;
status = HE5_SWwritedscaleattr(SWid1, "Bands",
    "label", H5T_NATIVE_CHAR, count, label);

count[0]= 6;
```
status = HE5_SWwritedscaleattr(SWid1, "Bands",
    "unit", H5T_NATIVE_CHAR, count, unit);

    count[0]= 4;
status = HE5_SWwritedscaleattr(SWid1, "Bands",
    "format", H5T_NATIVE_CHAR, count, format);

    int datbuf_i1[1] = {-999};
count[0]= 1;
status = HE5_SWwritedscaleattr(SWid1, "Bands",
    "MissingValue", H5T_NATIVE_INT, count, datbuf_i1);
    int datbuf_i2[3] = {-999,0,999};
count[0]= 3;
status = HE5_SWwritedscaleattr(SWid1, "Bands",
    "IntValues", H5T_NATIVE_INT, count, datbuf_i2);

FORTRAN

integer function he5_swwritedscaleattr (swathid, dimname, attrname, ntype, count, datbuf)
integer*4 swathid
character(*) dimname
character(*) attrname
integer*4 ntype
integer*4 count(*)
<valid type> datbuf(*)

The equivalent FORTRAN code for the example above is:

integer swid1
integer*4 datbuf_i1(1)
integer*4 datbuf_i2(2)
integer count(2)
count(1)= 12
status = HE5_SWwritedscaleattr(SWid1, "Bands",
   "label", HE5T_NATIVE_CHAR, count, "Bands Dim")

count(1)= 6
status = HE5_SWwritedscaleattr(SWid1, "Bands",
   "unit", HE5T_NATIVE_CHAR, count, "None")

count(1)= 4
status = HE5_SWwritedscaleattr(SWid1, "Bands",
   "format", HE5T_NATIVE_CHAR, count, "I2")

datbuf_i1(1) = -999

count(1)= 1
status = HE5_SWwritedscaleattr(SWid1, "Bands",
   "MissingValue", HE5T_NATIVE_INT, count, datbuf_i1)
datbuf_i(1) = -999
datbuf_i(2) = 0
datbuf_i(3) = 999

count(1)= 3
status = HE5_SWwritedscaleattr(SWid1, "Bands",
   "IntValues", HE5T_NATIVE_INT, count, datbuf_i)
Write Data to a Swath Field

**HE5_SWwritefield**

```c
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 \( (\text{dim} - \text{start}) / \text{stride} \) 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.

```c
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:

```c
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);
```

**FORTRAN**

```fortran
integer function he5_swwrflld(swathid,fieldname,start,stride,edge,data)
he5_swwrcharflld(swathid,fieldname,elemlen,numelem,start,stride,edge, data)
integer swathid
character(*) filename
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> data(*)
```

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

**Note:** `he5_swwrcharflld()` is only for writing an array of character string field. For writing an array of single character field, please use `he5_swwrflld()`.

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

```fortran
real*4 longitude(1000,2000)
integer*4 start(2), stride(2), edge(2)
start(1) = 0
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 1000
edge(2) = 2000
```
status = he5_swrfld(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_swrfld(swathid, "Longitude", start, stride, edge, newtrack)

Note: When writing data to a field with an unlimited dimension you must not write more data than the actual dimension of the field in first call to SWwritefield, otherwise only partial data will be written to the field. You should do this 1 or more calls to SWwritefield. In the first attempt you write less data than or equal to the actual dimension of the field. In the following attempts you can have anything for start and count (count > start), even start of second attempt can be larger than the count of the first attempt. Please note that in the second (and the following attempts) data buffer is written to the file starting from its 0th element.
Write/Update Group Attribute in “Geolocation Fields” Group

HE5_SWwritegeorgpattr

herr_t HE5_SWwritegeorgpattr(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:

```c
count[0] = 1;
attr_val = 3.14;
status = HE5_SWwritegeorgpattr(swathid, "ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```c
attr_val = 3.14159;
status = HE5_SWwritegeorgpattr(swathid, "ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```
integer function he5_swwrgeogattr(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_swwrgeogattr(swathid, "ScalarFloat",
   HE5T_NATIVE_FLOAT,count,datbuf)
Write Field Metadata to an Existing Swath Geolocation Field

HE5_SWritegeometa

herr_t HE5_SWritegeometa(hid_t swathID, const char *fieldname, char *dimlist, int mvalue)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>swathID</td>
<td>Swath ID returned by HE5_SCreate or HE5_SWAttach</td>
</tr>
<tr>
<td>fieldname</td>
<td>Name of field</td>
</tr>
<tr>
<td>dimlist</td>
<td>The list of geolocation dimensions defining the field</td>
</tr>
<tr>
<td>mvalue</td>
<td>The number type of the data stored in the field</td>
</tr>
</tbody>
</table>

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.

```c
status = HE5_SWritegeometa(swathID, "Latitude", "GeoTrack,GeoXtrack",H5T_NATIVE_FLOAT);
```

FORTRAN

integer function

he5_swwrgmeta(swathid,fieldname,dimlist,mvalue)

integer swathid
character(*) fieldname
character(*) dimlist
integer mvalue

The equivalent FORTRAN code for the example above is:

```fortran
parameter (HE5T_NATIVE_FLOAT = 10)
status = he5_swwrgmeta(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:

```c
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:

```c
attr_val = 3.14159;
status = HE5_SWwritegrpattr(swathid, "ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

FORTRAN integer function he5_swwritegrpattr(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_SWwritelocattr**

```c
herr_t HE5_SWwritelocattr(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:

```c
countt[0] = 1;
attr_val = 3.14;
status = HE5_SWwritelocattr(swathid, "DataField", "ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```c
attr_val = 3.14159;
status = HE5_SWwritelocattr(swathid, "DataField", "ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```
FORTRAN

integer function he5_swrrlattr(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_swrrlattr(swathid, "DataField", "ScalarFloat", HE5T_NATIVE_FLOAT,count, datbuf)
Define Profile Data Structure

HE5_PRdefine

herr_t HE5_PRdefine(hid_t swathID, const char *profilename, char *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, datatype_id
character(*) profilename, dimlist, maxdimlist(*)

The equivalent FORTRAN code for the example above is:

parameter (HE5T_NATIVE_INT = 0)

status = he5_prdefine(swathid, "SimpleProfile", dimlist, maxdimlist, HE5T_NATIVE_INT)

Note: Compression and Chunking defined will effect all profile field definitions after that. This will cease when the data is written to the field. All fields defined after that will not be chunked or compressed unless Compression and Chunking are redefined.
Return Information about a Profile Group Swath Attribute

**HE5_PRgrpatrinfo, HE5_PRgrpatrinfo2**

```c
herr_t HE5_PRgrpatrinfo(hid_t swathID, const char *attrname, hid_t *ntype,
                        hsize_t *count)

herr_t HE5_PRgrpatrinfo2(hid_t swathID, const char *attrname, hid_t *ntype,
                        hsize_t *count, hsize_t *size)
```

- **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
- **size** OUT: Buffer size of attribute element

**Purpose**
Returns information about a **group** attribute in the “Profile Fields” **group**. See Section 3.6 of Volume I (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.

```c
status = HE5_PRgrpatrinfo(swathID, "ScalarFloat", &nt, &count);
```

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

**FORTRAN**

```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:

```fortran
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”:

```c
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:

\[ \text{nattr} = \text{HE5_PRinqgrpattrs(swathID, NULL, &strbufsize)}; \]

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

\[ \text{attrnames} = \text{(char *)calloc(strbufsize+1, sizeof(char))}; \]
\[ \text{nattr} = \text{HE5_PRinqgrpattrs(swathID, attrnames, &strbufsize)}; \]

The variable, attrnames, will be set to: "attrOne,attr_2".

FORTRAN integer*4 function he5_prinqgattrs(swathid, attrnames, strbufsize)

integer swathid
character(*) attrnames
integer*4 strbufsize
integer*4 nattr

The equivalent FORTRAN code for the example above is:

\[ \text{nattr} = \text{he5_prinqgattrs(swathid, attrnames, strbufsize)}; \]
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:

```c
nprof = HE5_PRinquire(swathID, profnames, rank, classID);
```

**FORTRAN**

```fortran
integer*4 function he5_prinquire (swathid, profnames, rank, classID)
integer swathid
character(*) profnames
integer rank(*)
integer classID(*)
```

The equivalent FORTRAN code for the first example above is:

```fortran
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 hsize_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_PReclaimspace() 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_PRreclaimsace(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":

```c
status = HE5_PRreadgrpattr(swathID, "ScalarFloat", &data);
```

**FORTRAN**

```fortran
integer function he5_prrdgattr(swathid, attrname, datbuf)

integer swathid
character(*) attrname
<valid type> datbuf(*)
```

The equivalent FORTRAN code for the example above is:

```fortran
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”

```c
status = HE5_PRreclaimspace(swathID, "Profile-2000", buffer);
```

FORTRAN: Not needed.
Write Data to the Profile Swath Structure

**HE5_PRwrite**

```
herr_t HE5_PRwrite(int swathID, const char *profilename, const hsize_t start[],
    const hsize_t stride[], const hsize_t edge[], size_t size, void *datbuf)
```

- **swathID** IN: Swath ID returned by HE5_SWcreate or HE5_SWattach
- **profilename** IN: Profile structure name
- **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
- **size** IN: Size of data buffer (in bytes) for memory allocation routine
- **datbuf** 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":

```c
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);
```
FORTRAN

```
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 FORTRAN 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

endo

endo

start(1) = 0
stride(1) = 1
count(1) = 4

status = he5_prwrite(swathid, “SimpleProfile”, 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:

\[
\text{count[0] = 1;}
\]
\[
\text{attr_val = 3.14;}
\]
\[
\text{status = HE5_PRwritegrpattr(swathid, "ScalarFloat",}
\text{H5T_NATIVE_FLOAT, count, &attr_val);}
\]

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

\[
\text{attr_val = 3.14159;}
\]
\[
\text{status = HE5_PRwritegrpattr(swathid, "ScalarFloat",}
\text{H5T_NATIVE_FLOAT, count, &attr_val);}
\]

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.

Note: The hsize_t typedef uses the largest type of integer available on a machine (typically a 64-bit integer). So when compiling a FORTRAN code in a 64-bit structure one must declare integers as integer*8 (rather than integer *4) for integers whose C equivalent is declared as hsize_t, since underlying C code expects “long” type integer. For 32-bit compilation on a 64-bit machine “integer *4” should work fine.
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)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fid</strong></td>
<td>IN: Grid file ID returned by HE5_GDopen</td>
</tr>
<tr>
<td><strong>gridname</strong></td>
<td>IN: Name of grid to be attached</td>
</tr>
</tbody>
</table>

**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*:

```c
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:

```fortran
gridid = he5_gdattach(fid, "ExampleGrid")
```

**Note:** If unlike the above example user defines a gridname string and then copies the actual name into that string, then it is suggested that user initialize every single character in the gridname string in their code to "\0", before copying gridname into this string [before passing the string into HE5_GDattach() ]. If user is getting the grid name from another call, then user must initialize the gridname string before that call. Failing to do this may result in having some random characters in the gridname and, therefore, failing of HE5_GDAttach().
Return Information about a Grid Attribute

**HE5_GDattrinfo, HE5_GDattrinfo2**

herr_t HE5_GDattrinfo(hid_t gridID, const char *attrname, hid_t *ntype, hsize_t *count)

herr_t HE5_GDattrinfo2(hid_t gridID, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

- **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
- **size** OUT: Buffer size of attribute element

**Purpose**
Returns information about an object attribute in a specific grid object. 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.

```c
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_gdattrinfo(gridid, attrname, ntype, count,)
integer gridid
character(*) attrname
integer ntype
integer*4 count

The equivalent FORTRAN code for the first example above is:

```fortran
status = he5_gdattrinfo(gridid, "ScalarFloat", ntype, count)
```
Write Block SOM Offset

HE5_GDblkSOMoffset

herr_t HE5_GDblkSOMoffset(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_GDblkSOMoffset:

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_GDblkSOMoffset is going to
be called. The GCTP library doesn’t use the this parameter for the SOM
projection so that is used by the HDF-EOS library only. The
HE5_GDblkSOMoffset 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;
status = HE5_GDdefproj(GDid_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

Close an HDF-EOS File

HE5_GDclose

herr_t HE5_GDclose(hid_t fid)

<table>
<thead>
<tr>
<th>fid</th>
<th>IN: Grid file ID returned by HE5_GDopen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Closes file.</td>
</tr>
<tr>
<td>Return value</td>
<td>Returns SUCCEED(0) if successful or FAIL(-1) otherwise.</td>
</tr>
<tr>
<td>Description</td>
<td>This routine closes the HDF-EOS grid file.</td>
</tr>
<tr>
<td>Example</td>
<td>status = HE5_GDclose(fid);</td>
</tr>
</tbody>
</table>

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(*) filename
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**

```c
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

```c
uplft[0]=210584.50041;
uplft[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 \textit{xdim} and \textit{ydim} values are referenced in the field definition routines by the reserved dimensions: \textit{XDim} and \textit{YDim}.

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

In the case of the Geographic projection (linear scale in both longitude latitude), the \textit{upleftpt} and \textit{lowrightpt} arrays contain the longitude and latitude of these points in packed degree format (DDDMMMSSS.SS).

Note:

\textbf{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.

\textbf{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 \textit{meters} in double precision. These coordinates have to be computed using the \texttt{GCTP} software with the same projection parameters that have been specified in the \textit{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: \texttt{PGS\_GCT\_Init()} or \texttt{PGS\_GCT\_Proj()}. More information is contained in the \textit{SDP Toolkit Users Guide for the ECS Project}.
FORTRAN

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 FORTRAN 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.

```plaintext
cornerlon[0] = 0.;  cornerlat[0] = 90.;
cornerlon[1] = 90.;  cornerlat[1] = 0.;
regionID = HE5_GDdefboxregion(GDid, cornerlon, cornerlat);
```

**FORTRAN**

```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:
```
```plaintext
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 between 4 and 128. Shuffling and compression methods involving entropy coding and nearest neighbor coding require a single integer value for k split, which must be a power of 2 between 1 and 256 (HE5_HDFE_COMP_SHUF_SZIP_K13orEC = 16).
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_GDdeffield` for further information. Any portion of a compressed field can then be accessed with the `HE5_GDdeffield` 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.

```c
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**

```c
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

```c
c ompcode = 4;
compparm[0] = 6;
status = HE5_GDdefcomtile(gridID, compcode, compparm, tilerank, tiledim);
```

**FORTRAN**

```fortran
integer function he5_gdde体制机制(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

```fortran
status = he5_gdde体制机制(gridid, compcode, compparm, tilerank, tiledim)
```
Define a New Dimension within a Grid

**HE5_GDdefdim**

```c
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.

```c
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*:

```c
status = HE5_GDdefdim(gridID, "Unlim", H5S_UNLIMITED);
```

**FORTRAN**

```fortran
integer function he5_gddefdim(gridid, fieldname, dim)
  integer gridid
  character(*) fieldname
  integer*4 dim

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

```fortran
parameter (H5S_UNLIMITED_F=-1)
dim = 15
status = he5_gddefdim(gridid, "Band", dim)
status = he5_gddefdim(gridid, "Unlim", H5S_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)

gridID    IN:  Grid ID returned by HE5_GDcreate or HE5_GDattach
fieldname IN:  Name of field to be defined
dimlist   IN:  The list of data dimensions defining the field
maxdimlist IN:  The maximum dimensions list defining the field
ntype     IN:  The number type of the data stored in the field
merge     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 C order, that is, the last dimension is incremented first.

Note: User needs to define tiling and compression before every field definition.
Example In this example, we define a grid field, Temperature with dimensions XDim and YDim (as established by the HE5_GDcreate routine) containing 4-byte floating point numbers and a field, Spectra, with dimensions XDim, YDim, and Bands:

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);
The equivalent FORTRAN code for the example above is:

```fortran
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

```fortran
comppcode = 0;
comppparm[0] = 0;
status = HE5_GDdefcomtile(gridID, compcode, comppparm, 
tilerank, tiledims);
```

or alternatively call

```fortran
status = HE5_GDdeftile(gridID, HE5_HDFE_NOTILE, tilerank, 
tiledims);
status = HE5_GDdefcomp(gridID, HE5_HDFE_COMP_NONE, 
comppparm);
```

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_GDdefpreg(gridID, HE5_HDFE_CORNER);
```

**FORTRAN**

```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_gddefpreg(gridid, HE5_HDFE_CORNER)
```
Define Grid Projection

**HE5_GDdefproj**

```c
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

```c
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 spheriod.

```c
spherecode = 3;
for (i = 0; i < 13; i++) projparm[i] = 0;
/* Set Long below pole & true scale in DDDMMSSS.SSS form */
projparm[5] = 90000000.0;
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.

```c
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
endo
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**

```c
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_HDFE_TILE, HE5_HDFE_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.

```c
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_GDdeffield(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_GDdeffield(gridID, "Communities", "YDim,XDim", NULL, H5T_NATIVE_INT, HE5_HDFE_AUTOMERGE);

FORTRAN integer function he5_gddeftile(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_gddeftle(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_gddeftle(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_gddeftmeper(gridid, periodID, starttime, stoptime)
integer gridid
integer periodid
real*8 starttime
real*8 stoptime

The equivalent FORTRAN code for the examples above are:

parameter (HE5_HDFE_NOPREVSUB=-1)
starttime = 35232487.2
stoptime = 36609898.1
periodid = he5_gddeftmeper(gridid, HE5_HDFE_NOPREVSUB, starttime, stoptime)
periodid = he5_gddeftmeper(gridid, regionid, starttime, stoptime)
Define a Vertical Subset Region

**HE5_GDdefvrtregion**

```c
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. It 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:

```c
range[0] = 500.;
range[1] = 800.;
```
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:

```fortran
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:

```fortran
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:

```fortran
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)
```
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**

```c
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":

```c
dimsize = HE5_GDdiminfo(gridID, "Bands");
```

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

**FORTRAN**

```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")
```

```fortran
  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.

```c
strcpy(aliasname, "temps 0 to 30");
status = HE5_GDdropalias(gridID, HE5_HDFE_DATAGROUP, aliasname);
```

FORTRAN integer function he5_gddropalias (gridid, fldgroup, aliasname)

```fortran
integer gridid
character*(*) fldgroup
character*(*) aliasname

The equivalent FORTRAN code for the first example above is:

```fortran
aliasname = "temps 0 to 30"
status = he5_grdropalias(gridid, HE5_HDFE_DATAGROUP, aliasname)
```
Return Information about a Grid Dimension Scale Attribute

**HE5_GDdscaleattrinfo, HE5_GDdscaleattrinfo2**

herr_t HE5_GDdscaleattrinfo(hid_t gridID, const char *dimname, const char *attrname, hid_t *ntype, hsize_t *count)

herr_t HE5_GDdscaleattrinfo2(hid_t gridID, const char *dimname, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

- **gridID** IN: Grid ID returned by HE5_GDcreate or HE5_GDattach
- **dimname** IN: Dimension scale name
- **attrname** IN: Attribute name
- **ntype** OUT: Number type of attribute
- **count** OUT: Number of attribute elements
- **size** OUT: Buffer size of attribute element

**Purpose** Returns information about attribute(s) in a specific dimension scale.

**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 dimension scale attribute.

**Example** In this example, we return information about the *IntValues* attribute of *Bands* dimension scale.

```c
status = HE5_GDdscaleattrinfo(gridID, "Bands", "IntValues", &ntype, &count);
```

The **ntype** variable will have the value 0 and **count** will have the value of 3.

**FORTRAN**

```fortran
integer function he5_gddscalettrinfo(gridid, fieldname, attrname, ntype, count)
    integer gridid
    character(*) attrname
    integer ntype
    integer *4 count

    The equivalent FORTRAN code for the first example above is:

    status = he5_gddscalettrinfo(gridid, "Bands", "IntValues", ntype, count)
```

**2-212**

EED2-175-002
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:

```c
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)

```fortran
integer oldregionid
```

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

```fortran
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**

```c
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.

```c
datbuf = (float *)calloc(size, sizeof(float));
status = HE5_GDextractregion(GDid, regionID, "Temperature", datbuf);
```

**FORTRAN**

```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:

```fortran
status = he5_gdextreg(gridid, regionid, "Temperature", datbuf)
```
Retrieve Information about Data Field in a Grid

**HE5_GDfieldinfo**

```c
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:

```c
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:

```fortran
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 = 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 Dimension Scale for a Dimension of a Field within a Grid

HE5_GDgetdimscale

long HE5_GDgetdimscale(hid_t gridID, char *fieldname, char *dimname,
                        hsize_t *dimsizes, hid_t *numbertype, void *data)

gridID  IN:  Grid ID returned by HE5_GDcreate or HE5_GDattach
fieldname  IN:  Name of the field whose dimname dimension scale is read
dimname  IN:  The dimension for which scale values are read
dimsizes  OUT:  The size of the dimension to be read
numbertype  OUT:  The number type of the data stored in the scale. See Appendix A for number types.
data  OUT:  Values to be read for the dimension scale

Purpose  Gets dimension scale for a field dimension within the grid.
Return value  Returns data buffer size if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list or none-existing field.

Description  This routine gets dimension scale for a field dimension within the grid. The dimension scales attributes label, unit, format and others can be read using HE5_GDreaddscaleattr ()

Example  In this example, we get dimension scale for the Bands dimension in the Spectra field, defined using HE5_GDsetdimscale():

```c
long buffsize;

hsize_t nbands;

hid_t ntype;

int *bands;

/* First call, with NULL for data buffer, returns */
/* buffersize needed for the data buffer */

buffsize = HE5_GDgetdimscale(gridID, "Spectra", "Bands",
                           &nbands, &ntype, NULL);

/* allocate enough buffer for the data */

bands = (int *)malloc(buffsize);
```
`buffsize = HE5_GDgetdimscale(gridID, "Spectra", "Bands",
    &nbands, &ntype,(void *)bands);`

**FORTRAN**

```
integer function he5_gdgetdimscale(gridid, fieldname, dimname, dimsize, numbertype, data)
    integer*4    gridid
    character(*) fieldname
    character(*) dimname
    integer*4    dimsize
    integer*4    numbertype
    <valid type> data(*)

The equivalent FORTRAN code for the example above is:

integer*4    bands(15)
integer*4    nbands, ntype, buffsize

buffsize = he5_gdgetdimscale(gridid, "Spectra", "Bands",
    nbands, ntype, bands);
```
Get External Data File Information

**HE5_GDgetextdata**

```c
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:

```c
nfiles = HE5_GDgetextdata(gridID, "ExtData", namlen, filenames, offset, size);
```

**FORTRAN**

```fortran
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 *FORTRAN* code for the example above is:

```fortran
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);

FORTRAN integer function he5_gdgetfill(gridid,fieldname,fillvalue)

integer gridid
character*(*) fieldname
<valid type> fillvalue(*)

The equivalent FORTRAN 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:

```c
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.
FORTRAN

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 FORTRAN 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**

```c
long HE5_GDgetpixvalues(hid_t gridID, long nPixels, long pixRow[], long pixCol[], const char *fieldname, void *buffer)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gridID</td>
<td>hid_t</td>
<td>Grid ID returned by HE5_GDcreate or HE5_GDattach</td>
</tr>
<tr>
<td>nPixels</td>
<td>long</td>
<td>Number of pixels</td>
</tr>
<tr>
<td>pixRow</td>
<td>long[]</td>
<td>Pixel Rows</td>
</tr>
<tr>
<td>pixCol</td>
<td>long[]</td>
<td>Pixel Columns</td>
</tr>
<tr>
<td>fieldname</td>
<td>const char*</td>
<td>Field from which to extract data values</td>
</tr>
<tr>
<td>buffer</td>
<td>void *</td>
<td>Buffer for data values</td>
</tr>
</tbody>
</table>

**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 (i.e., 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:

```c
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 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

```c
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 of the upper left and lower right corners of the grid image. For all projections the unit for upleft and lowright coordinates will be in meters, except for the Geographic Projection, where the units will be in DMS degrees.

### Example

In this example, we retrieve information from a previously created grid with a call to `HE5_GDattach`:

```c
call status = HE5_GDgridinfo(gridID, &xdimsize, &ydimsize, upleft, lowrgt);
```

### FORTRAN

```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)
end
```

The equivalent FORTRAN code for the example above is:

```fortran
call status = he5_gdgridinfo(gridid, xdimsize, ydimsize, upleft, lowrgt)
```
Return Information about a Group Grid Attribute

**HE5_GDgrpattrinfo, HE5_GDgrpattrinfo2**

```c
herr_t HE5_GDgrpattrinfo(hid_t gridID, const char *attrname, hid_t *ntype,
                          hsize_t *count)
```

```c
herr_t HE5_GDgrpattrinfo2(hid_t gridID, const char *attrname, hid_t *ntype,
                           hsize_t *count, hsize_t *size)
```

**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

**size**  
OUT: Buffer size of attribute element

**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.

```c
status = HE5_GDgrpattrinfo(gridID, "ScalarFloat", &ntype, &count);
```

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

**FORTRAN**

```fortran
integer function he5_gdgattrinfo(gridid, attrname, ntype, count,)
integer     gridid
character(*) attrname
integer     ntype
integer *4  count
```

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

```fortran
status = he5_gdgattrinfo(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 object. 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, attrnames, will be set to:
"attrOne,attr_2".

FORTRAN integer*4 function he5_gdinqattrs(gridid,attrnames,strbufsize)
integer        gridid
character(*)    attrnames
integer*4      strbufsize

The equivalent FORTRAN code for the example above is:

nattr = he5_gdinqattrs(gridid, attrnames, strbufsize)
Return Data Type Information about Data Fields in Grid

**HE5_GDinqdatatype**

```c
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.

```c
status = HE5_GDinqdatatype(gridID, "Spectra", NULL, fieldgroup, &datatype, &classid, &order, &size);
```

**FORTRAN**

```fortran
integer function he5_gdinqdatatype(gridid, fieldname, attrname, fldgrp, dtype, classid, order, size)
  integer                    gridid
  integer                    dtype, classid, order
  integer*4                  size
```
character *(*) *fieldname*

integer \( \text{HE5\_HDFE\_DATAGROUP} \)

parameter \( \text{(HE5\_HDFE\_DATAGROUP}=1) \)

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

```fortran
status = he5_gdinqdatatype(gridid, "Spectra", " ",
HE5_HDFE_DATAGROUP, dtype, classid, order, size)
```
Retrieve Information about Dimensions Defined in Grid

**HE5_GDinqdims**

```c
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:

```c
ndim = HE5_GDinqdims(gridID, dimnames, dims);
```

The parameter, `dimnames`, will have the value: "Xgrid,Ygrid,Bands" with `dims[3]`={120,200,15}

**FORTRAN**

```fortran
integer function he5_gdinqdims(gridid, dimnames, dims)

integer         gridid
character(*)    dimnames
integer(*)     dims(*)
```

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

```fortran
ndim = he5_gdinqdims(gridid, dimnames, dims)
```
Retrieve Information for Grid Dimension Scale Attributes

HE5_GDinqdscaleattrs

long HE5_GDinqdscaleattrs(hid_t gridID, const char *dimname,
            char *attrnames, long *strbufsize)

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

dimname IN: Dimension scale name to retrieve attribute information

attrnames OUT: Attribute list (entries separated by commas)

strbufsize OUT: String length of attribute list

Purpose Retrieve information about the attributes defined for a specific dimension scale.

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 dimension scale
attributes defined for a field “Bands”. In the first call, set the parameter
attrnames to NULL. We assume that there are five attributes stored, label, unit, format, MissingValue, and IntValues :

nattr = HE5_GDinqdscaleattrs(gridID, “Bands”, NULL, &strbufsize);

The parameter, nattr, will have the value 5 and strbufsize will have value
40.

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

nattr = HE5_GDinqdscaleattrs(gridID, “Bands”, attrnames, &strbufsize);

The variable, attrlist, will be set to:

"label,unit,format,MissingValue,IntValues ".

FORTRAN integer*4 function he5_gdinqdscaleattrs(gridid , dimname, attrnames, strbufsize)

integer gridid
character(*), save :: dimname
character(*), save :: attrnames
integer*4 :: strbufsize

The equivalent FORTRAN code for the example above is:

nattr = he5_gdinqdscaleattrs(gridid, "Bands", attrnames, strbufsize)
Retrieve Information about Data Fields Defined in Grid

HE5_GDinqfields

int HE5_GDinqfields(hid_t gridID, char *fieldlist, int rank[], hid_t ntype[])

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"

FORTRAN integer function he5_gdinqdflds(gridid, fieldlist, rank, numbertype)

gridid integer
fieldlist character(*)
rank(*) integer
numbertype(*) integer

The equivalent FORTRAN code for the example above is:
nfld = he5_gdinqdflds(gridID, fieldlist, rank, numbertype)
The parameter, fieldlist, will have the value: "Spectra,Temperature"
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:

```c
nfldalias = HE5_GDinqfldalias(gridID, NULL, &strbufsize);
```

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

```c
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_gdinqlfldalias(gridid, fldalias, strbufsize)

integer  gridid
integer*4  strbufsize
integer*4  nfldalias
caracter*(*)  fldalias

The equivalent FORTRAN code for the example above is:

```fortran
nfldalias = he5_gdinqlfldalias(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 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));

gridlist = 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_GDInqgrpatttrs

long HE5_GDInqgrpatttrs(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_GDInqgrpatttrs(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_GDInqgrpatttrs(gridID, attrnames, &strbufsize);

The variable, attrlist, will be set to:
"attrOne,attr_2".

FORTRAN integer*4 function he5_gdinqgattrs(gridid, attrnames, strbufsize)

integer gridid
character(*), intent(inout) attrnames
integer*4 strbufsize

The equivalent FORTRAN code for the example above is:

nattr = he5_gdinqgattrs(gridid, attrnames, strbufsize)
Retrieve Information Grid Local Attributes

**HE5_GDinqlocattrs**

```c
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`:

```c
nattr = HE5_GDinqlocattrs(gridID, "DataField", NULL, &strbufsize);
```

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

```c
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**

```fortran
integer*4 function he5_gdinqlattrs(gridid,fieldname,attrnames,strbufsize)
  integer gridid
  character(*) filename
  character(*) attrnames
```
integer*4 \textit{strbufsize}

The equivalent \textit{FORTRAN} code for the example above is:

\begin{verbatim}
nattr = he5_gdinqlattrs(gridid, "DataField", attrnames, strbufsize)
\end{verbatim}
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 (2nd 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;
lamVal[0] = -20.8;
lonVal[1] = 15.8;
lamVal[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);

FORTRAN

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 FORTRAN 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, HE5_GDlocattrinfo2**

```c
herr_t HE5_GDlocattrinfo(hid_t gridID, const char *fieldname, const char *attrname, hid_t *ntype, hsize_t *count)
```

```c
herr_t HE5_GDlocattrinfo2(hid_t gridID, const char *fieldname, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)
```

- **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
- **size**  OUT: Buffer size of attribute element

**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.

**FORTRAN**
```
integer function he5_gdlattrinfo(gridid,fieldname,attrname,ntype,count)
integer    gridid
character(*)  fieldname
character(*)  attrname
integer    ntype
integer *4    count
```
The equivalent FORTRAN code for the first example above is:
```
status = he5_gdlattrinfo(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)

- **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 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_RDWRONLY=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 SDP Toolkit User Guide for the EOSDIS Evolution and Development Project (333-EED2-001, Revision 01), 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 Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Also the default value (0) will be returned for origincode if the value was not found in the Structure Metadata.

Description This routine retrieves the origin code.

Example In this example, we retrieve the origin code defined in HE5_GDdeforigin.

```c
status = HE5_GDorigininfo(gridID, &origincode);
```

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

FORTRAN integer function he5_gdorginfo(gridid,origincde)

```fortran
integer gridid
integer(*) origincode
```

The equivalent FORTRAN code for the above example is :

```fortran
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 Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Also the default value (0) will be returned for pixregcode if the value was not found in the Structure Metadata.

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:

```c
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:

```fortran
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)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>gridID</td>
<td>IN: Grid ID returned by HE5_GDcreate or HE5_GDattach</td>
<td>status = HE5_GDreadattr(gridID, &quot;ScalarFloat&quot;, &amp;attr_val);</td>
</tr>
<tr>
<td>attrname</td>
<td>IN: Attribute name</td>
<td>integer function he5_gdrdattr(gridid, attrname, datbuf)</td>
</tr>
<tr>
<td>datbuf</td>
<td>OUT: Buffer allocated to hold attribute values</td>
<td>The equivalent FORTRAN code for the example above is:</td>
</tr>
</tbody>
</table>

Purpose: Reads attribute from a specific grid object. 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);
```
Read Attribute for a Dimension scale within a Grid

**HE5_GDreaddscaleattr**

```c
herr_t HE5_GDreaddscaleattr(hid_t gridID, const char *dimname,
                              const char *attrname, void *datbuf)
```

- **gridID**
  - IN: Grid ID returned by HE5_GDcreate or HE5_GDattach
- **dimname**
  - IN: Dimension scale name for which attribute is written
- **attrname**
  - IN: Attribute name
- **datbuf**
  - OUT: Buffer allocated to hold attribute values

**Purpose**
Reads a dimension scale attribute from a specific dimension.

**Return value**
Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper grid id 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 attributes of the Bands dimension scale:

```c
herr_t status = FAIL;
hid_t GDid1 = FAIL;
int i;
long nattr;
long strbufsize;
char *attrlist;
size_t fldnmlen[HE5_HDFE_NAMBUFSIZE];
char *fldnm[HE5_HDFE_NAMBUFSIZE];
char *attrname = (char *)NULL;
hid_t *ntype;
hsize_t count = 0;
void *attr;
int *attr_int;
float *attr_flt;
float *attr_dbl;
char *attr_char;
```

```
nattr = HE5_GDinqdscaleattrs(GDid1, "Bands", NULL, &strbufsize);
attrlist = (char *) calloc(strbufsize + 2, sizeof(char));
nattr = HE5_GDinqdscaleattrs(GDid1, "Bands")
ntype, attrlist, &strbufsize);
nattr = HE5_EHparsestr(attrlist, ",", fldnm, fldnmlen);
for( i = 0; i < nattr; i++)
```
{  
    attrname = (char *)calloc(fldnmlen[i] + 1, sizeof(char));  
    memmove(attrname, fldnm[i], fldnmlen[i]);  
    ntype = (hid_t *)calloc(1, sizeof(hid_t));  
    if(strcmp(attrname, "REFERENCE_LIST") == 0 )  
    {  
        continue;  
    }  
    status = HE5_GDdscaleattrinfo(GDid1,"Bands",  
                                  attrname, ntype, &count);  
    if( (int)*ntype == 0 )  
    {  
        attr_int = (int *)malloc(count * sizeof(int));  
        attr = (void *) attr_int;  
    }  
    if( (int)*ntype == 10 )  
    {  
        attr_flt = (float *)malloc(count * sizeof(float));  
        attr = (void *) attr_flt;  
    }  
    if( (int)*ntype == 11 )  
    {  
        attr_dbl = (double *)malloc(count * sizeof(double));  
        attr = (void *) attr_dbl;  
    }  
    if( (int)*ntype == 57 )  
    {  
        attr_char = (char *)malloc((count+1) * sizeof(char));  
        attr = (void *) attr_char;  
    }  
    status = HE5_GDreaddscaleattr(GDid1,"Bands",attrname, attr);  
}

FORTRAN  
integer function he5_gdreaddscaleattr (gridid, dimname, attrname, datbuf)  
integer*4 gridid  
character*(*) dimname  
character*(*) attrname  
<valid type> datbuf(*)

The equivalent FORTRAN code for the example above is:

integer j, ntype  
integer gdid1  
integer attr_int(25)  
real*4 attr_flt(25)  
real*8 attr_dbl(25)  
character attr_char(25)  
integer nattr  
character*100 attrlist  
character*100 strbufsize  
character*15 attrname(10)
nattr = HE5_GDinqdscaleattrs(GDid1, "Bands", attrlist, strbufsize)

attrname(1) = 'label'
attrname(2) = 'unit'
attrname(3) = 'format'
attrname(4) = 'MissingValue'
attrname(5) = 'IntValues'
do j = 1,5
  attr_char = ''
count(1)= 0
count(2)= 0
  status = HE5_GDdscaleattrinfo(GDid1,"Bands", attrname(j), ntype, count)
  if( ntype .eq. 0) then
    status = HE5_GDreaddscaleattr(GDid1,"Bands", attrname(j), attr_int)
  endif
  if( ntype .eq. 10) then
    status = HE5_GDreaddscaleattr(GDid1,"Bands", attrname(j), attr_flt)
  endif
  if( ntype .eq. 11) then
    status = HE5_GDreaddscaleattr(GDid1,"Bands", attrname(j), attr_dbl)
  endif
  if( ntype .eq. 57) then
    status = HE5_GDreaddscaleattr(GDid1,"Bands", attrname(j), attr_char)
  endif
enddo
**HE5_GDreadfield**

```c
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 of 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.

```c
float row[120];

hssize_t start[2]={10,0}; 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,elemlen,numelem,start,stride,edge,buffer)

integer gridid
character*(*) filename
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_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(120)
integer*4 start(2), stride(2), edge(2)
start(1) = 0
start(2) = 10
stride(1) = 1
stride(2) = 1
edge(1) = 120
edge(2) = 1
status = he5_gdrdfld(gridid, "Temperature", start, stride, edge, row)
Read Group Grid Attribute

**HE5_GDreadgrpattr**

```c
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".

```c
status = HE5_GDreadgrpattr(gridID, "ScalarFloat", &attr_val);
```

**FORTRAN**  
```fortran
integer function he5_gdrdgattr(gridid, attrname, datbuf)

integer gridid
character(*) attrname
<valid type> attrval(*)
```

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

```fortran
status = he5_gdrdgattr(gridid, "ScalarFloat", attrval)
```
Read Local Grid Attribute

**HE5_GDreadlocattr**

```c
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":

```c
status = HE5_GDreadlocattr(gridID, "DataField", "ScalarFloat", &attr_val);
```

**FORTRAN**
integer function he5_gdrdlattr(gridid, filename, attrname, datbuf)

```fortran
integer gridid
character(*) filename
character(*) attrname
<valid type> attrval(*)
```

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

```fortran
status = he5_gdrdlattr(gridid, "DataField", "ScalarFloat", attrval)
```
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(GDid, 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*.

```c
strcpy(aliaslist, "temps 0 to 30");
status = HE5_GDsetalias(gridID, "Temperature", aliaslist);
```

**FORTRAN** integer function he5_gdsetalias (gridid, fieldname, aliaslist)

```fortran
integer gridid  
character*(*) fieldname  
character*(*) aliaslist
```

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

```fortran
aliaslist = "temps 0 to 30"  
status = he5_gdsetalias(gridid, "Temperature", aliaslist)
```
Set Dimension Scale for a Dimension of a Field or Fields within a Grid

**HE5_GDsetdimscale**

herr_t HE5_GDsetdimscale(hid_t gridID, char *fieldname, char *dimname,
                         const hsize_t dimsize, hid_t numbertype, void * data)

**HE5_GDdefdimscale**

herr_t HE5_GDdefdimscale(hid_t gridID, char *dimname,
                         const hsize_t dimsize, hid_t numbertype, void * data)

- **gridID** IN: Grid ID returned by HE5_GDcreate or HE5_GDattach
- **fieldname** IN: Name of the field whose **dimname** dimension scale is set
- **dimname** IN: The dimension for which scale is set in the field
- **dimsize** IN: The size of the dimension for which dimension is set
- **numbertype** IN: The number type of the data stored in the scale. See Appendix A for number types.
- **data** IN: Values to be written to the dimension scale

**Purpose**

HE5_GDsetdimscale sets dimension scale for a dimension in a field within the grid.

**Return value**

Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list, non-existing field, or having the same dimension set before.

**Description**

These routines set dimension scale for a field (or fields) dimension within the grid. Once the dimension scales is set user can write label, unit, format, and other attributes to it using HE5_GDwritedscaleattr ()

**Example 1**

In this example, we set dimension scale for the “Bands” dimension in the Spectra field, defined by:

status = HE5_GDdefdatafield( gridID, "Spectra",
                             "Bands,YDim,XDim", H5T_NATIVE_FLOAT,
                             HDF_E_NOMERGE);

int bands[15] = {1,2,3,4,5,6,7,10,11,12,13,14,15,16,17};
hsizet nbands = 15;
status = HE5_GDsetdimscale(gridID, "Spectra", "Bands",
                            nbands, H5T_NATIVE_INT, bands);
integer function he5_gdsetdimscale(gridid, fieldname, dimname, dimsize, numbertype, data)

  integer*4      gridid  
  character(*)   fieldname
  character(*)   dimname
  integer*4     dimsize     
  integer*4     numbertype
  <valid type>   data(*)     

The equivalent FORTRAN code for the example above is:

  integer*4     bands(15)  
  integer*4     nbands       
  nbands = 15    
  bands(1) = 1   
  ................................
  ................................
  bands(15) = 17 
  status = he5_gdsetdimscale(gridid, "Spectra", "Bands", 
                             nbands, HE5T_NATIVE_INT, bands);  

Example 2  In this example, we set dimension scale for the “Bands” dimension in all field, defined by HE5_GDdefdatafield() in the grid:

  int bands[15] = {1,2,3,4,5,6,7,10,11,12,13,14,15,16,17};
  hsize_t nbands = 15;
  status = HE5_GDdefdimscale(gridID, "Bands", 
                             nbands, H5T_NATIVE_INT, bands);  

FORTRAN  integer function he5_gddefdimscale(gridid, dimname, dimsize, numbertype, data)

  integer*4 gridid

Note: When setting dimension scale for XDim or YDim we need to use NULL for data buffer. HDF-EOS will calculate buffer values itself using internal grid corner values and xdim or ydim.

  xdim = 120;
  ydim = 200;
  status = HE5_GDsetdimscale(GDid1,"Vegetation", "XDim", xdim, 
                             H5T_NATIVE_DOUBLE, NULL);
  status = HE5_GDsetdimscale(GDid1,"Vegetation", "YDim", ydim, 
                             H5T_NATIVE_DOUBLE, NULL);

In Fortran one needs to declare buffer for XDim and YDim dimension scale buffere values, but they need not be populated before passing to gdsetdimscale().

  real*4      veg1(120),veg2(200)
  xdim = 120;
ydim = 200;
status = HE5_GDsetdimscale(GDid1, "Pollution", "XDim", xdim,
1  HE5T_NATIVE_DOUBLE, veg1)

status = HE5_GDsetdimscale(GDid1,"Vegetation", "YDim", ydim,
1  HE5T_NATIVE_DOUBLE, veg2)
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**

```c
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:

```c
tempfill = -999.0;
status = HE5_GDsetfillvalue(gridID, "Temperature", ntype, &tempfill);
```

**FORTRAN**
Integer function he5_gdsetfill(gridid,fieldname,ntype,fillvalue)

```fortran
integer*4 gridid
character(*) fieldname
integer ntype
<valid type> fillvalue(*)
```

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

```fortran
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:

```c
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**

```c
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 object. 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:

```c
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:

```c
attr_val = 3.14159;

status=HE5_GDwriteattr(gridid,"ScalarFloat",H5T_NATIVE_FLOAT, count, &attr_val);
```
FORTRAN

integer function he5_gdwrattr(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_gdwrattr(gridid,"ScalarFloat",HE5T_NATIVE_FLOAT, count, attrval)
Write/Update Attribute for a Dimension scale within a Grid

HE5_GDwritedscaleattr

herr_t HE5_GDwritedscaleattr(hid_t gridID, const char *dimname,
   const char *attrname, hid_t ntype, hsize_t count[], void *datbuf)

gridID  IN:  Grid ID returned by HE5_GDcreate or HE5_GDattach

dimname IN:  Dimension scale name for which attribute is written

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 a dimension scale attribute in a specific grid

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

Description  If the attribute does not exist, it is created. If it does exist, then the value(s)
   is (are) updated. The attribute is passed by reference rather than value in
   order that a single routine suffice for all numerical types. Because of this a
   literal numerical expression should not be used in the call.

Example  In this example, we write attributes label, unit, format, MissingValues, and
   IntValues for the Bands dimension scale:

   strcpy(label, "Bands Dim");

   strcpy(unit, "None");

   strcpy(format, "I2");

   count[0]= 12;

   status = HE5_GDwritedscaleattr(GDid1, "Bands",
                                 "label", H5T_NATIVE_CHAR, count, label);

   count[0]= 6;

   status = HE5_GDwritedscaleattr(GDid1, "Bands",
                                 "unit", H5T_NATIVE_CHAR, count, unit);

   count[0]= 4;
status = HE5_GDwritedscaleattr(GDid1, "Bands",
   "format", H5T_NATIVE_CHAR, count, format);
int datbuf_i1[1] = {-999};
count[0]= 1;
status = HE5_GDwritedscaleattr(GDid1, "Bands",
   "MissingValue", H5T_NATIVE_INT, count,
   datbuf_i1);
int datbuf_i2[3] = {-999,0,999};
count[0]= 3;
status = HE5_GDwritedscaleattr(GDid1, "Bands",
   "IntValues", H5T_NATIVE_INT, count,
   datbuf_i2);

**FORTRAN**

```fortran
integer function he5_gdwritedscaleattr (gridid, dimname, attrname, ntype, count, datbuf)
integer*4     gridid
character(*)  dimname
character(*)  attrname
integer*4     ntype
integer*4     count(*)
<valid type>  datbuf(*)
```

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

```fortran
integer     gdid1
integer*4   datbuf_i1(1)
integer*4   datbuf_i2(2)
integer     count(2)
count(1)= 12
status = HE5_GDwritedscaleattr(GDid1, "Bands",
   "label", H5T_NATIVE_CHAR, count, "Bands Dim")
count(1)= 6
status = HE5_GDwritedscaleattr(GDid1, "Bands",
   "unit", H5T_NATIVE_CHAR, count, "None")
count(1)= 4
```
status = HE5_GDwritedscaleattr(GDid1, "Bands",
   "format", HE5T_NATIVE_CHAR, count, "I2")
datbuf_i1(1) = -999
count(1) = 1
status = HE5_GDwritedscaleattr(GDid1, "Bands",
   "MissingValue", HE5T_NATIVE_INT, count, datbuf_i1)
datbuf_i(1) = -999
datbuf_i(2) = 0
datbuf_i(3) = 999
count(1) = 3
status = HE5_GDwritedscaleattr(GDid1, "Bands",
   "IntValues", HE5T_NATIVE_INT, count, datbuf_i)
Write Data to a Grid Field

**HE5_GDwritefield**

```c
herr_t HE5_GDwritefield(hid_t gridID, const char *fieldname, const hssize_t start[], const hsize_t stride[], const hsize_t edge[], void *data)
```

- `gridID` **IN**: Grid ID returned by HE5_GDcreate or HE5_GDattach
- `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 grid field.

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

**Description**
The values within `start`, `stride`, and `edge` arrays refer to the grid 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 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 `HE5_GDdeftile` for further information.

**Example**
In this example, we write data to the `Temperature` field (ydim=2000, xdim=1000).

```c
float temperature [2000][1000];
/* Define elements of temperature array */
status = HE5_GDwritefield(gridID, "Temperature", NULL, NULL, NULL, temperature);
```

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

```c
float newrow[1000];
```

```c
hssize_t start[2]={10,0}; hsize_t edge[2]={1, 1000};
/* 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,elemlen,nuemelem,start,stride,edge,data)

integer gridid
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> 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(1000, 2000)
integer*4 start(2), stride(2), edge(2)
start(1) = 0
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 1000
edge(2) = 2000
status = he5_gdwrfld(gridid, "Temperature", start, stride, edge, temperature)

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

real*4 newrow(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_gdwrfld(gridid, "Temperature", start, stride, edge, newrow)

Note: When writing data to a field with an unlimited dimension you must not write more data than the actual dimension of the field in first call to GDwritefield, otherwise only partial data will be written to the field. You should do this 1 2 or more calls to GDwritefield. In the first attempt you write less data than or equal to the actual dimension of the field. In the following attempts you can have anything for start and count (count > start), even start of second attempt can be larger than the count of the first attempt. Please note that in the second (and the following attempts) data buffer is written to the file starting from its 0th element.
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"l, "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:

```c
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:

```c
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
class

character(*) fieldname
class

character(*) attrname
class

integer*4 count(*)
class

<valid type> attrval(*)
class

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.

Note: The hsize_t typedef uses the largest type of integer available on a machine (typically a 64-bit integer). So when compiling a FORTRAN code in a 64-bit structure one must declare integers as integer*8 (rather than integer *4) for integers whose C equivalent is declared as hsize_t, since underlying C code expects “long” type integer. For 32-bit compilation on a 64-bit machine “integer *4” should work fine.
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 an integral number of degrees and minutes and a float point value of seconds packed as a single double number as follows: DDDMMSSSS.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)
- 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 HDF-EOS that was used to create the file. This string is of the form: “HDFEOS_Vmaj.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_EHglbattrinfo, HE5_EHglbattrinfo2

herr_t HE5_EHglbattrinfo(hid_t fileID, const char *attrname, hid_t *ntype, hsize_t *count)

herr_t HE5_EHglbattrinfo2(hid_t fileID, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

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

size OUT: Buffer size of attribute element

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_EHglbattrinfo(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)
Determine if the Data File is HDF-EOS5 Product

**HE5_EHHEisHE5**

```c
int HE5_EHHEisHE5(char *filename)
```

- **filename** IN: Name of the file.
- **Purpose** Determines if the input file type is HDF-EOS5
- **Return value** Returns TRUE (1) if file is HDF-EOS5, FALSE (0) if file is not HDF_EOS5, or FAIL (-1) otherwise. Typical reason for failure is failing to open the file.
- **Description** This routine tries to open the file with HDF_EOS5 calls and find at least one of the objects SWATH, GRID, POINT, or ZA in the file. If successful, the file is HDF-EOS5, otherwise a different type.
- **Example** In this example, we check the type of HDF file to see if it is HDF-EOS5 type:
  ```c
  int fileIsHe5 = -1;
  char *filename = "testHDF.hdf";
  fileIsHe5 = HE5_EHHEisHE5(filename);
  ```

  **FORTRAN**

  ```fortran
  integer function he5_ehheishe5(filename)
  character(*) filename

  The equivalent FORTRAN code for the example above is:
  ```fortran
  integer fileIsHe5
  fileIsHe5 = he5_ehheishe5(filename)
  ```
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 by used to create or access HDF5 structures such as groups, attributes, datasets within an HDF-EOS file.
Retrieve Information about Global File Attributes

HE5_EHinqglbattrs

long HE5_EHinqglbattrs(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_EHinqglbattrs(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_EHinqglbattrs(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_POpen

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_EHreadglobbattr**

```c
herr_t HE5_EHreadglobbattr(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":

```c
status = HE5_EHreadglobbattr(fileID, "FloatAttr", &data);
```

**FORTRAN**

```fortran
integer function he5_ehrdglatt(fileid,attrname,datbuf)
```

```fortran
type(fileid) fileid

type(char) attrname

type(valid type) datbuf
```

The equivalent FORTRAN code for the example above is:

```fortran
status = he5_ehrdglatt(fileid, "FloatAttr", datbuf)
```
Set Flag for Suppressing HDF5 Error Messages

**HE5_EHset_error_on**

```c
hsiz_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:

```c
status = HE5_EHset_error_on(1, 0);
```

To suppress both HDF5 and HDF-EOS5 error messages:

```c
status = HE5_EHset_error_on(2, 0);
```

**FORTRAN**

```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:

```fortran
status = he5_ehset_error_onf(1, 0)
status = he5_ehset_error_onf(2, 0)
```
Write/Update Global File Attribute

**HE5_EHwriteglbattr**

```c
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:

```c
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:

```c
attr_val = 3.14159;
status = HE5_EHwriteglbattr(fileid, "FloatAttr",
H5T_NATIVE_FLOAT, count, &attr_val);
```

**FORTRAN**

```fortran
INTEGER FUNCTION he5_ehwrglatt(fid,attrname,ntype,count,buffer)
```

---

2-289 EED2-175-002
integer $fid, status, ntype$

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:

$\text{count} = 1$

$status = \text{he5_ehwrglatt}(fid, \text{"FloatAttr"}, HE5T_NATIVE_FLOAT, count, buffer).$

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.

Note: The hsize_t typedef uses the largest type of integer available on a machine (typically a 64-bit integer). So when compiling a FORTRAN code in a 64-bit structure one must declare integers as integer*8 (rather than integer*4) for integers whose C equivalent is declared as hsize_t, since underlying C code expects “long” type integer. For 32-bit compilation on a 64-bit machine “integer*4” should work fine.
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*:

```c
zaID = HE5_ZAattach(fid, "ExampleZA");
```

The za can then be referenced by subsequent routines using the handle, *zaID*.

**FORTRAN**

```fortran
integer function he5_zaatattach(fid,zaname)
integer fid
character(*) zaname
The equivalent FORTRAN code for the example above is:
```

```fortran
zaid = he5_zaatattach(fid, "ExampleZA")
```
Return Information about a ZA Attribute

HE5_ZAattrinfo, HE5_ZAattrinfo2

herr_t HE5_ZAattrinfo(hid_t zaID, const char *attrname, hid_t *ntype, hsize_t *count)

herr_t HE5_ZAattrinfo2(hid_t zaID, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

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
size OUT: Buffer size of attribute element

Purpose Returns information about an object attribute in a specific za object. 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_ZAattrinfo(zaID, "ScalarFloat", &nt, &count);

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

FORTRAN integer function he5_zaatrinfoo(zaid, attrname, ntype, count)
integer zaid
character(*) attrname
integer ntype
integer*4 count

The equivalent FORTRAN code for the example above is:

status = he5_zaatrinfoo(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**

  ```c
  status = HE5_ZAclose(fid);
  ```

- **FORTRAN**  
  integer function he5_zaclose(fid)

  integer fid

  The equivalent FORTRAN code for the example above is:

  ```fortran
  status = he5_zaclose(fid)
  ```
Retrieve Compression Information for Field

HE5_ZAcompinfo

herr_t HE5_ZAcompinfo(hid_t zaID, const char *fieldname, int *comppcode, int comppparm[])

zaID IN: ZA ID returned by HE5_ZAcreate or HE5_ZAattach
fieldname IN: Fieldname
comppcode OUT: HDF compression code
comppparm 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, comppparm);

The compcode parameter will be set to 4 and comppparm[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 comppparm(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.

```c
zaID = HE5_ZAcreate(fid, "ExampleZA");
```

The za structure is referenced by subsequent routines using the handle, `zaID`.

**FORTRAN**

```fortran
INTEGER FUNCTION he5_zacreate(fid,zaname)

INTEGER  fid
CHARACTER(*) zaname

The equivalent FORTRAN code for the example above is:

```c
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**

```c
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

```c
ndims       = 2
compcode    = 4;
compparm[0] = 6;
dim[0]      = 100;
dim[1]      = 200;
status = HE5_ZAdefcomchunk(zaID, compcode, compparm, ndims, dim);
```

**FORTRAN**

```fortran
integer function he5_zadefcomch(zaid,comppcode, 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**

```c
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:** Shuffling, szip, and deflate compression are supported in this release. See HE5_GDdefcomp for more info.

**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 default deflate compression, the **Opacity** and **Spectra** fields using deflate compression with compression degree of 5, and use no compression for the **Temperature** field.

```c
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);
```

2-301  EED2-175-002
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);
compparm(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**

```c
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:

```c
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**:

```c
status = HE5_ZAdefdim(zaID, "Unlim", H5S_UNLIMITED);
```

**FORTRAN**

```fortran
integer function he5_zadefdim(zaid,dimname,dim)
integer       zaid
character(*)  dimname
integer*4     dim
```

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

```fortran
dim = 4000
```
status = he5_zadefdim(zaid, "DataTrack", dim)

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

```fortran
parameter (HE5S_UNLIMITED_F=-1)

status = he5_zadefdim(zaid, "Unlim", HE5S_UNLIMITED_F)
```
Define a New Data Field within a ZA

**HE5_ZAdefine**

```c
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. The user needs to define chunking and compression before every field definitions.

**Example**

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

```c
status = HE5_ZAdefine(zaID, "Spectra",
                        "Bands,DataTrack,DataXtrack", " ", H5T_NATIVE_FLOAT);
```

**FORTRAN**

```fortran
integer function he5_zadefine(zaid, za_name, dimlist, maxdimlist, dtype)
  integer zaid
  character(*) za_name
  character(*) dimlist
  character(*) maxdimlist
  integer dtype
end function he5_zadefine
```

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

```fortran
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);

FORTRAN integer function he5_zadetach(zaid)

    integer zaid

The equivalent FORTRAN code for the example above is:

status = he5_zadetach(zaid)
Retrieve Size of Specified Dimension

**HE5_ZAdiminfo**

```c
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":

```c
dimsize = HE5_ZAdiminfo(zaID, "DataTrack");
```

The return value, `dimsize`, will be equal to 4000.

**FORTRAN**

```fortran
integer*4 function he5_zadiminfo(zaid,dimname)
integer zaid
character(*) dimname
integer*4 dimsize
```

The equivalent FORTRAN code for the example above is:

```fortran
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);

FORTRAN integer function he5_zadropalias (zaid, fldgroup, aliasname)
integer _zaid
character*(*) _fldgroup
character*(*) _aliasname

The equivalent FORTRAN code for the first example above is:

aliasname = "temps 0 to 30"
status = he5_zadropalias(zaid, HE5_HDFE_DATAGROUP, aliasname)
Return Information about a ZA Dimension Scale Attribute

**HE5_ZAdscaleattrinfo, HE5_ZAdscaleattrinfo2**

herr_t HE5_ZAdscaleattrinfo(hid_t zaID, const char *dimname, const char *attrname, hid_t *ntype, hsize_t *count)

herr_t HE5_ZAdscaleattrinfo2(hid_t zaID, const char *dimname, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zaID</td>
<td>ZA ID returned by HE5_ZAcreate or HE5_ZAattach</td>
</tr>
<tr>
<td>dimname</td>
<td>Dimension scale name</td>
</tr>
<tr>
<td>attrname</td>
<td>Attribute name</td>
</tr>
<tr>
<td>ntype</td>
<td>OUT: Number type of attribute</td>
</tr>
<tr>
<td>count</td>
<td>OUT: Number of attribute elements</td>
</tr>
<tr>
<td>size</td>
<td>OUT: Buffer size of attribute element</td>
</tr>
</tbody>
</table>

**Purpose**
Returns information about attribute(s) in a specific dimension scale.

**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 dimension scale attribute.

**Example**
In this example, we return information about the **IntValues** attribute of **Bands** dimension scale.

```c
status = HE5_ZAdscaleattrinfo(zaID, “Bands”, “IntValues”, &ntype, &count);
```

The `ntype` variable will have the value 0 and `count` will have the value of 3.

**FORTRAN**

```fortran
integer function he5_zadscaleattrinfo(zaid, _fieldname, attrname, ntype, count)
    integer zaid
    character(*) attrname
    integer ntype
    integer *4 count
```
The equivalent *FORTRAN* code for the first example above is:

```fortran
status = he5_zadscaleattrinfo(zaid, "Bands", "IntValues", ntype, count)
```
Rename Zonal Average Data Field

**HE5_ZAfldrename**

```c
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*.

```c
strcpy(newfieldname, "temps 0 to 30");
status = HE5_ZAfldrename(zaID, "Temperature", newfieldname);
```

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

```fortran
integer function he5_zafldrename (zaid, oldfieldname, newfieldname)
   integer zaid
   character(*) oldfieldname
   character(*) newfieldname

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 Dimension Scale for a Dimension of a Field within a Zonal Average Structure

HE5_ZAgetdimscale

long HE5_ZAgetdimscale(hid_t zaID, char *fieldname, char *dimname,
                        hsize_t *dimsize, hid_t *numbertype, void *data)

zaID IN: ZA ID returned by HE5_ZAcreate or HE5_ZAattach
fieldname IN: Name of the field whose dimname dimension scale is read
dimname IN: The dimension for which scale values are read
dimsize OUT: The size of the dimension to be read
numbertype OUT: The number type of the data stored in the scale. See Appendix A for number types.
data OUT: Values to be read for the dimension scale

Purpose Gets dimension scale for a field dimension within the ZA.

Return value Returns data buffer size if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list or non-existing field.

Description This routine gets dimension scale for a field dimension within the ZA. The dimension scales attributes label, unit, format and others can be read using HE5_ZAreadscaleattr().

Example In this example, we get dimension scale for the Bands dimension in the Spectra field, defined using HE5_ZAsetdimscale() or HE5_ZAdefdimscale():

long buffsize;

hsize_t nbands;

hid_t ntype;

int *bands;

/* First call, with NULL for data buffer, returns */
/* buffsize needed for the data buffer */
buffsize = HE5_ZAgetdimscale(zaID, "Spectra", "Bands",
                             &nbands, &ntype, NULL);

/* allocate enough buffer for the data */
bands = (int *)malloc(buffsize);

buffsize = HE5_ZAgetdimscale(zaID, "Spectra", "Bands",
                                 &nbands, &ntype,(void *)bands);

FORTRAN

integer function he5_zagetdimscale(zaid, fieldname, dimname, dimsize,
                                    numbertype, data)

integer*4      zaid
character*(*)  fieldname
character*(*)  dimname
integer*4      dimsize
integer*4      numbertype
<valid type>   data(*)

The equivalent FORTRAN code for the example above is:

integer*4      bands(5)

integer*4      nbands, ntype, buffsize

buffsize = he5_zagetdimscale(zaid, "Spectra", "Bands",
                                 nbands, ntype, bands);
Get External Data File Information

**HE5_ZAgetexdata**

```c
int HE5_ZAgetexdata(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:

```c
nfiles = HE5_ZAgetexdata(zaID, "ExtData", namlen, filenames, offset, size);
```

**FORTRAN**
integer function he5_zagetxdat(zaid,fieldname,nlen, flist,offset, size)

```fortran
integer zaid
integer nfiles
integer*4 nlen
integer*4 offset(*)
integer*4 size(*)
character(*) filename
character(*) flist
```

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

```fortran
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, HE5_ZAgrpattrinfo2**

herr_t HE5_ZAgrpattrinfo(hid_t zaID, const char *attrname, hid_t *ntype, hsize_t *count)

herr_t HE5_ZAgrpattrinfo2(hid_t zaID, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

- **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
- **size** OUT: Buffer size of attribute element

**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.

```c
status = HE5_ZAgrpattrinfo(zaID, "ScalarFloat", &nt, &count);
```

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

**FORTRAN**

```fortran
integer function he5_zagattrinfo(zaid, attrname, ntype, count,)
integer zaid
character(*) attrname
integer ntype
integer *4 count
The equivalent FORTRAN code for the first example above is:

```
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:

```c
status = HE5_ZAinfo(zaID, "Spectra", &rank, dims, dtype, dimlist, maxdimlist);
```

The return parameters will have the following values:

```c
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.
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 object attributes defined in a specific ZA object. 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`:

```c
nattr = HE5_ZAinqattrs(zaID, NULL, &strbufsize);
```

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

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

```c
nattr = HE5_ZAinqattrs(zaID, attrnames, &strbufsize);
```

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

"attrOne,attr_2".

**FORTRAN**

```fortran
integer*4 function he5_zainqattrs(zaid,attrnames,strbufsize)
integer zaid
character(*) attrnames
integer*4 strbufsize
```

The equivalent FORTRAN code for the example above is:

```fortran
nattr = he5_zainqattrs(zaid, attrnames, strbufsize)
```
Return Data Type Information about Data Fields in Zonal Average

HE5_ZAinqdatatype

```c
herr_t HE5_ZAinqdatatype(hid_t zaID, const char *fieldname, const char *attrname, int group, hid_t *datatype, H5T_class_t *classid, H5T_order_t *order, size_t *size)
```

**zaid**
IN: ZA ID returned by HE5_ZAcreate or HE5_ZAattach

**fieldname**
IN: Field name

**attrname**
IN: Attribute name

**group**
IN: 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 zonal average.

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

**Description**
This routine returns information about field data in a zonal average.

**Example**
In this example we return the data type information for the Spectra field in the za defined in the HE5_ZAdefine routine.

```c
status = HE5_ZAinqdatatype(zaID, "Spectra", NULL, group, &datatype, &classid, &order, &size);
```

**FORTRAN**
integer function he5_zaidtype(zaid,fieldname,attrname,grp,dtype,classid,order, size)

integer zaid

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_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:

```c
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)

```fortran
integer zaid
character(*) dimlist
integer*4 dims(*)
```

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

```fortran
ndims = he5_zainqdims(zaid, dimlist, dims)
```
Retrieve Information for ZA Dimension Scale Attributes

**HE5_ZAinqdscaleattrs**

```c
long HE5_ZAinqdscaleattrs(hid_t zaID, const char *dimname, char *attrnames,
long *strbufsize)
```

- **zaID**  
  IN: ZA ID returned by HE5_ZAcreate or HE5_ZAattach

- **dimname**  
  IN: Dimension scale name to retrieve attribute information

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

- **strbufsize**  
  OUT: String length of attribute list

**Purpose** Retrieve information about the attributes defined for a specific dimension scale.

**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 dimension scale attributes defined for a field “Bands”. In the first call, set the parameter `attrnames` to NULL. We assume that there are five attributes stored, label, unit, format, MissingValue, and IntValues:

```c
nattr = HE5_ZAinqlocattrs(zaID, "Bands", NULL, &strbufsize);
```

The parameter, `nattr`, will have the value 5 and `strbufsize` will have value 40.

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

```c
nattr = HE5_ZAinqlocattrs(zaID, "Bands", attrnames, &strbufsize);
```

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

"label,unit,format,MissingValue,IntValues ".

**FORTRAN**

```fortran
integer*4 function he5_zainqdscaleattrs(zaid , dimname, attrnames, strbufsize)
```

```fortran
integer zaid
character*(*) dimname
```
character*(*)  attrnames
integer*4    strbufsize

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

```fortran
nattr = he5_zainqlattrs(zaid, "Bands", attrnames, strbufsize)
```
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".

FORTRAN integer*4 function he5_zainqgattrs(zaid,attrnames, strbufsize)

integer zaid
character(*) attrnames
integer*4 strbufsize
integer*4 nattr

The equivalent FORTRAN code for the example above is:

nattr = he5_zainqgattrs(zaid, attrnames, strbufsize)
Retrieve Information Zonal Average Local Attributes

HE5_ZAinqlocattrs

long HE5_ZAinqlocattrs(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_ZAinqlocattrs(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_zainqlocattrs(zaID, “DataField”, attrnames, &strbufsize);

The variable, attrlist, will be set to:

"attrOne,attr_2".

FORTRAN integer*4 function he5_zainqlattrs(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:

```c
nflds = HE5_ZAinquire(zaID, za_name_list, rank, dtype);
```

The parameter, za_name_list, will have the value:


FORTRAN integer*4 function he5_zainquire(zaid, za_name_list, rank, dtype)

```fortran
integer zaid
character(*) za_name_list
integer rank(*)
integer dtype(*)
```

The equivalent FORTRAN code for the example above is:

```fortran
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”.

FORTRAN integer*4 function he5_zainqza(filename,zalist,strbufsize)
character*(*) filename
character*(*) zalist
integer*4 strbufsize

The equivalent FORTRAN code for the example above is:
nza = he5_zainqza('za.he5', zalist, strbufsize)
Return Information about a Local Zonal Average Attribute

HE5_ZAlocattrinfo, HE5_ZAlocattrinfo2

herr_t HE5_ZAlocattrinfo(hid_t zaID, const char *fieldname, const char *attrname, hid_t *ntype, hsize_t *count)
herr_t HE5_ZAlocattrinfo2(hid_t zaID, const char *fieldname, const char *attrname, hid_t *ntype, hsize_t *count, hsize_t *size)

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
size OUT: Buffer size of attribute element

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.

FORTRAN integer function he5_zalattrinfo(zaid, *fieldname, attrname, ntype, count)
integer zaid
character(*) attrname
integer ntype
integer *4 count

The equivalent FORTRAN 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

```c
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**

\[
\text{long HE5_ZAnentries(hid_t zaID, int entrycode, long *strbufsize)}
\]

- **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.

\[
\text{ndflds} = \text{HE5_ZAnentries(zaID, HE5_HDFE_NENTDFLD, &bufsize)};
\]

The return value, \( \text{ndflds} \), will be equal to 4 and \( \text{bufsz} = 39 \)

**FORTRAN**

\[
\text{integer*4 function he5_zanentries(zaid, entrycode, bufsize)}
\]

\[
\begin{align*}
\text{integer} & \quad \text{zaid} \\
\text{integer} & \quad \text{entrycode} \\
\text{integer*4} & \quad \text{bufsize}
\end{align*}
\]

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

\[
\text{parameter (HE5_HDFE_NENTDFLD=4)}
\]

\[
\text{ndflds} = \text{he5_zanentries(zaid, HE5_HDFE_NENTDFLD, bufsize)}
\]
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`.

```c
fid = HE5_ZAopen("ZA.he5", H5F_ACC_TRUNC);
```

**FORTRAN**

integer function he5_zaopen(filename, access)

character(*) filename

integer access

The access codes should be defined as parameters:

```fortran
parameter (HE5F_ACC_RDONLY = 101)
parameter (HE5F_ACC_RDWR = 100)
parameter (HE5F_ACC_TRUNC = 102)
```

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

```fortran
fid = he5_zaopen("za.he5", HE5F_ACC_TRUNC)
```

**Note to users of the SDP Toolkit:** Please refer to the *SDP Toolkit User Guide for the EOSDIS Evolution and Development Project (333-EED2-001, Revision 01)*, 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**

```c
herr_t HE5_ZAread(hid_t zaID, char *za_name, const hsize_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 read
- **start** IN: Array specifying the starting location within each dimension
- **stride** IN: Array specifying the number of values to skip along each dimension
- **count** IN: Array specifying the number of values to read along each dimension
- **datbuf** 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 field name.

**Description**
The values within `start`, `stride`, and `count` arrays refer to the zonal average field (input) dimensions. The output data in `datbuf` 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 `count` 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 use `HE5_ZAreadlocattr` to get the value of maximum string length in the local attribute `StringLengthAttribute`.

**Example**
In this example, we read data from the `Spectra` field.

```c
float     plane[15][40][20];
hssize_t  start[3] = {0, 0, 0};
hszie_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 object attribute from a specific ZA object. 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);

FORTRAN integer function he5_zardattr(zaid,attrname,datbuf)

integer zaid
<valid type> datbuf(*)

The equivalent FORTRAN code for the example above is:

status = he5_zardattr(zaid, "ScalarFloat", datbuf)
Read Attribute for a Dimension scale within a Zonal Average Structure

HE5_ZAreaddscaleattr

herr_t HE5_ZAreaddscaleattr(hid_t zaID, const char *dimname,
                          const char *attrname, void *datbuf)

zaID IN: ZA ID returned by HE5_ZAcreate or HE5_ZAattach
dimname IN: Dimension scale name for which attribute is written
attrname IN: Attribute name
datbuf OUT: Buffer allocated to hold attribute values

Purpose Reads a dimension scale attribute from a specific dimension.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper za id 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 attributes of the Bands dimension scale:

```c
herr_t          status = FAIL;
hid_t           ZAid1 = FAIL;
int             i;
long            nattr;
long            strbufsize;
char           *attrlist;
size_t          fldnmlen[HE5_HDFE_NAMBUFSIZE];
char           *fldnm[HE5_HDFE_NAMBUFSIZE];
char           *attrname = (char *)NULL;
hid_t          *ntype;
hsize_t         count = 0;
void           *attr;
int             *attr_int;
float           *attr_flt;
float           *attr_dbl;
char           *attr_char;

nattr = HE5_ZAinqdscaleattrs(ZAid1, "Bands", NULL,
          &strbufsize);
attrlist = (char *) calloc(strbufsize + 2, sizeof(char));
```
nattr = HE5_ZAinqdscaleattrs(ZAid1, "Bands", attrlist, &strbufsize);
 nattr = HE5_EHparsestr(attrlist, ',', fldnm, fldnmlen);
for( i = 0; i < nattr; i++ )
{
    attrname = (char *)calloc(fldnmlen[i] + 1, sizeof(char));
    memmove(attrname, fldnm[i], fldnmlen[i]);
    ntype = (hid_t *)calloc(1, sizeof(hid_t));
    if(strcmp(attrname, "REFERENCE_LIST") == 0 )
    {
        continue;
    }
    status = HE5_ZAdscaleattrinfo(ZAid1,"Bands", attrname, ntype, &count);
    if( (int)*ntype == 0 ) {
        attr_int = (int *)malloc(count * sizeof(int));
        attr = (void *) attr_int;
    }
    if( (int)*ntype == 10 ) {
        attr_flt = (float *)malloc(count * sizeof(float));
        attr = (void *) attr_flt;
    }
    if( (int)*ntype == 11 ) {
        attr_dbl = (double *)malloc(count * sizeof(double));
        attr = (void *) attr_dbl;
    }
    if( (int)*ntype == 57 ) {
        attr_char = (char *)malloc((count+1) * sizeof(char));
        attr = (void *) attr_char;
    }
    status = HE5_ZAreaddscaleattr(ZAid1,"Bands",attrname,
        attr);
}

FORTRAN

integer function he5_zareaddscaleattr (zaid, dimname, attrname, datbuf)

integer*4      zaid
character(*)    dimname
character(*)    attrname
<valid type>    datbuf(*)

The equivalent FORTRAN code for the example above is:

integer         j, ntype
integer         zaid1
integer         attr_int(25)
real*4           attr_flt(25)
real*8          attr_dbl(25)
character       attr_char(25)
integer         nattr
character*100   attrlist
character*100   strbufsize
character*15    attrname(10)
nattr = HE5_ZAinqdscaleattrs(ZAid1, "Bands", attrlist, strbufsize)
attrname(1) = 'label'
attrname(2) = 'unit'
attrname(3) = 'format'
attrname(4) = 'MissingValue'
attrname(5) = 'IntValues'
do j = 1,5
   attr_char = ''
   count(1)= 0
   count(2)= 0
   status = HE5_ZAdscaleattrinfo(ZAid1,"Bands",
      attrname(j), ntype, count)
   if( ntype .eq. 0) then
      status = HE5_ZAreaddscaleattr(ZAid1,"Bands",
         attrname(j), attr_int)
   endif
   if( ntype .eq. 10) then
      status = HE5_ZAreaddscaleattr(ZAid1,"Bands",
         attrname(j), attr_flt)
   endif
   if( ntype .eq. 11) then
      status = HE5_ZAreaddscaleattr(ZAid1,"Bands",
         attrname(j), attr_dbl)
   endif
   if( ntype .eq. 57) then
      status = HE5_ZAreaddscaleattr(ZAid1,"Bands",
         attrname(j), attr_char)
   endif
enddo
Read External Data Set

HE5_ZAreadexternal

herr_t HE5_ZAreadexternal(hid_t zaID, int fldgroup, const char *fieldname, void *buffer)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zaID</td>
<td>IN: ZA ID returned by HE5_ZAcreate or HE5_ZAattach</td>
</tr>
<tr>
<td>fldgroup</td>
<td>IN: Field group flag</td>
</tr>
<tr>
<td>fieldname</td>
<td>IN: Name of field to read</td>
</tr>
<tr>
<td>buffer</td>
<td>OUT: Output data buffer</td>
</tr>
</tbody>
</table>

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:

```c
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);

FORTRAN integer function he5_zardgattr(zaid,attrname,datbuf)

integer zaid
character(*) attrname
<valid type> datbuf(*)

The equivalent FORTRAN code for the example above is:

status = he5_zardgattr(zaid, "ScalarFloat", datbuf)
Read Local Zonal Average Attribute

**HE5_ZAreadlocattr**

```c
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":

```c
status = HE5_ZAreadlocattr(zaID, "DataField", "ScalarFloat", &data);
```

**FORTRAN**

```fortran
integer function he5_zardlattr(zaid, fieldname, attrname, datbuf)
  integer       zaid
  character(*)  fieldname
  character(*)  attrname
  <valid type>  datbuf(*)
```

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

```fortran
status = he5_zardlattr(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 and 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 Dimension Scale for a Dimension of a Field or Fields within a Zonal Average Structure

HE5_ZAsetdimscale

herr_t HE5_ZAsetdimscale(hid_t zaID, char *fieldname, char *dimname,
const hsize_t dimsize, hid_t numbertype, void *data)

HE5_ZAsetdimscale

herr_t HE5_ZAsetdimscale(hid_t zaID, char *dimname,
const hsize_t dimsize, hid_t numbertype, void *data)

zaID IN: ZA ID returned by HE5_ZAcreate or HE5_ZAattach
fieldname IN: Name of the field whose dimname dimension scale is set
dimname IN: The dimension for which scale is set in the field
dimsizel IN: The size of the dimension for which dimension is set
numbertype IN: The number type of the data stored in the scale. See Appendix A for number types.
data IN: Values to be written to the dimension scale

Purpose HE5_ZAsetdimscale sets dimension scale for a field dimension within the za. HE5_ZAsetdimscale sets dimension scale for a dimension of all field within the za.
Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list, none-existing field, or having the same dimension set before.
Description These routines set dimension scale for a field (or fields) dimension within the za. Once the dimension scales is set user can write label, unit, format, and other attributes to it using HE5_ZAwritedscaleattr().
Example 1 In this example, we set dimension scale for the “Bands” dimension in the Spectra field, defined by:

```c
status = HE5_ZAdefdatafield( zaID, "Spectra",
"Bands,DataTrack,DataXtrack", " ", H5T_NATIVE_FLOAT);
int bands[5] = {1,3,6,7,8};
hsize_t nbands = 5;
status = HE5_ZAsetdimscale(zaID, "Spectra", "Bands",
nbands, H5T_NATIVE_INT, bands);
```
The equivalent FORTRAN code for the example above is:

```fortran
integer*4     bands(5)
integer*4     nbands
nbands = 5
bands(1) = 1
bands(2) = 3
bands(3) = 6
bands(4) = 7
bands(5) = 8
status = he5_zasetdimscale(zaid, "Spectra", "Bands",
                           nbands, HE5T_NATIVE_INT, bands);
```

Example 2
In this example, we set dimension scale for the “Bands” dimension in all field, defined by HE5_ZAdefdatafield() in the ZA:

```c
int bands[5] = {1,3,6,7,8};
hsize_t nbands = 5;
status = HE5_ZAdefdimscale(zaID, "Bands",
                           nbands, H5T_NATIVE_INT, bands);
```

The equivalent FORTRAN code for the example above is:

```fortran
integer*4     bands(5)
integer*4     nbands
nbands = 5
bands(1) = 1
bands(2) = 3
bands(3) = 6
bands(4) = 7
bands(5) = 8
status = he5_zadefdimscale(zaid, "Bands",
                           nbands, HE5T_NATIVE_INT, bands);
```
**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:

```c
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(*) flist

    status = he5_zasetxdat(zaid,fllist,offset, size)
```

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

```c
status = he5_zasetxdat(zaid,fllist,offset, size)
```
Set Fill Value for a Specified Field

**HE5_ZAsetfillvalue**

```c
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:

```c
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:

```fortran
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**

```c
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 \((\text{dim} - \text{start}) / \text{stride}\) where \(\text{dim}\) 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.

```c
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

```fortran
he5_zawrite(zaID,za_name,start,stride,count,datbuf)
```
The `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:

```fortran
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)
```

Note: When writing data to a field with an unlimited dimension you must not write more data than the actual dimension of the field in first call to ZAwrite, otherwise only partial data will be written to the field. You should do this 1 or 2 more calls to ZAwrite. In the first attempt you write less data than or equal to the actual dimension of the field. In the following attempts you can have anything for start and count (count > start), even start of second attempt can be larger than the count of the first attempt. Please note that in the second (and the following attempts) data buffer is written to the file starting from its 0th element.
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 object attribute in a specific ZA object. 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:

```c
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:

```c
attr_val = 3.14159;
status = HE5_ZAwriteattr(zaid, "ScalarFloat",
H5T_NATIVE_FLOAT, 1, &attr_val);
```
FORTRAN  

integer function he5_zawrattr(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_zawrattr(zaid, "ScalarFloat",
                      HE5T_NATIVE_FLOAT, count, datbuf)
Write Field Metadata for an Existing Zonal Average Data Field

**HE5_ZAwritedatameta**

```
he5_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.

```c
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:

```fortran
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 Attribute for a Dimension scale within a Zonal Average Structure

HE5_ZAWriteScaleAttr

herr_t HE5_ZAWriteScaleAttr(hid_t zaID, const char *dimname,
                            const char *attrname, hid_t ntype, hsize_t count[], void *datbuf)

zaID IN: ZA ID returned by HE5_ZAcreate or HE5_ZAAttach

dimname IN: Dimension scale name for which attribute is written

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 a dimension scale attribute in a specific ZA.

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 attributes label, unit, format, MissingValues, and IntValues for the Bands dimension scale:

```c
strcpy(label, "Bands Dim");
strcpy(unit, "None");
strcpy(format, "I2");
count[0]= 12;
status = HE5_ZAWriteScaleAttr(ZAid1, "Bands",
                              "label", H5T_NATIVE_CHAR, count, label);
count[0]= 6;
status = HE5_ZAWriteScaleAttr(ZAid1, "Bands",
                              "unit", H5T_NATIVE_CHAR, count, unit);
count[0]= 4;
```
status = HE5_ZA writedscaleattr (ZAid1, "Bands",
   "format", H5T_NATIVE_CHAR, count, format);

int datbuf_i1[1] = {-999};
count[0]= 1;
status = HE5_ZA writedscaleattr (ZAid1, "Bands",
   "MissingValue", H5T_NATIVE_INT, count,
   datbuf_i1);

int datbuf_i2[3] = {-999,0,999};
count[0]= 3;
status = HE5_ZA writedscaleattr (ZAid1, "Bands",
   "IntValues", H5T_NATIVE_INT, count,
   datbuf_i2);

FORTRAN integer function he5_zawritedscaleattr (zaid, dimname, attrname, ntype, count, datbuf)

integer*4 zaid
character*(*) dimname
character*(*) attrname
integer*4 ntype
integer*4 count(*)
<valid type> datbuf(*)

The equivalent FORTRAN code for the example above is:

integer zaid1
integer*4 datbuf_i1[1]
integer*4 datbuf_i2[2]
integer count[2]
count(1)= 12
status = HE5_ZA writedscaleattr (ZAid1, "Bands",
   "label", H5T_NATIVE_CHAR, count, "Bands Dim")
count(1)= 6
status = HE5_ZA writedscaleattr (ZAid1, "Bands",
   "unit", H5T_NATIVE_CHAR, count, "None")
count(1) = 4
status = HE5_ZAwritedscaleattr(ZAid1, "Bands",
    "format", HE5T_NATIVE_CHAR, count, "I2")
datbuf_i1(1) = -999

count(1) = 1
status = HE5_ZAwritedscaleattr(ZAid1, "Bands",
    "MissingValue", HE5T_NATIVE_INT, count, datbuf_i1)
datbuf_i(1) = -999
datbuf_i(2) = 0
datbuf_i(3) = 999

count(1) = 3
status = HE5_ZAwritedscaleattr(ZAid1, "Bands",
    "IntValues", HE5T_NATIVE_INT, count, datbuf_i)
Write/Update Group Zonal Average Attribute

**HE5_ZAwritegrpattr**

```c
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:

```c
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:

```c
attr_val = 3.14159;
status = HE5_ZAwritegrpattr(zaid, "ScalarFloat",
H5T_NATIVE_FLOAT, count, &attr_val);
```

**FORTRAN**

```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_ZAwritelocattr**

```c
herr_t HE5_ZAwritelocattr(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:

```c
count[0] = 1;
attr_val = 3.14;
status = HE5_ZAwritelocattr(zaid, "DataField", "ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

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

```c
attr_val = 3.14159;
status = HE5_ZAwritelocattr(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)
```

- **projcode** IN: GCTP projection code
- **zonecode** IN: GCTP zone code used by UTM projection
- **projparm** IN: Projection parameters
- **spherecode** IN: GCTP spherecode
- **xdimsize** IN: xdimsize from HE5_GDgridinfo()
- **ydimsize** IN: ydimsize from HE5_GDgridinfo()
- **upleft** IN: Upper left corner of the grid in meter (all projections except Geographic) or DMS degree (Geographic projection), values from HE5_GDgridinfo()
- **lowright** IN: Lower right corner of the grid in meter or DMS degree (Geographic projection), values from HE5_GDgridinfo()
- **npnts** IN: number of lon-lat points
- **row** IN: row numbers of the pixels (zero based)
- **col** IN: column numbers of the pixels (zero based)
- **pixcen** IN: Code from HE5_GDpixreginfo
- **pixcnr** IN: Code from HE5_GDorigininfo
- **longitude** OUT: longitude array (decimal degrees)
- **latitude** 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

```c
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;
```

```c
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);
```

For the example above, the equivalent FORTRAN code is:

```fortran
integer function he5_gdij2ll( projcode, zonecode, projparm, spherecode, xdimsize, ydimsize,upleft, lowright, npnts, rows, cols, longitude, latitude, pixregcode, origincode)
```

```fortran
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:

```fortran
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_gdpreginfo(gridid, pixregcode)
status = he5_gdorginfo(gridid, origincode)
status = he5_gdij2ll(projcode, zonecode, projparm, spherecode, xdimsize,
&
ydimsize, upleft, lowright, npnts, rows, cols, longitude, latitude,
&
pixregcode, origincode)

Note: If the pixel (i,j) is at the poles then this routine will return 90 (north pole) or
-90 (south pole) for the latitude. However depending on the floating point
accuracy one may get different results for longitude of this pixel from gctp.
The returned value for longitude could be any number between -180 and
+180.
Convert Grid Coordinates (Longitude, Latitude) to (i,j)

HE5_GDl2ij

intn HE5_GDl2ij(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[])

projcode     IN:  GCTP projection code
zonecode     IN:  GCTP zone code used by UTM projection
projparm     IN:  Projection parameters
spherecode   IN:  GCTP spherecode
xdimsize     IN:  xdimsize from HE5_GDgridinfo( )
ydimsize     IN:  ydimsize from HE5_GDgridinfo( )
upleft       IN:  Upper left corner of the grid in meter (all projections except Geographic) or DMS degree (Geographic projection), values from HE5_GDgridinfo( )
lowright     IN:  Lower right corner of the grid in meter or DMS degree, Geographic) or DMS degree (Geographic projection), values from HE5_GDgridinfo( )
npnts        IN:  number of lon-lat points
longitude     IN:  longitude array (decimal degrees)
latitude     IN:  latitude array (decimal degrees)
row          OUT:  row numbers of the pixels (zero based)
col          OUT:  column numbers of the pixels (zero based)
xval         OUT:  x array
yval         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

```fortran
! Example code

def gridid, npnts = 2;
def projcode, origincode, pixregcode, zonecode, spherecode;
def upleft[2], lowright[2];
def projparm[13];
def xcord[2], ycord[2];
def cols[2], rows[2], lon[2], lat[2];
def xdimsize, ydimsize;

l[0]= 48.0;
lon[0]= -120.0;
l[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_GDII2ij(projcode, zonecode, projparm, spherecode, xdimsize, ydimsize, upleft, lowright, npnts, lon, lat, , rows, cols, xcord, ycord);
```

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_gdpreginfo(gridid, pixregcode)
status = he5_gdorginfo(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 \((\text{longitude, latitude})\)

**HE5_GDrs2ll**

```c
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)
```

- **projcode** IN: GCTP projection code (HE5_GCTP_BCEA)
- **projparm** IN: Projection parameters array
- **xdimsize** IN: \(xdimsize\) from HE5_GDgridinfo()
- **ydimsize** IN: \(ydimsize\) from HE5_GDgridinfo()
- **upleft** IN: Upper left corner lon/lat of the grid in DMS format, value from HE5_GDgridinfo()
- **lowright** IN: Lower right corner lon/lat of the grid in DMS format, value from HE5_GDgridinfo()
- **npnts** IN: Number of lon-lat points
- **r** IN: Array of EASE grid’s \(r\) coordinate
- **s** IN: Array of EASE grid’s \(s\) coordinate
- **pixcen** IN: Code from HE5_GDpixreginfo()
- **pixcnr** IN: Code from HE5_GDorigininfo()
- **longitude** OUT: longitude array (decimal degrees)
- **latitude** 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**

```c
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_gpreginfo(gridid, pixregcode)
status = he5_gdorginfo(gridid, origincode)
status = he5_gdstr2ll(HE5_GCTP_BCEA, projparm, xdimsize, ydimsize, upleft, lowright, npnts, rcord, scord, longitude, latitude, pixregcode, origincode)
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 \texttt{HE5T\_arch\_base} where \texttt{arch} is an architecture name and \texttt{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.

| Table A1 |
|-----------------|-----------------|-----------------|
| \texttt{HE5T\_NATIVE\_INT} | 0 | \texttt{HE5T\_STD\_I8LE} | 29 |
| \texttt{HE5T\_NATIVE\_UINT} | 1 | \texttt{HE5T\_STD\_I16BE} | 30 |
| \texttt{HE5T\_NATIVE\_SHORT} | 2 | \texttt{HE5T\_STD\_I16LE} | 31 |
| \texttt{HE5T\_NATIVE\_USHORT} | 3 | \texttt{HE5T\_STD\_I32BE} | 32 |
| \texttt{HE5T\_NATIVE\_SCHAR} | 4 | \texttt{HE5T\_STD\_I32LE} | 33 |
| \texttt{HE5T\_NATIVE\_UCHAR} | 5 | \texttt{HE5T\_STD\_I64BE} | 34 |
| \texttt{HE5T\_NATIVE\_LONG} | 6 | \texttt{HE5T\_STD\_I64LE} | 35 |
| \texttt{HE5T\_NATIVE\_ULONG} | 7 | \texttt{HE5T\_STD\_U8BE} | 36 |
| \texttt{HE5T\_NATIVE\_LLONG} | 8 | \texttt{HE5T\_STD\_U8LE} | 37 |
| \texttt{HE5T\_NATIVE\_ULLONG} | 9 | \texttt{HE5T\_STD\_U16BE} | 38 |
| \texttt{HE5T\_NATIVE\_FLOAT} | 10 | \texttt{HE5T\_STD\_U16LE} | 39 |
| \texttt{HE5T\_NATIVE\_REAL} | 10 | \texttt{HE5T\_STD\_U32BE} | 40 |
| \texttt{HE5T\_NATIVE\_DOUBLE} | 11 | \texttt{HE5T\_STD\_U32LE} | 41 |
| \texttt{HE5T\_NATIVE\_LDOUBLE} | 12 | \texttt{HE5T\_STD\_U64BE} | 42 |
| \texttt{HE5T\_NATIVE\_INT8} | 13 | \texttt{HE5T\_STD\_U64LE} | 43 |
| \texttt{HE5T\_NATIVE\_UINT8} | 14 | \texttt{HE5T\_STD\_B8BE} | 44 |
| \texttt{HE5T\_NATIVE\_INT16} | 15 | \texttt{HE5T\_STD\_B8LE} | 45 |
| \texttt{HE5T\_NATIVE\_UINT16} | 16 | \texttt{HE5T\_STD\_B16BE} | 46 |
| \texttt{HE5T\_NATIVE\_INT32} | 17 | \texttt{HE5T\_STD\_B16LE} | 47 |
### Table A1 (continued)

<table>
<thead>
<tr>
<th>Architecture Name</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEST_NATIVE_UINT32</td>
<td>18</td>
<td>HEST_STD_B32BE 48</td>
</tr>
<tr>
<td>HEST_NATIVE_INT64</td>
<td>19</td>
<td>HEST_STD_B32LE 49</td>
</tr>
<tr>
<td>HEST_NATIVE_UINT64</td>
<td>20</td>
<td>HEST_STD_B64BE 50</td>
</tr>
<tr>
<td>HEST_NATIVE_B8</td>
<td>21</td>
<td>HEST_STD_B64LE 51</td>
</tr>
<tr>
<td>HEST_NATIVE_B16</td>
<td>22</td>
<td>HEST_IEEE_F32BE 52</td>
</tr>
<tr>
<td>HEST_NATIVE_B32</td>
<td>23</td>
<td>HEST_IEEE_F32LE 53</td>
</tr>
<tr>
<td>HEST_NATIVE_B64</td>
<td>24</td>
<td>HEST_IEEE_F64BE 54</td>
</tr>
<tr>
<td>HEST_NATIVE_HSIZE</td>
<td>25</td>
<td>HEST_IEEE_F64LE 55</td>
</tr>
<tr>
<td>HEST_NATIVE_HERR</td>
<td>26</td>
<td>HEST_NATIVE_CHAR 56</td>
</tr>
<tr>
<td>HEST_NATIVE_HBOOL</td>
<td>27</td>
<td>HEST_CHARSTRING 57</td>
</tr>
<tr>
<td>HEST_STD_I8BE</td>
<td>28</td>
<td></td>
</tr>
</tbody>
</table>

### Table A2

<table>
<thead>
<tr>
<th>Architecture Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE</td>
<td>IEEE-754 standard floating point types in various byte orders.</td>
</tr>
<tr>
<td>STD</td>
<td>This is an architecture that contains semi-standard datatypes like signed two's complement integers, unsigned integers, and bitfields in various byte orders.</td>
</tr>
<tr>
<td>NATIVE</td>
<td>This architecture contains C-like datatypes for the machine on which the library was compiled.</td>
</tr>
</tbody>
</table>
### Table A3

<table>
<thead>
<tr>
<th>B</th>
<th>Bitfield</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Floating point</td>
</tr>
<tr>
<td>I</td>
<td>Signed integer</td>
</tr>
<tr>
<td>S</td>
<td>Character string</td>
</tr>
<tr>
<td>U</td>
<td>Unsigned integer</td>
</tr>
</tbody>
</table>

### Table A4

<table>
<thead>
<tr>
<th>BE</th>
<th>Big endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>LE</td>
<td>Little endian</td>
</tr>
<tr>
<td>VX</td>
<td>Vax order</td>
</tr>
</tbody>
</table>

### Table A5

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HE5T_IEEE_F64LE</td>
<td>Eight-byte, little-endian, IEEE floating-point</td>
</tr>
<tr>
<td>HE5T_IEEE_F32BE</td>
<td>Four-byte, big-endian, IEEE floating point</td>
</tr>
<tr>
<td>HE5T_STD_I32LE</td>
<td>Four-byte, little-endian, signed two's complement integer</td>
</tr>
<tr>
<td>HE5T_STD_U16BE</td>
<td>Two-byte, big-endian, unsigned integer</td>
</tr>
<tr>
<td>HE5T_NATIVE_B64</td>
<td>Native Eight-byte bit field</td>
</tr>
</tbody>
</table>
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)
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESDIS</td>
<td>Earth Science Data and Information System (GSFC Code 505)</td>
</tr>
<tr>
<td>FDF</td>
<td>flight dynamics facility</td>
</tr>
<tr>
<td>FOV</td>
<td>field of view</td>
</tr>
<tr>
<td>ftp</td>
<td>file transfer protocol</td>
</tr>
<tr>
<td>GCT</td>
<td>geo–coordinate transformation</td>
</tr>
<tr>
<td>GCTP</td>
<td>general cartographic transformation package</td>
</tr>
<tr>
<td>GD</td>
<td>grid</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GSFC</td>
<td>Goddard Space Flight Center</td>
</tr>
<tr>
<td>HDF</td>
<td>hierarchical data format</td>
</tr>
<tr>
<td>HEG</td>
<td>HDF-EOS to GeoTIFF Conversion Tool</td>
</tr>
<tr>
<td>HITC</td>
<td>Hughes Information Technology Corporation</td>
</tr>
<tr>
<td>http</td>
<td>hypertext transport protocol</td>
</tr>
<tr>
<td>I&amp;T</td>
<td>integration &amp; test</td>
</tr>
<tr>
<td>ICD</td>
<td>interface control document</td>
</tr>
<tr>
<td>IDL</td>
<td>interactive data language</td>
</tr>
<tr>
<td>IP</td>
<td>Internet protocol</td>
</tr>
<tr>
<td>IWG</td>
<td>Investigator Working Group</td>
</tr>
<tr>
<td>JPL</td>
<td>Jet Propulsion Laboratory</td>
</tr>
<tr>
<td>LaRC</td>
<td>Langley Research Center</td>
</tr>
<tr>
<td>LIS</td>
<td>Lightening Imaging Sensor</td>
</tr>
<tr>
<td>M&amp;O</td>
<td>maintenance and operations</td>
</tr>
<tr>
<td>MCF</td>
<td>metadata configuration file</td>
</tr>
<tr>
<td>MET</td>
<td>metadata</td>
</tr>
<tr>
<td>MODIS</td>
<td>Moderate–Resolution Imaging Spectroradiometer</td>
</tr>
<tr>
<td>MSFC</td>
<td>Marshall Space Flight Center</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>NCSA</td>
<td>National Center for Supercomputer Applications</td>
</tr>
<tr>
<td>netCDF</td>
<td>network common data format</td>
</tr>
<tr>
<td>NGDC</td>
<td>National Geophysical Data Center</td>
</tr>
</tbody>
</table>
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
SMAP Soil Moisture Active Passive
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
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USNO</td>
<td>United States Naval Observatory</td>
</tr>
<tr>
<td>UT</td>
<td>universal time</td>
</tr>
<tr>
<td>UTC</td>
<td>Coordinated Universal Time</td>
</tr>
<tr>
<td>UTCF</td>
<td>universal time correlation factor</td>
</tr>
<tr>
<td>UTM</td>
<td>universal transverse mercator</td>
</tr>
<tr>
<td>VPF</td>
<td>vector product format</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>ZA</td>
<td>Zonal Average</td>
</tr>
</tbody>
</table>