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

White Paper

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

December 1999

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

Larry Klein /s/ 12/16/99
David Wynne, Alex Muslimov, Abe Taaheri, Date
Ray Milburn, Larry Klein
EOSDIS Core System Project

SUBMITTED BY

Robert Plante /s/ 12/16/99
Robert Plante, Manager, Science Office Date
EOSDIS Core System Project

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

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

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

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

Technical Points of Contact within EOS are:

Larry Klein, larry@eos.hitc.com
David Wynne, davidw@eos.hitc.com
An email address has been provided for user help:
pgstlkit@eos.hitec.com

Any questions should be addressed to:
Data Management Office
The ECS Project Office
Raytheon Systems Company
1616 McCormick Drive
Upper Marlboro, MD 20774-5301
Abstract

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

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

**Preface**

**Abstract**

**1. Introduction**

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

**2. Function Reference**

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

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

Abbreviations and Acronyms
1. Introduction

1.1 Purpose

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

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

1.2 Organization

This paper is organized as follows:

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

1.3 Swath Data

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

1.3.1 The Swath Data Interface

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

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

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

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>C FORTRAN</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>SWopen</td>
<td>swopen</td>
<td>Opens or creates HDF file in order to create, read, or write a swath</td>
<td>2-44</td>
</tr>
<tr>
<td></td>
<td>SWcreate</td>
<td>swcreate</td>
<td>Creates a swath within the file</td>
<td>2-6</td>
</tr>
<tr>
<td></td>
<td>Swattach</td>
<td>swattach</td>
<td>Attaches to an existing swath within the file</td>
<td>2-2</td>
</tr>
<tr>
<td></td>
<td>SWdetach</td>
<td>swdetach</td>
<td>Detaches from swath interface</td>
<td>2-24</td>
</tr>
<tr>
<td></td>
<td>SWclose</td>
<td>swclose</td>
<td>Closes file</td>
<td>2-4</td>
</tr>
<tr>
<td></td>
<td>SWdefdim</td>
<td>swdefdim</td>
<td>Defines a new dimension within the swath</td>
<td>2-13</td>
</tr>
<tr>
<td></td>
<td>SWdefdimmap</td>
<td>swdefdimmap</td>
<td>Defines the mapping between the geolocation and data dimensions</td>
<td>2-14</td>
</tr>
<tr>
<td></td>
<td>SWdefidxmap</td>
<td>swdefidxmap</td>
<td>Defines a non-regular mapping between the geolocation and data dimension</td>
<td>2-18</td>
</tr>
<tr>
<td></td>
<td>SWdefgeofield</td>
<td>swdefgeofield</td>
<td>Defines a new geolocation field within the swath</td>
<td>2-16</td>
</tr>
<tr>
<td></td>
<td>SWdefdatafield</td>
<td>swdefdatafield</td>
<td>Defines a new data field within the swath</td>
<td>2-11</td>
</tr>
<tr>
<td></td>
<td>SWdefcomp</td>
<td>swdefcomp</td>
<td>Defines a field compression scheme</td>
<td>2-9</td>
</tr>
<tr>
<td></td>
<td>SWwritefield</td>
<td>swwritefield</td>
<td>Writes data to a swath field</td>
<td>2-65</td>
</tr>
<tr>
<td></td>
<td>SWreadfield</td>
<td>swreadfield</td>
<td>Reads data from a swath field</td>
<td>2-54</td>
</tr>
<tr>
<td>Basic I/O</td>
<td>SWwriteattr</td>
<td>swwriteattr</td>
<td>Writes/updates attribute in a swath</td>
<td>2-63</td>
</tr>
<tr>
<td></td>
<td>SWreadattr</td>
<td>swreadattr</td>
<td>Reads attribute from a swath</td>
<td>2-53</td>
</tr>
<tr>
<td></td>
<td>SWsetfillvalue</td>
<td>swsetfillvalue</td>
<td>sets fill value for the specified field</td>
<td>2-60</td>
</tr>
<tr>
<td></td>
<td>SWgetfillvalue</td>
<td>swgetfillvalue</td>
<td>Retrieves fill value for the specified field</td>
<td>2-31</td>
</tr>
<tr>
<td></td>
<td>SWinqdims</td>
<td>swinqdims</td>
<td>Retrieves information about dimensions defined in swath</td>
<td>2-37</td>
</tr>
<tr>
<td></td>
<td>SWinqmaps</td>
<td>swinqmaps</td>
<td>Retrieves information about the geolocation relations defined</td>
<td>2-40</td>
</tr>
<tr>
<td></td>
<td>SWinqidxmaps</td>
<td>swinqidxmaps</td>
<td>Retrieves information about the indexed geolocation/data mappings defined</td>
<td>2-39</td>
</tr>
<tr>
<td></td>
<td>SWinqgeofields</td>
<td>swinqgeofields</td>
<td>Retrieves information about the geolocation fields defined</td>
<td>2-38</td>
</tr>
<tr>
<td></td>
<td>SWinqdatafields</td>
<td>swinqdatafields</td>
<td>Retrieves information about the data fields defined</td>
<td>2-35</td>
</tr>
<tr>
<td></td>
<td>SWinqdatatype</td>
<td>swinqdatatype</td>
<td>Retrieves information about data type of a field</td>
<td>2-36</td>
</tr>
<tr>
<td></td>
<td>SWinqatts</td>
<td>swinqatts</td>
<td>Retrieves number and names of attributes defined</td>
<td>2-34</td>
</tr>
<tr>
<td></td>
<td>SWnentries</td>
<td>swnentries</td>
<td>Returns number of entries and descriptive string buffer size for a specified entity</td>
<td>2-43</td>
</tr>
<tr>
<td>Inquiry</td>
<td>SWdiminfo</td>
<td>swdiminfo</td>
<td>Retrieve size of specified dimension</td>
<td>2-25</td>
</tr>
<tr>
<td></td>
<td>SWmapinfo</td>
<td>swmapinfo</td>
<td>Retrieve offset and increment of specified geolocation mapping</td>
<td>2-42</td>
</tr>
<tr>
<td></td>
<td>SWidxmapinfo</td>
<td>swidxmapinfo</td>
<td>Retrieve offset and increment of specified geolocation mapping</td>
<td>2-33</td>
</tr>
<tr>
<td></td>
<td>SWATtrinfo</td>
<td>SWATtrinfo</td>
<td>Returns information about swath attributes</td>
<td>2-3</td>
</tr>
<tr>
<td></td>
<td>SWfieldinfo</td>
<td>SWfieldinfo</td>
<td>Retrieves information about a specific geolocation or data field</td>
<td>2-29</td>
</tr>
<tr>
<td></td>
<td>SWcompinfo</td>
<td>SWcompinfo</td>
<td>Retrieve compression information about a field</td>
<td>2-5</td>
</tr>
<tr>
<td></td>
<td>SWings swath</td>
<td>SWings swath</td>
<td>Returns number and names of swaths in file</td>
<td>2-41</td>
</tr>
<tr>
<td></td>
<td>Swregionindex</td>
<td>Swregionindex</td>
<td>Returns information about the swath region ID</td>
<td>2-56</td>
</tr>
<tr>
<td>Ragged Arrays</td>
<td>SUpdateidxmap</td>
<td>SUpdateidxmap</td>
<td>Update map index for a specified region</td>
<td>2-61</td>
</tr>
<tr>
<td></td>
<td>SWraopen</td>
<td>SWraopen</td>
<td>Opens ragged array</td>
<td>2-49</td>
</tr>
<tr>
<td></td>
<td>SWrwrite</td>
<td>SWrwrite</td>
<td>Writes data to the ragged array</td>
<td>2-52</td>
</tr>
</tbody>
</table>
### Table 1-3. Summary of the Swath Interface (2 of 2)

<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>C</th>
<th>FORTRAN</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>FORTRAN</td>
<td></td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Subset</td>
<td>SWraread</td>
<td>swraread</td>
<td></td>
<td>Reads out data from the ragged array</td>
<td>2-50</td>
</tr>
<tr>
<td></td>
<td>SWraclose</td>
<td>swraclose</td>
<td></td>
<td>Closes the ragged array</td>
<td>2-47</td>
</tr>
<tr>
<td></td>
<td>SWgeomapinfo</td>
<td>swgmapinfo</td>
<td></td>
<td>Retrieves type of dimension mapping when first dimension is geodim</td>
<td>2-32</td>
</tr>
<tr>
<td></td>
<td>SWdefboxregion</td>
<td>swdefboxreg</td>
<td></td>
<td>Define region of interest by latitude/longitude</td>
<td>2-7</td>
</tr>
<tr>
<td></td>
<td>SWregioninfo</td>
<td>swregioninfo</td>
<td></td>
<td>Returns information about defined region</td>
<td>2-58</td>
</tr>
<tr>
<td></td>
<td>SWextractregion</td>
<td>swextractreg</td>
<td></td>
<td>Read a region of interest from a field</td>
<td>2-28</td>
</tr>
<tr>
<td></td>
<td>SWdftimeperiod</td>
<td>swdftimeper</td>
<td></td>
<td>Define a time period of interest</td>
<td>2-19</td>
</tr>
<tr>
<td></td>
<td>SWperiodinfo</td>
<td>swperiodinfo</td>
<td></td>
<td>Returns information about a defined time period</td>
<td>2-45</td>
</tr>
<tr>
<td></td>
<td>SWextractperiod</td>
<td>swextractper</td>
<td></td>
<td>Extract a defined time period</td>
<td>2-27</td>
</tr>
<tr>
<td></td>
<td>SWdefvrtregion</td>
<td>swdefvrtreg</td>
<td></td>
<td>Define a region of interest by vertical field</td>
<td>2-21</td>
</tr>
<tr>
<td></td>
<td>SWdupregion</td>
<td>swdupreg</td>
<td></td>
<td>Duplicate a region or time period</td>
<td>2-26</td>
</tr>
</tbody>
</table>

### 1.4 Grid Data

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

#### 1.4.1 The Grid Data Interface

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

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

#### 1.4.2 List of Grid API ROUTINES

The GD function calls are listed below in Table 1-4 and are described in detail in Section 2 of this document. The listing in Section 2 is in alphabetical order.
<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>GDopen</td>
<td>Creates a new file or opens an existing one</td>
<td>2-108</td>
</tr>
<tr>
<td></td>
<td>GDcreate</td>
<td>Creates a new grid in the file</td>
<td>2-72</td>
</tr>
<tr>
<td></td>
<td>GAttach</td>
<td>Attaches to a grid</td>
<td>2-68</td>
</tr>
<tr>
<td></td>
<td>GDdetach</td>
<td>Detaches from grid interface</td>
<td>2-92</td>
</tr>
<tr>
<td></td>
<td>GDclose</td>
<td>Closes file</td>
<td>2-70</td>
</tr>
<tr>
<td></td>
<td>GDdeforigin</td>
<td>Defines origin of grid</td>
<td>2-81</td>
</tr>
<tr>
<td></td>
<td>GDdefdim</td>
<td>Defines dimensions for a grid</td>
<td>2-78</td>
</tr>
<tr>
<td></td>
<td>GDdefproj</td>
<td>Defines projection of grid</td>
<td>2-83</td>
</tr>
<tr>
<td></td>
<td>GDdefpixreg</td>
<td>Defines pixel registration within grid cell</td>
<td>2-82</td>
</tr>
<tr>
<td></td>
<td>GDdeffield</td>
<td>Defines data fields to be stored in a grid</td>
<td>2-79</td>
</tr>
<tr>
<td></td>
<td>GDdefcomp</td>
<td>Defines a field compression scheme</td>
<td>2-76</td>
</tr>
<tr>
<td></td>
<td>GDwritefield</td>
<td>Writes data to a grid field.</td>
<td>2-121</td>
</tr>
<tr>
<td></td>
<td>GDreadfield</td>
<td>Reads data from a grid field</td>
<td>2-114</td>
</tr>
<tr>
<td></td>
<td>GDwriteattr</td>
<td>Writes/updates attribute in a grid.</td>
<td>2-119</td>
</tr>
<tr>
<td></td>
<td>GDreadattr</td>
<td>Reads attribute from a grid</td>
<td>2-113</td>
</tr>
<tr>
<td></td>
<td>GDsetfillvalue</td>
<td>sets fill value for the specified field</td>
<td>2-118</td>
</tr>
<tr>
<td></td>
<td>GDgetfillvalue</td>
<td>Retrieves fill value for the specified field</td>
<td>2-98</td>
</tr>
<tr>
<td></td>
<td>GDwritefield</td>
<td>Writes data to a grid field.</td>
<td>2-121</td>
</tr>
<tr>
<td></td>
<td>GDdeforig</td>
<td>Defines origin of grid</td>
<td>2-81</td>
</tr>
<tr>
<td></td>
<td>GDdefdim</td>
<td>Defines dimensions for a grid</td>
<td>2-78</td>
</tr>
<tr>
<td></td>
<td>GDdefproj</td>
<td>Defines projection of grid</td>
<td>2-83</td>
</tr>
<tr>
<td></td>
<td>GDdefpixreg</td>
<td>Defines pixel registration within grid cell</td>
<td>2-82</td>
</tr>
<tr>
<td></td>
<td>GDdeffield</td>
<td>Defines data fields to be stored in a grid</td>
<td>2-79</td>
</tr>
<tr>
<td></td>
<td>GDdefcomp</td>
<td>Defines a field compression scheme</td>
<td>2-76</td>
</tr>
<tr>
<td></td>
<td>GDwritefield</td>
<td>Writes data to a grid field.</td>
<td>2-121</td>
</tr>
<tr>
<td></td>
<td>GDreadfield</td>
<td>Reads data from a grid field</td>
<td>2-114</td>
</tr>
<tr>
<td></td>
<td>GDwriteattr</td>
<td>Writes/updates attribute in a grid.</td>
<td>2-119</td>
</tr>
<tr>
<td></td>
<td>GDreadattr</td>
<td>Reads attribute from a grid</td>
<td>2-113</td>
</tr>
<tr>
<td></td>
<td>GDsetfillvalue</td>
<td>sets fill value for the specified field</td>
<td>2-118</td>
</tr>
<tr>
<td></td>
<td>GDgetfillvalue</td>
<td>Retrieves fill value for the specified field</td>
<td>2-98</td>
</tr>
<tr>
<td></td>
<td>GDwritefield</td>
<td>Writes data to a grid field.</td>
<td>2-121</td>
</tr>
<tr>
<td></td>
<td>GDdeforig</td>
<td>Defines origin of grid</td>
<td>2-81</td>
</tr>
<tr>
<td></td>
<td>GDdefdim</td>
<td>Defines dimensions for a grid</td>
<td>2-78</td>
</tr>
<tr>
<td></td>
<td>GDdefproj</td>
<td>Defines projection of grid</td>
<td>2-83</td>
</tr>
<tr>
<td></td>
<td>GDdefpixreg</td>
<td>Defines pixel registration within grid cell</td>
<td>2-82</td>
</tr>
<tr>
<td></td>
<td>GDdeffield</td>
<td>Defines data fields to be stored in a grid</td>
<td>2-79</td>
</tr>
<tr>
<td></td>
<td>GDdefcomp</td>
<td>Defines a field compression scheme</td>
<td>2-76</td>
</tr>
<tr>
<td></td>
<td>GDwritefield</td>
<td>Writes data to a grid field.</td>
<td>2-121</td>
</tr>
<tr>
<td></td>
<td>GDreadfield</td>
<td>Reads data from a grid field</td>
<td>2-114</td>
</tr>
<tr>
<td></td>
<td>GDwriteattr</td>
<td>Writes/updates attribute in a grid.</td>
<td>2-119</td>
</tr>
<tr>
<td></td>
<td>GDreadattr</td>
<td>Reads attribute from a grid</td>
<td>2-113</td>
</tr>
<tr>
<td></td>
<td>GDsetfillvalue</td>
<td>sets fill value for the specified field</td>
<td>2-118</td>
</tr>
<tr>
<td></td>
<td>GDgetfillvalue</td>
<td>Retrieves fill value for the specified field</td>
<td>2-98</td>
</tr>
<tr>
<td></td>
<td>GDwritefield</td>
<td>Writes data to a grid field.</td>
<td>2-121</td>
</tr>
<tr>
<td></td>
<td>GDdeforig</td>
<td>Defines origin of grid</td>
<td>2-81</td>
</tr>
<tr>
<td></td>
<td>GDdefdim</td>
<td>Defines dimensions for a grid</td>
<td>2-78</td>
</tr>
<tr>
<td></td>
<td>GDdefproj</td>
<td>Defines projection of grid</td>
<td>2-83</td>
</tr>
<tr>
<td></td>
<td>GDdefpixreg</td>
<td>Defines pixel registration within grid cell</td>
<td>2-82</td>
</tr>
<tr>
<td></td>
<td>GDdeffield</td>
<td>Defines data fields to be stored in a grid</td>
<td>2-79</td>
</tr>
<tr>
<td></td>
<td>GDdefcomp</td>
<td>Defines a field compression scheme</td>
<td>2-76</td>
</tr>
<tr>
<td></td>
<td>GDwritefield</td>
<td>Writes data to a grid field.</td>
<td>2-121</td>
</tr>
<tr>
<td></td>
<td>GDreadfield</td>
<td>Reads data from a grid field</td>
<td>2-114</td>
</tr>
<tr>
<td></td>
<td>GDwriteattr</td>
<td>Writes/updates attribute in a grid.</td>
<td>2-119</td>
</tr>
<tr>
<td></td>
<td>GDreadattr</td>
<td>Reads attribute from a grid</td>
<td>2-113</td>
</tr>
<tr>
<td></td>
<td>GDsetfillvalue</td>
<td>sets fill value for the specified field</td>
<td>2-118</td>
</tr>
<tr>
<td></td>
<td>GDgetfillvalue</td>
<td>Retrieves fill value for the specified field</td>
<td>2-98</td>
</tr>
<tr>
<td></td>
<td>GDwritefield</td>
<td>Writes data to a grid field.</td>
<td>2-121</td>
</tr>
<tr>
<td></td>
<td>GDdeforig</td>
<td>Defines origin of grid</td>
<td>2-81</td>
</tr>
<tr>
<td></td>
<td>GDdefdim</td>
<td>Defines dimensions for a grid</td>
<td>2-78</td>
</tr>
<tr>
<td></td>
<td>GDdefproj</td>
<td>Defines projection of grid</td>
<td>2-83</td>
</tr>
<tr>
<td></td>
<td>GDdefpixreg</td>
<td>Defines pixel registration within grid cell</td>
<td>2-82</td>
</tr>
<tr>
<td></td>
<td>GDdeffield</td>
<td>Defines data fields to be stored in a grid</td>
<td>2-79</td>
</tr>
<tr>
<td></td>
<td>GDdefcomp</td>
<td>Defines a field compression scheme</td>
<td>2-76</td>
</tr>
<tr>
<td></td>
<td>GDwritefield</td>
<td>Writes data to a grid field.</td>
<td>2-121</td>
</tr>
<tr>
<td></td>
<td>GDreadfield</td>
<td>Reads data from a grid field</td>
<td>2-114</td>
</tr>
<tr>
<td></td>
<td>GDwriteattr</td>
<td>Writes/updates attribute in a grid.</td>
<td>2-119</td>
</tr>
<tr>
<td></td>
<td>GDreadattr</td>
<td>Reads attribute from a grid</td>
<td>2-113</td>
</tr>
<tr>
<td></td>
<td>GDsetfillvalue</td>
<td>sets fill value for the specified field</td>
<td>2-118</td>
</tr>
<tr>
<td></td>
<td>GDgetfillvalue</td>
<td>Retrieves fill value for the specified field</td>
<td>2-98</td>
</tr>
</tbody>
</table>
Table 5-1. Summary of the Grid Interface (2 of 2)

<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset</td>
<td>GDextractregion</td>
<td>read a region of interest from a field</td>
<td>2-95</td>
</tr>
<tr>
<td></td>
<td>GDdeftimeperiod</td>
<td>Define a time period of interest</td>
<td>2-87</td>
</tr>
<tr>
<td></td>
<td>GDdefvrtregion</td>
<td>Define a region of interest by vertical field</td>
<td>2-89</td>
</tr>
<tr>
<td></td>
<td>GDgetpixels</td>
<td>get row/columns for lon/lat pairs</td>
<td>2-99</td>
</tr>
<tr>
<td></td>
<td>GDdupregion</td>
<td>Duplicate a region or time period</td>
<td>2-94</td>
</tr>
<tr>
<td>Tiling</td>
<td>GDdeftile</td>
<td>Define a tiling scheme</td>
<td>2-85</td>
</tr>
</tbody>
</table>

1.5 GCTP Usage

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

1.5.1 GCTP Projection Codes

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

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

* 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.

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

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

<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-180E</td>
</tr>
</tbody>
</table>

1.5.3 GCTP Spheroid Codes

<table>
<thead>
<tr>
<th>Spheroid</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clarke 1866 (default)</td>
<td>(0)</td>
</tr>
<tr>
<td>Clarke 1880</td>
<td>(1)</td>
</tr>
<tr>
<td>Bessel</td>
<td>(2)</td>
</tr>
<tr>
<td>International 1967</td>
<td>(3)</td>
</tr>
<tr>
<td>International 1909</td>
<td>(4)</td>
</tr>
<tr>
<td>WGS 72</td>
<td>(5)</td>
</tr>
<tr>
<td>Everest</td>
<td>(6)</td>
</tr>
<tr>
<td>WGS 66</td>
<td>(7)</td>
</tr>
<tr>
<td>GRS 1980</td>
<td>(8)</td>
</tr>
<tr>
<td>Airy</td>
<td>(9)</td>
</tr>
<tr>
<td>Modified Airy</td>
<td>(10)</td>
</tr>
</tbody>
</table>
1.5.4 GCTP Projection Parameters

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

<table>
<thead>
<tr>
<th>Code &amp; Projection Id</th>
<th>Array Element 1</th>
<th>Array Element 2</th>
<th>Array Element 3</th>
<th>Array Element 4</th>
<th>Array Element 5</th>
<th>Array Element 6</th>
<th>Array Element 7</th>
<th>Array Element 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Geographic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 U T M</td>
<td>Lon/Z</td>
<td>Lat/Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Lambert Conformal C</td>
<td>SMajor</td>
<td>SMinor</td>
<td>STDPR1</td>
<td>STDPR2</td>
<td>CentMer</td>
<td>OriginLat</td>
<td>FE</td>
<td>FN</td>
</tr>
<tr>
<td>6 Polar Stereographic</td>
<td>SMajor</td>
<td>SMinor</td>
<td></td>
<td>LongPol</td>
<td>TrueScale</td>
<td>FE</td>
<td>FN</td>
<td></td>
</tr>
<tr>
<td>7 Polyconic</td>
<td>SMajor</td>
<td>SMinor</td>
<td></td>
<td>CentMer</td>
<td>OriginLat</td>
<td>FE</td>
<td>FN</td>
<td></td>
</tr>
<tr>
<td>9 Transverse Mercator</td>
<td>SMajor</td>
<td>SMinor</td>
<td>Factor</td>
<td>CentMer</td>
<td>OriginLat</td>
<td>FE</td>
<td>FN</td>
<td></td>
</tr>
<tr>
<td>11 Lambert Azimuthal</td>
<td>Sphere</td>
<td></td>
<td></td>
<td>CentLon</td>
<td>CenterLat</td>
<td>FE</td>
<td>FN</td>
<td></td>
</tr>
<tr>
<td>20 Hotin Oblique Merc A</td>
<td>SMajor</td>
<td>SMinor</td>
<td>Factor</td>
<td></td>
<td>OriginLat</td>
<td>FE</td>
<td>FN</td>
<td></td>
</tr>
<tr>
<td>20 Hotin Oblique Merc B</td>
<td>SMajor</td>
<td>SMinor</td>
<td>Factor</td>
<td>AziAng</td>
<td>AzmthPt</td>
<td>OriginLat</td>
<td>FE</td>
<td>FN</td>
</tr>
<tr>
<td>22 Space Oblique Merc A</td>
<td>SMajor</td>
<td>SMinor</td>
<td>IncAng</td>
<td>AscLong</td>
<td></td>
<td>FE</td>
<td>FN</td>
<td></td>
</tr>
<tr>
<td>22 Space Oblique Merc B</td>
<td>SMajor</td>
<td>SMinor</td>
<td>Satnum</td>
<td>Path</td>
<td></td>
<td>FE</td>
<td>FN</td>
<td></td>
</tr>
<tr>
<td>24 Interrupted Goode</td>
<td>Sphere</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99 Integerized Sinusoidal</td>
<td>Sphere</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code &amp; Projection Id</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 Geographic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 U T M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Lambert Conformal C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Polar Stereographic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 Polyconic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 Transverse Mercator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 Lambert Azimuthal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 Hotin Oblique Merc A</td>
<td>Long1</td>
<td>Lat1</td>
<td>Long2</td>
<td>Lat2</td>
<td>Zero</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 Hotin Oblique Merc B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>One</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22 Space Oblique Merc A</td>
<td>PSRev</td>
<td>Srat</td>
<td>PFlag</td>
<td></td>
<td>Zero</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22 Space Oblique Merc B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>One</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 Interrupted Goode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99 Integerized Sinusoidal</td>
<td>NZone</td>
<td></td>
<td></td>
<td>RFlag</td>
<td></td>
<td></td>
<td></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.

Sminor Eccentricity squared of the ellipsoid if less than zero, if zero, a spherical form is assumed, or if greater than zero, the semi-minor axis of ellipsoid.

Sphere Radius of reference sphere. If zero, 6370997 meters is used.

STDPR1 Latitude of the first standard parallel

STDPR2 Latitude of the second standard parallel
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentMer</td>
<td>Longitude of the central meridian</td>
</tr>
<tr>
<td>OriginLat</td>
<td>Latitude of the projection origin</td>
</tr>
<tr>
<td>FE</td>
<td>False easting in the same units as the semi-major axis</td>
</tr>
<tr>
<td>FN</td>
<td>False northing in the same units as the semi-major axis</td>
</tr>
<tr>
<td>TrueScale</td>
<td>Latitude of true scale</td>
</tr>
<tr>
<td>LongPol</td>
<td>Longitude down below pole of map</td>
</tr>
<tr>
<td>Factor</td>
<td>Scale factor at central meridian (Transverse Mercator) or center of projection</td>
</tr>
<tr>
<td>CentLon</td>
<td>Longitude of center of projection</td>
</tr>
<tr>
<td>CenterLat</td>
<td>Latitude of center of projection</td>
</tr>
<tr>
<td>Long1</td>
<td>Longitude of first point on center line (Hotine Oblique Mercator)</td>
</tr>
<tr>
<td>Long2</td>
<td>Longitude of second point on center line (Hotine Oblique Mercator, fmt A)</td>
</tr>
<tr>
<td>Lat1</td>
<td>Latitude of first point on center line (Hotine Oblique Mercator, fmt A)</td>
</tr>
<tr>
<td>Lat2</td>
<td>Latitude of second point on center line (Hotine Oblique Mercator, fmt A)</td>
</tr>
<tr>
<td>AziAng</td>
<td>Azimuth angle east of north of center line (Hotine Oblique Mercator, fmt B)</td>
</tr>
<tr>
<td>AzmthPt</td>
<td>Longitude of point on central meridian where azimuth occurs (Hotine Oblique Mercator, fmt B)</td>
</tr>
<tr>
<td>IncAng</td>
<td>Inclination of orbit at ascending node, counter-clockwise from equator (SOM, fmt A)</td>
</tr>
<tr>
<td>AscLong</td>
<td>Longitude of ascending orbit at equator (SOM, fmt A)</td>
</tr>
<tr>
<td>PSRev</td>
<td>Period of satellite revolution in minutes (SOM, fmt A)</td>
</tr>
<tr>
<td>SRat</td>
<td>Satellite ratio to specify the start and end point of x,y values on earth surface (SOM, fmt A -- for Landsat use 0.5201613)</td>
</tr>
<tr>
<td>PFlag</td>
<td>End of path flag for Landsat: 0 = start of path, 1 = end of path (SOM, fmt A)</td>
</tr>
<tr>
<td>Satnum</td>
<td>Landsat Satellite Number (SOM, fmt B)</td>
</tr>
<tr>
<td>Path</td>
<td>Landsat Path Number (Use WRS-1 for Landsat 1, 2 and 3 and WRS-2 for Landsat 4 and 5.) (SOM, fmt B)</td>
</tr>
</tbody>
</table>
Nzone  Number of equally spaced latitudinal zones (rows); must be two or larger and even

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

Notes:

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

All angles (latitudes, longitudes, azimuths, etc.) are entered in packed degrees/ minutes/ seconds (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
2. Function Reference

2.1 Format

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

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

2.1.1 Swath Interface Functions

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

**SWattach**

hid_t SWattach(hid_t *fid, char *swathname)

- **fid** IN: Swath file id returned by SWopen
- **swathname** IN: Name of swath to be attached

**Purpose**
Attaches to an existing swath within the file.

**Return value**
Returns the swath handle (swathID) if successful 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 file, SwathFile.h5, referred to by the handle, *fid*:

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

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

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

integer   fid
character(*) swathname

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

```fortran
swathid = swattach(fid, "ExampleSwath")
```
Return Information About a Swath Attribute

SWattrinfo

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

swathID IN: Swath id returned by SWcreate or SWattach
attrname IN: Attribute name
numbertype OUT: Data type class ID of attribute
count OUT: Number of attribute elements

Purpose Returns information about a swath attribute

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

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

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

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

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

FORTRAN

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

The equivalent FORTRAN code for the first example above is:

    status = swattrinfo(swathid, "ScalarFloat",nt,count);
Close an HDF-EOS File

**SWclose**

herr_t SWclose(hid_t fid)

- **fid** IN: Swath file id returned by SWopen

**Purpose**
Closes file.

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

**Description**
This routine closes the HDF swath file.

**Example**

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

**FORTRAN**

```fortran
integer function swclose(fid)
integer         fid

The equivalent FORTRAN code for the example above is:

```fortran
status = swclose(fid)
```
Retrive Compression Information for Field

**SWcompinfo**

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

- **swathID** IN: Swath id returned by SWcreate or SWattach
- **fieldname** IN: Fieldname
- **compcode** OUT: HDF compression code
- **compparm** OUT: Compression parameters

**Purpose** Retrieves compression information about a field.

**Return value** Returns SUCCEED(0) if successful 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 SWdefcomp section:

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

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

**FORTRAN**

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

The equivalent FORTRAN code for the example above is:

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

The *compcode* parameter will be set to 4 and *compparm*(1) to 5.
Create a New Swath Structure

SWcreate

hid_t SWcreate(hid_t fid, const char *swathname)

- **fid**: IN: Swath file id returned by SWopen
- **swathname**: IN: Name of swath to be created

**Purpose**: Creates a swath within the file.

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

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

**Example**: In this example, we create a new swath structure, ExampleSwath, in the previously created file, SwathFile.h5.

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

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

**FORTRAN**

```fortran
integer function swcreate(fid,swathname)
integer fid
character*(*) swathname
```

The equivalent FORTRAN code for the example above is:

```fortran
swathid = swcreate(fid, "ExampleSwath")
```
Define a Longitude-Latitude Box Region for a Swath

SWdefboxregion

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

swathID     IN:  Swath id returned by SWcreate or SWattach
cornerlon   IN:  Longitude in decimal degrees of box corners
cornerlat   IN:  Latitude in decimal degrees of box corners
mode        IN:  Cross Track inclusion mode

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

Return value  Returns the swath region ID if successful 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 SWextractregion routine to read all
              the entries of a data field within the region. A cross track is within a region
              if 1) its midpoint is within the longitude-latitude "box"
              (HDFE_MIDPOINT), or 2) either of its endpoints is within the longitude-
              latitude "box" (HDFE_ENDPOINT), or 3) any point of the cross track is
              within the longitude-latitude "box" (HDFE_ANYPOINT), depending on
              the inclusion mode designated by the user. All elements within an
              included cross track are considered to be within the region even though a
              particular element of the cross track might be outside the region. The
              swath structure must have both Longitude and Latitude (or Colatitude)
              fields defined

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

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

    cornerlon[0] = 3.;
    cornerlat[0] = 5.;
    cornerlon[1] = 7.;
    cornerlat[1] = 12.;
regionID = SWdefboxregion(swathID, cornerlon, cornerlat, HDFE_MIDPOINT);

FORTRAN

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

 Integer  swathid
 real*8    cornerlon
 real*8    cornerlat
 integer*4  mode

The equivalent FORTRAN code for the example above is:

parameter (HDFE_MIDPOINT=0)

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

regionid = swdefboxreg(swathid, cornerlon, cornerlat, HDFE_MIDPOINT)
Set Swath Field Compression

**SWdefcomp**

herr_t SWdefcomp(hid_t swathID, int32_t compcode, intn *compparm)

- **swathID** IN: Swath id returned by SWcreate or SWattach
- **compcode** IN: HDF compression code
- **compparm** IN: Compression parameters (if applicable)

**Purpose**
Sets the field compression for all subsequent field definitions.

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

**Description**
This routine sets the HDF field compression for subsequent swath field definitions. The compression does not apply to one-dimensional fields. The compression schemes currently supported are:

- (HDFE_COMP_DEFLLATE=4) and no compression
- (HDFE_COMP_NONE = 0, the default). Deflate compression requires a single integer compression parameter in the range of one to nine with higher values corresponding to greater compression. Compressed fields are written using the standard SWwritefield routine, however, the entire field must be written in a single call. Any portion of a compressed field can then be accessed with the SWreadfield routine. Compression takes precedence over merging so that multi-dimensional fields that are compressed are not merged. The user should refer to the HDF Reference Manual for a fuller explanation of the compression schemes and parameters.

**Example**
Suppose we wish to compress the *Pressure* using run length encoding, the *Opacity* field using deflate compression, the *Spectra* field with skipping Huffman compression, and use no compression for the *Temperature* field.

```c
status = SWdefcomp(swathID, 4, NULL);
status = SWdefdatafield(swathID, "Pressure", "Track,Xtrack", NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);
compparm[0] = 5;
status = SWdefcomp(swathID, 0, compparm);
status = SWdefdatafield(swathID, "Opacity", "Track,Xtrack", NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);
status = SWdefcomp(swathID, 0, NULL);
```
status = SWdefdatafield(swathID, "Temperature", "Track,Xtrack", NULL, H5T_NATIVE_FLOAT, HDFE_AUTOMERGE);

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

FORTRAN

integer function swdefcomp(swathid, compcode, compparm)

integer swathid
integer*4 compcode
integer compparm

The equivalent FORTRAN code for the example above is:

integer*4 HDFE_COMP_NONE
parameter (HDFE_COMP_NONE=0)
integer*4 HDFE_COMP_DEFLATE
parameter (HDFE_COMP_DEFLATE=4)
integer HDFE_NATIVE_FLOAT
parameter (HDFE_NATIVE_FLOAT = 1)
integer compparm(5)

status = swdefcomp(swathid, HDFE_COMP_DEFLATE, compparm)
status = swdefdfld(swathid, "Pressure", "Track,Xtrack", H5T_NATIVE_FLOAT, HDFE_NOMERGE)
compparm(1) = 5
status = swdefcomp(swathid, HDFE_COMP_NONE, compparm)
status = swdefdfld(swathid, "Opacity", "Track,Xtrack", H5T_NATIVE_FLOAT, HDFE_NOMERGE)
Define a New Data Field Within a Swath

SWdefdatafield

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

swathID  IN: Swath id returned by SWcreate or SWattach
fieldname IN: Name of field to be defined
dimlist IN: The list of data dimensions defining the field
maxdimlist IN: The list of maximum data dimensions defining the field
numbertype IN: The number type of the data stored in the field
merge IN: Merge code (HDFE_NOMERGE (0) - no merge, HDFE_AUTOMERGE (1) -merge)

Purpose Defines a new data field within the swath.

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

Description This routine defines geolocation fields to be stored in the swath. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in C order, that is, the last dimension is incremented first. The API will attempt to merge into a single object those fields that share dimensions and in case of multidimensional fields, numbertype. Two and three dimensional fields will be merged into a single three-dimensional object if the last two dimensions (in C order) are equal. If the merge code for a field is set to HDF_NOMERGE (0), the API will not attempt to merge it with other fields. Because merging breaks the one-to-one correspondence between HDF-EOS fields and HDF SDS arrays, it should not be set if the user wishes to access the HDF-EOS field directly using HDF routines or, for example, to create an HDF attribute corresponding to the field. If maximum dimensions are the same as input dimensions, the “NULL” should be passed as a fourth parameter. In case of unlimited dimension (appendable field) the corresponding dimension should be defined by calling SWdefdim() with the H5S_UNLIMITED as third parameter.

Example In this example, we define a three dimensional data field named Spectra with dimensions Bands, DataTrack, and DataXtrack:
status = SWdefdatafield(swathID, "Spectra",
    "Bands,DataTrack,DataXtrack", NULL, H5T_NATIVE_FLOAT, 
    HDFE_AUTOMERGE);

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

FORTRAN

integer function swdefdfld(swathid, fieldname, dimlist, 
    maxdimlist,numbertype,merge)

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

The equivalent FORTRAN code for the example above is:

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

status = swdefdfld(swathid, "Spectra",
    "DataXtrack,DataTrack,Bands", " ", HDFE_NATIVE_FLOAT, 
    HDFE_AUTOMERGE)
Define a New Dimension Within a Swath

**SWdefdim**

```c
herr_t SWdefdim(hid_t swathID, char *dimname, int32_t dim)
```

- **swathID** IN: swath returned by Swcreate or SWattach
- **fieldname** IN: Name of dimension to be defined
- **dim** IN: The size of the dimension

**Purpose**
Defines a new dimension within the swath.

**Return value**
Returns SUCCEED (0) if successful 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 = SWdefdim(swathID, "GeoTrack", 2000);
status = SWdefdim(swathID, "GeoXtrack", 1000);
status = SWdefdim(swathID, "DataTrack", 4000);
status = SWdefdim(swathID, "DataXtrack", 2000);
status = SWdefdim(swathID, "Bands", 5);
```

To specify an unlimited dimension which can be used to define an appendable array, the dimension value should be set to zero or equivalently, H5S_UNLIMITED:

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

**FORTRAN**

```fortran
integer function swdefdim(swathid,fieldname,dim)
integer swathid
character(*) fieldname
integer*4 dim
The equivalent FORTRAN code for the first example above is:
status = swdefdim(swathid, "GeoTrack", 2000)
The equivalent FORTRAN code for the unlimited dimension example above is:
parameter (H5S_UNLIMITED=0)
status = swdefdim(swathid, "Unlim", H5S_UNLIMITED)
```
# Define Mapping Between Geolocation and Data Dimensions

**SWdefdimmap**

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

- **swathID**
  - IN: Swath id returned by SWcreate or 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 = SWdefdimmap(swathID, "GeoTrack", "DataTrack", 0, 2);

status = SWdefdimmap(swathID, "GeoXtrack", "DataXtrack", 1, 2);
```
FORTRAN    integer function
            swdefmap(swathid,geodim,datadim,offset,increment)

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

            The equivalent FORTRAN code for the second example above is:

            status = swdefmap(swathid, "GeoTrack", "DataTrack", 0, 2)
            status = swdefmap(swathid, "GeoXtrack", "DataXtrack", 1, 2)
Define a New Geolocation Field Within a Swath

**SWdefgeofield**

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

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

- **fieldname**  
  IN: Name of field to be defined

- **dimlist**  
  IN: The list of geolocation dimensions defining the field

- **maxdimlist**  
  IN: The list of maximum geolocation dimensions defining the field

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

- **merge**  
  IN: Merge code (HDFE_NOMERGE (0) - no merge, HDFE_AUTOMERGE (1) -merge)

**Purpose**  
Defines a new geolocation field within the swath.

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

**Description**  
This routine defines geolocation fields to be stored in the swath. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in C order, that is, the last dimension is incremented first. The API will attempt to merge into a single object those fields that share dimensions and in case of multidimensional fields, numbertype. Two and three dimensional fields will be merged into a single three-dimensional object if the last two dimensions (in C order are equal). If the merge code for a field is set to 0, the API will not attempt to merge it with other fields. Fields using the unlimited dimension will not be merged. Because merging breaks the one-to-one correspondence between HDF-EOS fields and HDF SDS arrays, it should not be set if the user wishes to access the HDF-EOS field directly using HDF routines or, for example, to create an HDF attribute corresponding to the field. If maximum dimensions are the same as input dimensions, the “NULL” should be passed as a fourth parameter. In case of unlimited dimension (appendable field) the corresponding dimension should be defined by calling SWdefdim() with the H5S_UNLIMITED as third parameter.
Example

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

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

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

**FORTRAN**

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

The equivalent FORTRAN code for the first example above is:

```fortran
parameter (HDFE_NATIVE_FLOAT=1)
parameter (HDFE_AUTOMERGE=1)
status = swdefgfld(swathid, "Longitude", "GeoXtrack,GeoTrack", " ", HDFE_NATIVE_FLOAT,HDFE_AUTOMERGE)
```

The dimensions are entered in FORTRAN order with the first dimension incremented first.
Define Indexed Mapping Between Geolocation and Data Dimension

SWdefidxmap

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

- **swathID**: Swath id returned by SWcreate or SWattach
- **geodim**: Geolocation dimension name
- **datadim**: Data dimension name
- **index**: 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
int32 index[5] = {0, 2, 3, 6, 7};
status = SWdefidxmap(swathID, "IdxGeo", "IdxData", index);
```

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

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

The equivalent FORTRAN code for the example above is:

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

Note: The index array should be 0-based.
Define a Time Period of Interest

SWdeftimeperiod

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

swathID IN: Swath id returned by SWcreate or SWattach
starttime IN: Start time of period
stoptime IN: Stop time of period
mode IN: Cross Track inclusion mode

Purpose Defines a time period for a swath.

Return value Returns the swath period ID if successful 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 SWextractperiod routine to read all the entries of a data field within the time period. A cross track is within a time period if 1) its midpoint is within the time period "box", or 2) either of its endpoints is within the time period "box", or 3) any point of the cross track is within the time period "box", depending on the inclusion mode designated by the user. All elements within an included cross track are considered to be within the time period even though a particular element of the cross track might be outside the time period. The swath structure must have the Time field defined.

Example In this example, we define a time period with a start time of 35232487.2 and a stop time of 36609898.1. We will consider a cross track to be within the time period if either one of the time values at the endpoints of a cross track are within the time period.

```
    starttime = 35232487.2;
    stoptime = 36609898.1;

    periodID = SWdeftimeperiod(swathID, starttime, stoptime, HDFE_ENDPOINT);
```
FORTRAN function swdeftmelper(swathid, starttime, stoptime, mode)

integer*4 swathid
real*8 starttime
real*8 stoptime
integer*4 mode

The equivalent FORTRAN code for the example above is:

parameter (HDFE_ENDPOINT=1)

starttime = 35232487.2
stoptime = 36609898.1

periodID = swdeftmelper(swathID, starttime, stoptime, HDFE_ENDPOINT)
Define a Vertical Subset Region

SWdefvrtregion

int32_t SWdefvrtregion(hid_t swathID, int32_t regionID, char *vertObj, float64 range[])

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

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

**vertObj** IN: Dimension or field to subset by

**range** IN: Minimum and maximum range for subset

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

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

**Description** Whereas the **SWdefboxregion** and **SWdeftimeperiod** routines perform subsetting along the “Track” dimension, this routine allows the user to subset along any dimension. The region is specified by a set of minimum and maximum values and can represent either a dimension index (case 1) or field value range (case 2). In the second case, the field must be one-dimensional and the values must be **monotonic** (strictly increasing or decreasing) in order that the resulting dimension index range be contiguous. (For the current version of this routine, the second option is restricted to fields with number type: INT16, INT32, FLOAT32, FLOAT64.) This routine may be called after **SWdefboxregion** or **SWdeftimeperiod** to provide both geographic or time and “vertical” subsetting. In this case the user provides the id from the previous subset call. (This same id is then returned by the function.) This routine may also be called “stand-alone” by setting the input id to HDFE_NOPREVSUB (-1).

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

The **SWregioninfo** and **SWextractregion** routines work as before, however because there is no mapping performed between geolocation dimensions and data dimensions the field to be subsetted, (the field specified in the call to **SWregioninfo** and **SWextractregion**) must contain the dimension used explicitly in the call to **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:

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

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

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

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

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

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

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

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

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

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

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

integer*4 function swdefvrtreg(swathid, regionid, vertobj, range)
integer swathid
integer*4 regionid
character(*) vertobj
real*8 range

The equivalent FORTRAN code for the examples above is:

parameter (HDFE_NOPREVSUB=-1)
range(1) = 500.
range(2) = 800.
regionid = swdefvrtreg(swathid, HDFE_NOPREVSUB, "Pressure", range)
range(1) = 3 ! Note 1-based element numbers
range(2) = 6
regionid = swdefvrtreg(swathid, HDFE_NOPREVSUB, "DIM:Height", range)
regionid = swdefvrtreg(swathid, subsetid, "Pressure", range)
regionid = swdefvrtreg(swathid, regionid, "FreqRange", freq)
Detach from a Swath Structure

**SWdetach**

```c
herr_t SWdetach(hid_t swathID)
```

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

**Purpose**
Detaches from swath interface.

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

**Description**
This routine should be run before exiting from the swath file for every swath opened by SWcreate or SWattach.

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

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

**FORTRAN**

```fortran
integer function swdetach(swathid)

integer swathid

The equivalent FORTRAN code for the example above is:

```c
status = swdetach(swathid)
```
Retrieve Size of Specified Dimension

SWdiminfo

hsiz_t SWdiminfo(hid_t swathID, char *dimname)

- **swathID**: IN: Swath id returned by SWcreate or SWattach
- **dimname**: IN: Dimension name

**Purpose**
Retrieve size of specified dimension.

**Return value**
Size of dimension if successful or FAIL (-1) otherwise. If -1, could signify an improper swath id or dimension name.

**Description**
This routine retrieves the size of specified dimension.

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

```c
    dsize = SWdiminfo(swathID, "GeoTrack");
```

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

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

```fortran
    integer swathid
    character(*) dimname
```

The equivalent FORTRAN code for the example above is:

```fortran
    dsize = swdiminfo(swathid, "GeoTrack")
```
**Duplicate a Region or Period**

**SWdupregion**

`int32_t SWdupregion(int32_t regionID)`

- **regionID**
  - **IN**: Region or period id returned by SWdefboxregion, SWdeftimeperiod, or SWdefvrregion.

- **Purpose**: Duplicates a region.

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

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

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

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

**FORTRAN**

```fortran
integer*4 swdupreg(regionid)
integer*4 regionid
```

The equivalent FORTRAN code for the example above is:

```fortran
parameter (HDFE_MIDPOINT=0)
regionid = swdefboxreg(swathid, cornerlon, cornerlat, HDFE_MIDPOINT)
regionid2 = swdupreg(regionid)
regionid = swdefvrtrreg(swathid, regionid, 'Pressure', rangePres)
regionid2 = swdefvrtrreg(swathid, regionid2, 'Temperature', rangeTemp)
```
Read Data from a Defined Time Period

**SWextractperiod**

herr_t SWextractperiod(hid_t swathID, hid_t periodID, char *fieldname, int32_t external_mode, void *buffer)

- **periodID** IN: Period id returned by SWdeftimeperiod
- **fieldname** IN: Field to subset
- **external_mode** 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.

**Description**
This routine reads data into the data buffer from the subsetted time period. Only complete crosstracks are extracted. If the `external_mode` flag is set to `HDFE_EXTERNAL` (1) then the geolocation fields and the data field can be in different swaths. If set to `HDFE_INTERNAL` (0), then these fields must be in the same swath structure.

**Example**
In this example, we read data within the subsetted time period defined in `SWdeftimeperiod` from the `Spectra` field. Both the geolocation fields and the `Spectra` data field are in the same swath.

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

**FORTRAN**

```fortran
integer function swextper(periodid, fieldname, external_mode, buffer)
    integer*4       periodid
    character(*)    fieldname
    integer*4       external_mode
    <valid type>    buffer(*)

The equivalent FORTRAN code for the example above is:
```
```
Read Data from a Geographic Region

SWextractregion

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

* swathID  IN:  Swath id returned by SWcreate or SWattach
* regionID  IN:  Region id returned by SWdefboxregion
* fieldname  IN:  Field to subset
* external_mode  IN:  External geolocation mode
* buffer  OUT:  Data buffer

Purpose  Extracts (reads) from subsetted region.

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

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

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

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

FORTRAN  integer function swextreg(swathid, regionid, fieldname, external_mode, buffer)

integer  swathid
integer*4  regionid
character(*)  fieldname
integer*4  external_mode
<valid type>  buffer(*)

The equivalent FORTRAN code for the example above is:

parameter (HDFE_INTERNAL=0)

status = swextreg(swathid, regionid, "Spectra", HDFE_INTERNAL, datbuf)
Retrieve Information About a Swath Field

**SWfieldinfo**

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

- **swathID** IN: Swath id returned by SWcreate or SWattach
- **fieldname** IN: Fieldname
- **rank** OUT: Rank of field
- **dims** OUT: Array containing the dimension sizes of the field
- **numbertype** OUT: Data type class ID of the field
- **dimlist** OUT: List of dimensions in field

**Purpose**
Retrieves 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:

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

The return parameters will have the following values:

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

If one of the dimensions in the field is appendable, then the current value for that dimension will be returned in the `dims` array.
FORTRAN  integer function swfldinfo(swathid, fieldname, rank, dims, numbertype, dimlist)

  Integer swathid
  character(*) fieldname
  integer rank
  integer*4 dims(*)
  integer numbertype
  character dimlist

  The equivalent FORTRAN code for the example above is:

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

  The return parameters will have the following values:

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

  Note that the dimensions array and dimension list are in FORTRAN order.
Get Fill Value for a Specified Field

SWgetfillvalue

herr_t SWgetfillvalue(hid_t swathID, char *fieldname, void *fillvalue)

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

Purpose Retrieves fill value for the specified field.
Return value Returns SUCCEED (0) if successful 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 = SWgetfillvalue(swathID, "Temperature", &tempfill);
```

FORTRAN integer function

swgetfill(swathid,fieldname,fillvalue)

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

The equivalent FORTRAN code for the example above is:

```fortran
status = swgetfill(swathid, "Temperature", tempfill)
```
Retrieve type of dimension mapping when first dimension is geodim

**SWgeomapinfo**

**herr_t SWgeomapinfo(hid_t swathID, char *geodim)**

- **swathID** IN: Swath id returned by SWcreate or 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 Swdefidxmap.

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

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

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

**FORTRAN**

```c
integer function swgmapinfo(swathid, geodim)
  integer swathid
  character(*) geodim

The equivalent FORTRAN code for the example above is:

```c
status = swgmapinfo(swathid, geodim)
```
Retrieve Indexed Geolocation Mapping

SWidxmapinfo

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

- **swathID**: IN: Swath id returned by SWcreate or SWattach
- **geodim**: IN: Indexed Geolocation dimension name
- **datadim**: IN: Indexed Data dimension name
- **index**: OUT: Mapping offset

**Purpose**: Retrieve indexed array of specified geolocation mapping.

**Return value**: Returns size of indexed array if successful or FAIL (-1) otherwise. A typical reason for failure is the specified mapping does not exist.

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

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

```c
idxsz = 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 swimapinfo(swathid, geodim, datadim, index)
integer swathid
character(*) geodim
character(*) datadim
integer*4 index(*)
```

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

```fortran
status = swimapinfo(swathid, "IdxGeo", "IdxData", index)
```
Retrieve Information Swath Attributes

**SWinqattrs**

```
int32_t SWinqattrs(hid_t swathID, char *attrlist, int32_t *strbufsize)
```

**Parameters**

- **swathID**
  - IN: Swath id returned by SWcreate or SWattach
- **attrlist**
  - OUT: Attribute list (entries separated by commas)
- **strbufsize**
  - OUT: String length of attribute list

**Purpose**
Retrieve information about attributes defined in swath.

**Return value**
Number of attributes found 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. We assume that there are two attributes stored, `attrOne` and `attr_2`:

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

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

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

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

**FORTRAN**

```
integer*4 function swinqattrs(swathid, attrlist, strbufsize)
```

```fortran
integer swathid
character(*) attrlist
integer*4 strbufsize
```

The equivalent FORTRAN code for the example above is:

```fortran
nattr = swinqattrs(swathid, attrlist, strbufsize)
```
Retrieve Information About Data Fields Defined in Swath

**SWinqdatafields**

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

- `swathID` **IN:** Swath id returned by SWcreate or SWattach
- `fieldlist` **OUT:** Listing of data fields (entries separated by commas)
- `rank` **OUT:** Array containing the rank of each data field
- `numbertype` **OUT:** Array containing the data type class ID for each data field

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

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

**Description**
The field list is returned as a string with each data field separated by commas. The `rank` and `numbertype` arrays will have an entry for each field. Output parameters set to `NULL` will not be returned.

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

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

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

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

**FORTRAN**

```fortran
integer*4 function swinqdflds(swathid, fieldlist, rank, numbertype)
integer swathid
character(*) fieldlist
integer*4 rank(*)
integer numbertype(*)
```

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

```fortran
nflds = swinqdflds(swathid, fieldlist, rank, numbertype)
```
Retrieve Information About Data Type of a Data Field Defined in Swath

**SWinqdatatype**

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

- **swathID** IN: Swath id returned by SWcreate or SWattach
- **fieldname** IN:: String containing the name of a field
- **fieldgroup** IN: Group (Data Field /Geolocation Field ) the data field belongs to
- **datatype** OUT: Data type ID of a data field dataset
- **classid** OUT: Data type class ID
- **order** OUT: Byte order of an atomic datatype
- **size** OUT: Size of a datatype (in bytes)

**Purpose** Retrieve information about the data type of a specified data field.

**Return value** Status variable (0 if successful or -1 otherwise).

**Description**

**Example**

FORTRAN
Retrieve Information About Dimensions Defined in Swath

SWinqdims

int32_t SWinqdims(hid_t swathID, char *dimname, int32_t dims[])

swathID IN: Swath id returned by SWcreate or SWattach
dimname OUT: Dimension list (entries separated by commas)
dims OUT: Array containing size of each dimension

Purpose Retrieve information about all of the dimensions defined in swath.

Return value Number of dimension entries found if 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:

\[
\text{ndims} = \text{SWinqdims(swathID, dimname, dims);} \\
\]

The parameter, dimname, will have the value:

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

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

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

integer swathid
character(*) dimname
integer*4 dims(*)

The equivalent FORTRAN code for the example above is:

\[
\text{ndims} = \text{swinqdims(swathid, dimname, dims)} \\
\]
Retrieve Information About Geolocation Fields Defined in Swath

**SWinqgeofields**

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

- **swathID** IN: Swath id returned by SWcreate or SWattach
- **fieldlist** OUT: Listing of geolocation fields (entries separated by commas)
- **rank** OUT: Array containing the rank of each geolocation field
- **numbertype** OUT: Array containing the data type class ID for each geolocation field

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

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

**Description** The field list is returned as a string with each geolocation field separated by commas. The `rank` and `numbertype` arrays will have an entry for each field. Output parameters set to `NULL` will not be returned.

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

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

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

**FORTRAN**

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

The equivalent FORTRAN code for the example above is:

```fortran
nflds = swinqgflds(swathid, fieldlist, rank, numbertype)
```
Retrieve Information About Indexed Mappings Defined in Swath

**SWinqidxmaps**

```c
int32_t SWinqidxmaps(hid_t swathID, char *idxmap, int32_t idxsizes[])
```

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

```c
nidxmaps = SWinqidxmaps(swathID, idxmap, idxsizes);
```

The variable, `idxmap`, will contain the string:

"IdxGeo/IdxData" with `nidxmaps = 1` and `idxsizes[1] = 5`.

**FORTRAN**

```fortran
integer*4 function swinqimaps(swathid, dimmap, idxsizes)
integer swathid
character(*) dimmap
integer*4 idxsizes(*)

The equivalent FORTRAN code for the example above is:

```c
nidxmaps = swinqimaps(swathid, dimmap, idxsizes)
```
Retrieve Information About Dimension Mappings Defined in Swath

SWinqmaps

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

swathID
IN: Swath id returned by SWcreate or 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 = SWinqmaps(swathID, dimmap, offset, increment);
```

The variable, dimmap, will contain the string: "GeoTrack/DataTrack,GeoXtrack/DataXtrack" with nmaps = 2, offset[2]={0,1} and increment[2]={2,2}.

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

The equivalent FORTRAN code for the example above is:

```fortran
nmaps = swinqmaps(swathid, dimmap, offset, increment)
```
Retrieve Swath Structures Defined in HDF-EOS File

**SWinqswath**

```c
int32_t SWinqswath(char *filename, char *swathlist, int32_t *strbufsize)
```

- **filename**  
  IN: HDF-EOS filename

- **swathlist**  
  OUT: Swath list (entries separated by commas)

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

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

**Return value**
Number of swaths found 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, `HDFEOS.h5`. We assume that there are two swaths stored, `SwathOne` and `Swath_2`:

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

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

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

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

```
"SwathOne,Swath_2".
```

**FORTRAN**

```fortran
integer*4 function swinqswath(filename,swathlist,strbufsize)
character(*) filename
character(*) swathlist
integer*4 strbufsize
The equivalent FORTRAN code for the example above is:

```c
nswath = SWinqswath('HDFEOS.h5', swathlist, strbufsize)
```
Retrieve Offset and Increment of Specific Dimension Mapping

**SWmapinfo**

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

- **swathID**  
  IN: Swath id returned by SWcreate or SWattach
- **geodim**  
  IN: Geolocation dimension name
- **datadim**  
  IN: Data dimension name
- **offset**  
  OUT: Mapping offset
- **increment**  
  OUT: Mapping increment

**Purpose**  
Retrieve offset and increment of specific monotonic geolocation mapping.

**Return value**  
Returns SUCCEED (0) if successful 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 = SWmapinfo(swathID, "GeoTrack", "DataTrack", &offset, &increment);
```

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

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

The equivalent FORTRAN code for the example above is:

```fortran
status = swmapinfo(swathid, "GeoTrack", "DataTrack", offset, increment)
```
Return Number of Specified Objects in a Swath

SWnentries

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

swathID IN: Swath id returned by SWcreate or SWattach
entrycode IN: Entrycode
strbufsize OUT: String buffer size

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

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

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

The entry codes are:

- HDFE_NENTDIM (0) - Dimensions
- HDFE_NENTMAP (1) - Dimension Mappings
- HDFE_NENTIMAP (2) - Indexed Dimension Mappings
- HDFE_NENTGFLD (3) - Geolocation Fields
- HDFE_NENTDFLD (4) - Data Fields

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

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

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

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

integer swathid
integer*4 entrycode
integer*4 bufsize

The equivalent FORTRAN code for the example above is:

parameter (HDFE_NENTMAP=1)
nmaps = swnentries(swathid, HDFE_NENTMAP, bufsize)
Open HDF-EOS File

**SWopen**

hid_t  SWopen(char *filename, unsigned flags)

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

*flags* 
IN: *File access flags*

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

**Return value**
Returns the swath file id handle (fid) if successful or FAIL (-1) otherwise.

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

Access flags:

- **H5F_ACC_RDONLY**: Open for read only. If file does not exist, error
- **H5F_ACC_RDWR**: Open for read/write. If file does not exist, error
- **H5F_ACC_TRUNC**: If file exist, truncate it, then open a new file for read/write
- **H5F_ACC_EXCL**: Fail if file already exists
- **H5F_ACC_DEBUG**: Print debug information
- **H5F_ACC_DEFAULT**: Apply default file access and creation properties

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

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

**FORTRAN**

```fortran
integer function swopen(filename,flags)
character(*) filename
integer flags

The access codes should be defined as parameters:

parameter (H5F_ACC_TRUNC = 2)

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

```c
fid = swopen("SwathFile.h5", H5F_ACC_TRUNC)
```
Return Information About a Defined Time Period

**SWperiodinfo**

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

- **swathID**: IN: Swath id returned by SWcreate or SWattach
- **periodID**: IN: Period id returned by SWdeftimeperiod
- **fieldname**: IN: Field to subset
- **ntype**: OUT: Data type class ID 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 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 = SWperiodinfo(SWid, periodID, "Spectra", &ntype, &rank, dims, &size);

/* Allocate space */
datbuf = (float64 *) malloc(size);
```
FORTRAN integer function swperinfo(swathid, periodid, fieldname, ntype, rank, dims, size)

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>swathid</td>
</tr>
<tr>
<td>integer*4</td>
<td>periodid</td>
</tr>
<tr>
<td>character*(*</td>
<td>fieldname</td>
</tr>
<tr>
<td>integer</td>
<td>ntype</td>
</tr>
<tr>
<td>integer*4</td>
<td>rank</td>
</tr>
<tr>
<td>integer*4</td>
<td>dims(*)</td>
</tr>
<tr>
<td>integer*4</td>
<td>size</td>
</tr>
</tbody>
</table>

The equivalent FORTRAN code for the example above is:

```fortran
status = swperinfo(swid, periodid, "Spectra", ntype, rank, dims, size)
```
Close Swath Ragged Array

SWraclose

herr_t SWraclose(hid_t swathID, char *raname)

swathID  IN:   Swath id returned by SWcreate or SWattach
raname   IN:   Ragged array name
Purpose  Closes the ragged array
Return value  Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or incorrect ragged array name.
Example  In this example, we close the ragged array with the name "RA_1":
        status = Swraclose(SwathID,"RA_1");
FORTRAN integer function swraclose(swathid, raname)
        integer swathid
        character*(*) raname
        The equivalent FORTRAN code for the example above is:
        status = swraclose(swathid, "RA_1")
Define Swath Ragged Array

SWradefine

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

swathID      IN: Swath id returned by SWcreate or SWattach
raname       IN: Ragged array name
chunk_size   IN: Array containing the size of each chunk:
               chunk_size[0] - length
               chunk_size[1] - width
numbertype   IN: Data type of ragged array data

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

Example   In this example, we define the ragged array with the name "RA_1":

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

FORTRAN    integer function swradefine(swathid,raname,chunksz, )
            integer swathid
            character(*) raname
            integer*4 chunksz(2)

The equivalent FORTRAN code for the example above is:

status = swradefine(swathid, "RA_1", chunksz, )
Open Swath Ragged Array

**SWraopen**

hid_t SWraopen(hid_t swathID, const char *raname)

- **swathID** (IN): Swath id returned by SWcreate or SWattach
- **raname** (IN): Ragged array name

Purpose: Open ragged array in a swath

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

Example: In this example, we open the ragged array with the name "RA_1":

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

FORTRAN: integer function swraopen(swathid, raname)

```fortran
integer swathid
character(*) raname
```

The equivalent FORTRAN code for the example above is:

```fortran
raid = swraopen(swathid, "RA_1")
```
Read Rows from the Swath Ragged Array

**SWraread**

```c
herr_t SWraread(hid_t swathID, char raname, hsize_t start_row, hsize_t nrows, hid_t numbertype, hsize_t size[], void **buf))
```

- **swathID**  
  IN: Swath id returned by SWcreate or SWattach
- **raname**  
  IN: Ragged array name
- **start_row**  
  IN: Row at which the reading will start
- **nrows**  
  IN: number of rows to read out
- **numbertype**  
  IN: Data type of ragged array data
- **size[]**  
  IN: Array (of nrows elements) containing lengths of rows to read out
- **buf[]**  
  OUT: Pointers to buffers containing the data

**Purpose**  
Read out data from the ragged array in a swath

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

**Example**  
In this example, we read two rows (starting from row 3) from the ragged array with the name "RA_1":

```c
int32_t *buf[2];
for (i =0; i< 2; i++){
  buf[i] = (int32_t *)calloc(10+i, sizeof(int32_t *));
}
size[0] = 10;
size[1] = 11;
status = SWraread(SwathID,"RA_1",3,2 H5T_NATIVE_INT, size,(void **)buf);
```

**FORTRAN**

```fortran
integer function swraread(swathid,raname,start,nrows,ntype,size,buf )
integer swathid
character*(*) raname
integer*4
```
The equivalent *FORTRAN* code for the example above is:

```
status = swraread(swathid, "RA_1", 3, 2, 0, size, buf)
```
Write Rows to the Swath Ragged Array

SWrawrite

herr_t SWrawrite(hid_t swathID, char raname, hssize_t start_row, hsize_t nrows, hid_t numbertype, hsize_t size[], void **buf))

- *swathID* IN: Swath id returned by SWcreate or SWattach
- *raname* IN: Ragged array name
- *start_row* IN: Row at which the writing will start
- *nrows* IN: Number of rows to write to
- *numbertype* IN: Data type of ragged array data
- *size[]* IN: Array (of nrows elements) containing lengths of rows to write to
- *buf[]* IN: Pointers to buffers containing the data

Purpose
Write data to the ragged array in a swath

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

Example
In this example, we write two rows (starting from row 3) to the ragged array "RA_1":

```
size[0] = 10;
size[1] = 11;
status = SWrawrite(SwathID,"RA_1",3,2 H5T_NATIVE_INT, size,(void **)buf);
```

FORTRAN

```
integer function swrawrite(swathid,raname,start,nrows,ntype,size,buf )

integer swathid
character(*) raname
integer*4 size(*)
<valid type> buf(*)
```

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

Parameter(HDFE_NATIVE_INT=0)

```
status = swrawrite(swathid, "RA_1",3,2, HDFE_NATIVE_INT,size,buf)
```
Read Swath Attribute

**SWreadattr**

```c
herr_t SWreadattr(hid_t swathID, char *attrname, void *datbuf)
```

- **swathID** IN: Swath id returned by SWcreate or SWattach
- **attrname** IN: Attribute name
- **datbuf** OUT: Buffer allocated to hold attribute values

Purpose

Reads attribute from a swath.

Return value

Returns SUCCEED (0) if successful 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 = SWreadattr(swathID, "ScalarFloat", &f32);
```

FORTRAN

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

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

The equivalent FORTRAN code for the example above is:

```fortran
parameter (HDFE_NATIVE_FLOAT=1)
status = swrdattr(swathid, "ScalarFloat", f32)
```
Read Data From a Swath Field

SWreadfield

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

*swathID* IN: Swath id returned by SWcreate or 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 value for *stride* is 1 and is used if this parameter is set to *NULL*.

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

    float32 track[1000];
    hssize_t start[2] = {10, 1};
    hsize_t edge[2] = {1, 1000};
    status = SWreadfield(swathID, "Longitude", start, NULL, edge, track);
FORTRAN    integer function
            swrdfld(swathid, fieldname, start, stride, edge, buffer)

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

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

            The equivalent FORTRAN code for the example above is:

            real*4 track(1000)
            integer*4 start(2), stride(2), edge(2)
            start(1) = 0
            start(2) = 10
            stride(1) = 1
            stride(2) = 1
            edge(1) = 1000
            edge(2) = 1

            status = swrdfld(swathid, "Longitude", start, stride, edge, track);
Define a Longitude-Latitude Box Region for a Swath

SWregionindex

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

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

**cornerlon IN:** Longitude in decimal degrees of box corners

**cornerlat IN:** Latitude in decimal degrees of box corners

**mode IN:** Cross Track inclusion mode

**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 SWdefboxregion is the geolocation track dimension name and the range of that dimension are returned in addition to a regionID. Other than that difference they are the same function and this function is used just like SWdefboxregion. This routine defines a longitude-latitude box region for a swath. It returns a swath region ID which is used by the SWextractregion routine to read all the entries of a data field within the region. A cross track is within a region if 1) its midpoint is within the longitude-latitude "box" (HDFE_MIDPOINT), or 2) either of its endpoints is within the longitude-latitude "box" (HDFE_ENDPOINT), or 3) any point of the cross track is within the longitude-latitude "box" (HDFE_ANYPOINT), depending on the inclusion mode designated by the user. All elements within an included cross track are considered to be within the region even though a particular element of the cross track might be outside the region. The swath structure must have both Longitude and Latitude (or Colatitude) fields defined.

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

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

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

FORTRAN integer*4 function swregidx(swathid, cornerlon, cornerlat, mode, geodim, idxrange)
integer          swathid
real*8           cornerlon
real*8           cornerlat
integer*4        mode
character(*)     geodim
integer*4        idxrange(*)

The equivalent FORTRAN code for the example above is:

parameter (HDFE_MIDPOINT=0)
cornerlon(1) = 3.
cornerlat(1) = 5.
cornerlon(2) = 7.
cornerlat(2) = 12.

regionid = swregidx(swathid, cornerlon, cornerlat,
                        HDFE_MIDPOINT, geodim, idxrange)
Return Information About a Defined Region

**SWregioninfo**

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

- `swathID` **IN:** Swath id returned by SWcreate or SWattach
- `regionID` **IN:** Region id returned by SWdefboxregion
- `fieldname` **IN:** Field to subset
- `ntype` **OUT:** Data type class ID 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 `SWdefboxregion` for the Spectra field. We use this to allocate space for data in the subsetted region.

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

/* Allocate space */
datbuf = (float64 *) malloc(size);
```
FORTRAN

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

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

The equivalent FORTRAN code for the example above is:

status = swreginfo(swid, regionid, "Spectra", ntype, rank, dims, size)
Set Fill Value for a Specified Field

SWsetfillvalue

herr_t SWsetfillvalue(hid_t swathID, char *fieldname, void *fillvalue)

swathID      IN: Swath id returned by SWcreate or SWattach
fieldname    IN: Fieldname
fillvalue    IN: Pointer to the fill value to be used

Purpose      Sets fill value for the specified field.

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

Description  The fill value is placed in all elements of the field which have not been explicitly defined. The field must have 2 or more dimensions.

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

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

FORTRAN      integer function

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

The equivalent FORTRAN code for the example above is:

            status = swsetfill(swathid, "Temperature", -999.0)
Update map index for a specified region

SWupdateidxmap

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

swathID IN: Swath id returned by SWcreate or Swattach.

regionID IN: Region id returned by 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 Theis routine retrieves the size of he indexed array and the array of indexed elements of the specified geolocation mapping for the specified region.

Example In this example, we retrieve information about the indexed mapping between the “IdxGeo” and “IdxData” dimensions, defined by Swdefboxregion:

/* Get size of index_region array */
idxsz = SWupdateidxmap(swathID, regionID, index, NULL, indices);

/* Allocate memory for index_region */
index_region = (int32*)malloc(sizeof(int32) * idxsz);

/* Get the array index_region */
idxsz = Swupdateidxmap(swathID, regionID, index, index_region, indices);

FORTRAN integer*4 function swupimap(swathid, regionid, indexin, indexout)

integer swathid

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

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

Note: The indexed arrays should be 0-based.
Write/Update Swath Attribute

SWwriteattr

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

- **swathID** IN: Swath id returned by SWcreate or 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 attribute in a swath.

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

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

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

```c
cnt[0] = 1;

f32 = 3.14;

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

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

```c
f32 = 3.14159;

status = SWwriteattr(swathid, "ScalarFloat", H5T_NATIVE_FLOAT, cnt, &f32);
```
FORTRAN  integer function swwrattr(swathid, attrname, ntype, count, datbuf)
           integer        swathid
           character(*)   attrname
           integer        ntype
           integer*4      count(*)
           <valid type>   datbuf(*)

The equivalent FORTRAN code for the first example above is:

   parameter (DFNT_FLOAT32=5)
   f32 = 3.14

   status = swwrattr(swathid, "ScalarFloat", DFNT_FLOAT32, 1, f32)
Write Data to a Swath Field

**SWwritefield**

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

**swathID**  IN:  Swath id returned by SWcreate or 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 value for stride is 1 is used if this parameter is set to NULL. It is the users responsibility to make sure the data buffer contains sufficient entries to write to the field. Note that the data buffer for a compressed field must be the size of the entire field as incremental writes are not supported by the underlying HDF5 routines.

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

```c
float32 longitude [2000][1000];
hssize_t start[2] = {10,0};
hsize_t edge[2] = {2000, 1000};
/* Define elements of longitude array */
status = SWwritefield(swathID, "Longitude", start, NULL, edge, longitude);
```
We now update Track 10 (0-based) in this field:

```c
float32 newtrack[1000];

hssize_t start[2]=(0,10);

hsize_t edge[2]=(1,1000);

/* Define elements of newtrack array */

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

**FORTRAN**

```fortran
integer function swwrflfd(swathid,fieldname,start,stride,edge,data)

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

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

The equivalent FORTRAN code for the example above is:

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

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

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

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

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

```fortran
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 = swwrflld(swathid, "Longitude", start, stride, edge, newtrack)

2.1.2 Grid Interface Functions

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

GDattach

hid_t GDattach(hid_t fid, char *gridname)

*fid* IN: Grid file id returned by GDopen

*gridname* IN: Name of grid to be attached

Purpose Attaches to an existing grid within the file.

Return value Returns the grid handle(gridID) if successful 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 file, GridFile.h5, referred to by the handle, *fid*:

gridID = GDattach(fid, "ExampleGrid");

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

FORTRAN integer function gdattach(fid, gridname)

integer *fid*

character(*) *gridname*

The equivalent FORTRAN code for the example above is:

gridid = gdattach(fid, "ExampleGrid")
Return Information About a Grid Attribute

GDattrinfo

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

gridID IN: Grid id returned by GDcreate or GDattach
attrname IN: Attribute name
numbertype OUT: Data type class ID of attribute
count OUT: Number of total bytes in attribute

Purpose Returns information about a grid attribute

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

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

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

status = GDattrinfo(pointID, "ScalarFloat", &nt, &count);
The nt variable will have the value 1 and count will have the value 4.

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

The equivalent FORTRAN code for the first example above is:

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

status = GDdefproj(GDid_som, GCTP_SOM, NULL, NULL, projparm);

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

Close an HDF-EOS File

**GDclose**

herr_t GDclose(hid_t \textit{fid})

\textit{fid} \quad \textbf{IN:} \quad \text{Grid file id returned by GDopen}

**Purpose** Closes file.

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

**Description** This routine closes the HDF grid file.

**Example**

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

**FORTRAN**

```fortran
integer function gdclose(int32 \textit{fid})
integer \textit{fid}
```

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

```fortran
status = gdclose(fid)
```
Retreive Compression Information for Field

**GDcompinfo**

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

- **gridID** (IN): Grid id returned by GDcreate or GDattach
- **fieldname** (IN): Fieldname
- **compcode** (OUT): HDF compression code
- **compparm** (OUT): Compression parameters

**Purpose**
Retrieves compression information about a field.

**Return value**
Returns SUCCEED(0) if successful 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 *GDdefcomp* section:

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

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

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

integer  gridid
character*(*) fieldname
character*(*) compcode
integer  compparm(*)

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

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

The `compcode` parameter will be set to 4 and `compparm(1)` to 5.
Create a New Grid Structure

GDcreate

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

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

**Purpose**: Creates a grid within the file.

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

**Description**: The grid is created as a Vgroup within the HDF file with the name gridname and class GRID. This routine establishes the resolution of the grid, ie, the number of rows and columns, and it's location within the complete global projection through the upleftpt and lowrightpt arrays. These arrays should be in meters for all GCTP projections other than the Geographic Projection, which should be in packed degree format. q.v. below.

**Example**: In this example, we create a UTM grid bounded by 54 E to 60 E longitude and 20 N to 30 N latitude. We divide it into 120 bins along the x-axis and 200 bins along the y-axis

```c
uplft[0]=210584.50041;
uplft[1]=3322395.95445;
lowrgt[0]=813931.10959;
lowrgt[1]=2214162.53278;
xdim=120;
ydim=200;
gridID = GDcreate(fid, "UTMGrid", xdim, ydim, uplft, lowrgt);
```
The grid structure is then referenced by subsequent routines using the handle, gridID.

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

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

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

Note:

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

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

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

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

To convert lat/long to x-y coordinates, it is also possible to use SDP Toolkit routines: PGS_GCT_Init() or PGS_GCT_Proj(). More information is contained in the SDP Toolkit Users Guide for the ECS Project (333-CD-500-001).
The equivalent FORTRAN code for the example above is:

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

The default values for the Polar Stereographic and Goode Homolosine can be designated by setting all elements in the `uplft` and `lowrgt` arrays to 0.
Define Region of Interest by Latitude/Longitude

**GDdefboxregion**

hid_t GDdefboxregion(hid_t gridID, float64 cornerlon[], float64 cornerlat[])

- **gridID**
  - IN: Grid id returned by GDcreate or GDattach
- **cornerlon**
  - IN: Longitude in decimal degrees of box corners
- **cornerlat**
  - IN: Latitude in decimal degrees of box corners

**Purpose**
Defines a longitude-latitude box region for a grid.

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

**Description**
This routine defines a longitude-latitude box region for a grid. It returns a grid region ID which is used by the 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 = GDdefboxregion(GDid, cornerlon, cornerlat);
```

**FORTRAN**

```fortran
integer function gddefboxreg(gridid, cornerlon, cornerlat)
   integer      gridid
   real*8      cornerlon
   real*8      cornerlat
   regionid = gddefboxreg(gridid, cornerlon, cornerlat)
```

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

```fortran
cornerlon(1) = 0.
cornerlat(1) = 90.
cornerlon(2) = 90.
cornerlat(2) = 0.
regionid = gddefboxreg(gridid, cornerlon,cornerlat)
```
Set Grid Field Compression

**GDdefcomp**

```c
herr_t GDdefcomp(hid_t gridID, int32_t compcode, intn compparm[])
```

- **gridID** IN: Grid id returned by GDcreate or GDattach
- **compcode** IN: Compression method
- **compparm** IN: Compression parameters (if applicable)

**Purpose**
Sets the field compression for all subsequent field definitions.

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

**Description**
This routine sets the HDF field compression for subsequent grid field definitions. The compression does not apply to one-dimensional fields. The compression methods currently supported are: deflate (HDFE_COMP_DEFLATE=4) and no compression (HDFE_COMP_NONE = 0, the default). Deflate compression requires a single integer compression parameter in the range of one to nine with higher values corresponding to greater compression. Compressed fields are written using the standard `GDwritefield` routine, however, the entire field must be written in a single call. If this is not possible, the user should consider tiling. See `GDdeftile` for further information. Any portion of a compressed field can then be accessed with the `GDreadfield` routine. Compression takes precedence over merging so that multi-dimensional fields that are compressed are not merged. The user should refer to the HDF Reference Manual for a fuller explanation of the compression schemes and parameters.

**Example**
Suppose we wish to compress the *Opacity* field using deflate compression, and use no compression for the *Temperature* field.

```c
status = GDdeffield(gridID, "Pressure", "YDim,XDim", NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);
compparm[0] = 5;
status = GDdefcomp(gridID, HDFE_COMP_DEFLATE, compparm);
status = GDdefcomp(gridID, HDFE_COMP_NONE, NULL);
status = GDdeffield(gridID, "Temperature", "YDim,XDim", NULL, DFNT_FLOAT32, HDFE_AUTOMERGE);
```
Note that the HDFE_AUTOMERGE parameter will be ignored in the Temperature field definition.

**FORTRAN**

```fortran
integer function gddefcomp(gridid, compcode, compparm)
  integer       gridid
  integer       compcode
  integer       compparm

The equivalent FORTRAN code for the example above is:

```fortran
parameter (HDFE_COMP_NONE=0)
parameter (HDFE_COMP_DEFLATE=4)
integer compparm(5)
compparm(1) = 5
status = gddefcomp(gridid, HDFE_COMP_DEFLATE, compparm)
status = gdeffld(gridid, "Opacity", "YDim,XDim",
HDFE_NATIVE_FLOAT, HDFE_NOMERGE)
status = gddefcomp(gridid, HDFE_COMP_NONE, compparm)
status = gdeffld(gridid, "Temperature", "YDim,XDim",
H5T_NATIVE_FLOAT, HDFE_AUTOMERGE)
```
Define a New Dimension Within a Grid

GDdefdim

herr_t GDdefdim(hid_t gridID, char *dimname, int32_t dim)

- **gridID** IN: Grid id returned by GDcreate or GDattach
- **dimname** IN: Name of dimension to be defined
- **dim** IN: The size of the dimension

**Purpose**
Defines a new dimension within the grid.

**Return value**
Returns SUCCEED(0) if successful 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 = GDdefdim(gridID, "Band", 15)
```

To specify an unlimited dimension which can be used to define an appendable array, the dimension value should be set to H5S_UNLIMITED:

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

**FORTRAN**

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

The equivalent FORTRAN code for the example above is:

```fortran
parameter (UNLIMITED=0)
status = gddefdim(gridid, "Band", 15)
status = gddefdim(gridid, "Unlim", UNLIMITED)
```
Define a New Data Field Within a Grid

GDdeffield

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

gridID IN: Grid id returned by GDcreate or GDattach
fieldname IN: Name of field to be defined
dimlist IN: The list of data dimensions defining the field
maxdimlist IN: The list of data maximum dimensions defining the field
numbertype IN: The data type of the data stored in the field
merge IN: Merge code (HDFE-NOMERGE (0) - no merge, HDFE_AUTOMERGE (1) -merge)

Purpose Defines a new data field within the grid.

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

Description This routine defines data fields to be stored in the grid. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in C order, that is, the last dimension is incremented first. The API will attempt to merge into a single object those fields that share dimensions and in case of multidimensional fields, numbertype. Two and three dimensional fields will be merged into a single three-dimensional object if the last two dimensions (in C order are equal). If the merge code for a field is set to 0, the API will not attempt to merge it with other fields. Fields using the unlimited dimension will not be merged. Because merging breaks the one-to-one correspondence between HDF-EOS fields and HDF SDS arrays, it should not be set if the user wishes to access the HDF-EOS field directly using HDF routines or, for example, to create an HDF attribute corresponding to the field. If maximum dimensions are the same as input dimensions, the “NULL” should be passed as a fourth parameter. In case of unlimited dimension (appendable field) the corresponding dimension should be defined by calling GDdefdim() with the H5S_UNLIMITED as third parameter.

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

\[
\text{status} = \text{GDdeffield(gridID, "Temperature", "YDim,XDim", NULL, H5T_NATIVE_FLOAT, HDFE_AUTOMERGE);}
\]

\[
\text{status} = \text{GDdeffield(gridID, "Spectra", "Bands,YDim,XDim", NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);}
\]

**FORTRAN**

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

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

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

\[
\text{status} = \text{gddeffld(gridid, "Temperature", "XDim,YDim", NULL, HDFE_NATIVE_FLOAT, HDFE_AUTOMERGE)}
\]

\[
\text{status} = \text{gddeffld(gridid, "Spectra", "XDim,YDim,Bands", "", HDFE_NATIVE_FLOAT, HDFE_NOMERGE)}
\]

The dimensions are entered in *FORTRAN* order with the first dimension incremented first.
Define the Origin of the Grid Data

GDdeforigin

herr_t GDdeforigin(hid_t gridID, int32_t origincode)

(gridID) IN: Grid id returned by GDcreate or GDattach

(origincode) IN: Location of the origin of the grid data

Purpose Defines the origin of the grid data

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

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

Origin Codes:

HDFE_GD_UL(Default)(0) Upper Left corner of grid
HDFE_GD_UR(1) Upper Right corner of grid
HDFE_GD_LL(2) Lower Left corner of grid
HDFE_GD_LR(3) Lower Right corner of grid

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

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

FORTRAN integer function gddeforg(gridid, origincode)

integer            gridid

integer*4           origincode

The equivalent FORTRAN code for the above example is:

```fortran
parameter (HDFE_GD_LR=3)
status = gddeforg(gridid, HDFE_GD_LR)
```

Define a Pixel Registration Within a Grid

**GDdefpixreg**

```
herr_t GDdefpixreg(hid_t gridID, int32_t pixreg)
```

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

**pixreg**
IN: Pixel registration

**Purpose**
Defines pixel registration within grid cell

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

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

**Registration Codes:**
- HDFE_CENTER (0) (Default) Center of pixel cell
- HDFE_CORNER (1) Corner of a pixel cell

**Example**
In this example, we define the pixel registration to be the corner of the pixel cell:
```
status = GDdefpixreg(gridID, HDFE_CORNER);
```

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

integer    gridid
integer*4   pixreg

parameter (HDFE_CORNER=1)
status = gddefpreg(gridid, HDFE_CORNER)
```

The equivalent FORTRAN code for the example above is:
```
parameter (HDFE_CORNER=1)
status = gddefpreg(gridid, HDFE_CORNER)
```
Define Grid Projection

**GDdefproj**

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

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

**projcode**  IN: GCTP projection code

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

**spherecode**  IN: GCTP spheroid code

**projparm**  IN: GCTP projection parameter array

**Purpose** Defines projection of grid

**Return Value** Returns SUCCEED(0) if successful 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 = GDdefproj(gridID, GCTP_UTM, zonecode, spherecode, NULL);
```

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

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

Finally we define a Geographic projection. In this case neither the zone code, sphere code or the projection parameters are used.
status = GDdefproj(gridID, GCTP_GEO, NULL, NULL, NULL)

FORTRAN

```fortran
integer function gddefproj(gridid, projcode, zonecode, spherecode, projparm)

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

The equivalent FORTRAN code for the examples above is:

parameter (GCTP_UTM=1)
spherecode = 0
zonecode = 40
status = gddefproj(gridid, GCTP_UTM, zonecode, spherecode, dummy)

parameter (GCTP_PS=6)
spherecode = 6
do i=1,13
    projparm(i) = 0
enddo
projparm(6) = 90000000.00
status = gddefproj(gridid, GCTP_PS, dummy, spherecode, projparm)

parameter (GCTP_GEO=0)
status = gddefproj(gridid, GCTP_GEO, dummy, dummy, dummy)
```

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

**GDdeftile**

```c
herr_t GDdeftile(hid_t gridID, int32_t tilecode, int32_t tilerank, int32_t tiledims[])
```

- **gridID** IN: Grid id returned by GDcreate or GDattach
- **tilecode** IN: Tile code: HDF TILE, HDF NOTILE (default)
- **tilerank** IN: The number of tile dimensions
- **tiledims** IN: Tile dimensions

**Purpose** Defines tiling dimensions for subsequent field definitions

**Return Value** Returns SUCCEED(0) if successful 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 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 = GDdeftile(gridID, HDFE TILE, 2, tiledims);
status = GDdeffield(gridID, "Pressure", "YDim,XDim", NULL, H5T_NATIVE_INT, HDFE_NOMERGE);
status = GDdeffield(gridID, "Temperature", "YDim,XDim", NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);
tiledims[0] = 1;
tiledims[1] = 150;
tiledims[2] = 100;
status = GDdeftile(gridID, HDFE TILE, 3, tiledims);
```
status = GDdeffield(gridID, "Spectra", "Bands,YDim,XDim", NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);
status = GDdeftile(gridID, HDFE_NOTILE, 0, NULL);
status = GDdeffield(gridID, "Communities", "YDim,XDim", NULL, H5T_NATIVE_INT, HDFE_AUTOMERGE);

FORTRAN

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

The equivalent FORTRAN code for the example above is:

parameter (HDFE_NOTILE=0)
parameter (HDFE_NATIVE_INT = 0)
parameter (HDFE_TILE=1)
tiledims(1) = 200
tiledims(2) = 100
status = gddeftile(gridid, HDFE_TILE, 2, tiledims)
status = gddefld(gridid, 'Pressure', 'XDim,YDim', " ", HDFE_NATIVE_INT, HDFE_NOMERGE)
status = gddefld(gridid, 'Temperature', 'XDim,YDim', " ", HDFE_NATIVE_FLOAT, HDFE_NOMERGE)
tiledims[1] = 100
tiledims[2] = 150
tiledims[3] = 1
status = gddeftile(gridid, HDFE_TILE, 3, tiledims)
status = gddefld(gridid, 'Spectra', 'XDim,YDim,Bands', " ", HDFE_NATIVE_FLOAT, HDFE_NOMERGE)
status = gddeftile(gridid, HDFE_NOTILE, 0, tiledims);
status = gddefld(gridid, 'Communities', 'XDim,YDim', " ", HDFE_NATIVE_INT, HDFE_AUTOMERGE)
Define a Time Period of Interest

GDdeftimeperiod

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

gridID IN: Grid id returned by GDcreate or GDattach

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

starttime IN: Start time of period

stoptime IN: Stop time of period

Purpose Defines a time period for a grid.

Return value Returns the grid period ID if successful 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 GDextractperiod routine to read all the entries of a data field within the time period. The grid structure must have the Time field defined. This routine may be called after GDdefboxregion to provide both geographic and time subsetting. In this case the user provides the id from the previous subset call. (This same id is then returned by the function.) Furthermore it can be called before or after GDdefvrtregion to further refine a region. This routine may also be called “stand-alone” by setting the input id to HDFE_NOPREVSUB (-1).

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

starttime = 35232487.2;

stoptime = 36609898.1;

periodID = GDdeftimeperiod(gridID, HDFE_NOPREVSUB
starttime, stoptime);

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

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

Note that periodID will have the same value as regionID.
integer function gddeftmeper(gridid, periodID, starttime, stoptime)

integer gridid
integer periodid
real*8 starttime
real*8 stoptime

The equivalent FORTRAN code for the examples above are:

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

GDdefvrtregion

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

gridID IN: Grid id returned by GDcreate or GDattach
regionID IN: Region (or period) id from previous subset call
vertObj IN: Dimension or field to subset
range IN: Minimum and maximum range for subset

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

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

Description Whereas the GDdefboxregion routine subsets along the XDim and YDim dimensions, this routine allows the user to subset along any other dimension. The region is specified by a set of minimum and maximum values and can represent either a dimension index (case 1) or field value range (case 2). In the second case, the field must be one-dimensional and the values must be monotonic (strictly increasing or decreasing) in order that the resulting dimension index range be contiguous. (For the current version of this routine, the second option is restricted to fields with number type: INT16, INT32, FLOAT32, FLOAT64.) This routine may be called after GDdefboxregion to provide both geographic and “vertical” subsetting. In this case the user provides the id from the previous subset call. (This same id is then returned by the function.) This routine may also be called “stand-alone” by setting the input id to HDFE_NOPREVSUB (-1).

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

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

Example Suppose we have a field called Pressure of dimension Height (= 10) whose values increase from 100 to 1000. If we desire all the elements with values between 500 and 800, we make the call:
range[0] = 500.;
range[1] = 800.;
regionID = GDdefvrtrg(gridID, HDFE_NOPREVSUB, "Pressure", range);

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

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

range[0] = 2;
range[1] = 5;
regionID = GDdefvrtrg(gridID, HDFE_NOPREVSUB, "DIM:Height", range);

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

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

regionID = GDdefvrtrg(gridID, subsetID, "Pressure", range);

The return value, \textit{regionID} is set equal to \textit{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 \textit{Height} dimension.

\textbf{FORTRAN} \hspace{1cm} integer \hspace{1cm} function gddefvrtrg(gridid, regionid, vertobj, range)

integer \hspace{1cm} gridid
integer \hspace{1cm} regionid
character*(*) \hspace{1cm} vertobj
real*8 \hspace{1cm} range

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

parameter (HDFE_NOPREVSUB=-1)
range(1) = 500.
range(2) = 800.
regionID = gddefvrtrg(gridid, HDFE_NOPREVSUB, "Pressure", range)
range(1) = 3 \hspace{1cm} ! Note 1-based element numbers
range(2) = 6
regionid = gddefvrtreg(gridid, HDFE_NOPREVSUB, "DIM:Height", range)
regionid = gddefvrtreg(gridid, subsetid, "Pressure", range)
Detach from Grid Structure

**GDdetach**

herr_t GDdetach(hid_t gridID)

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

- **Purpose**: Detaches from grid interface.

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

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

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

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

**FORTRAN**

integer function gddetach(gridid)

integer gridid

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

```fortran
status = gddetach(gridid)
```
Retrieve Size of Specified Dimension

**GDdiminfo**

`hsize_t GDdiminfo(hid_t gridID, char *dimname)`

- **gridID** IN: Grid id returned by GDcreate or GDattach
- **dimname** IN: Dimension name

**Purpose** Retrieve size of specified dimension.

**Return value** Size of dimension if successful or FAIL(-1) otherwise. A typical reason for failure is an improper grid id or dimension name.

**Description** This routine retrieves the size of specified dimension.

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

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

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

**FORTRAN**

```fortran
integer*4 function gddiminfo(gridid,dimname)  
integer      gridid  
character(*)  dimname  
```

The equivalent FORTRAN code for the example above is:

```fortran
    dimsize = gddiminfo(gridid, "Bands")
```
Duplicate a Region or Period

GDdupregion

hid_t GDdupregion(hid_t regionID)

regionID IN: Region or period id returned by GDdefboxregion, GDdeftimeperiod, or 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 GDdefboxregion, duplicate the region creating a new region ID, regionID2, and then perform two different vertical subsets of these (identical) geographic subset regions:

regionID = GDdefboxregion(gridID, cornerlon, cornerlat);
regionID2 = GDdupregion(regionID);
regionID = GDdefvrtregion(gridID, regionID, "Pressure", rangePres);
regionID2 = GDdefvrtregion(gridID, regionID2, "Temperature", rangeTemp);

FORTRAN integer gddupreg(regionid)
integer regionid

The equivalent FORTRAN code for the example above is:

regionid = gddefboxreg(gridid, cornerlon, cornerlat)
regionid2 = gddupreg(regionid)
regionid = gddefvrtreg(gridid, regionid, 'Pressure', rangePres)
regionid2 = gddefvrtreg(gridid, regionid2, 'Temperature', rangeTemp)
Read a Region of Interest from a Field

GDextractregion

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

.gridID IN: Grid id returned by GDcreate or GDattach

.regionID IN: Region (period) id returned by GDdefboxregion (GDdeftimeperiod)

.fieldname IN: Field to subset

.buffer OUT: Data Buffer

Purpose Extracts (reads) from subsetted region.

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

Description This routine reads data into the data buffer from a subsetted region as defined by GDdefboxregion.

Example In this example, we extract data from the “Temperature” field from the region defined in GDdefboxregion. We first allocate space for the data buffer. The size of the subsetted region for the field is given by the Gdregioninfo routine.

datbuf = (float32) calloc(size, 4);

status = GDextractregion(GDid, regionID, "Temperature", datbuf32);

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

The equivalent FORTRAN code for the example above is:

status = gdextreg(gridid, regionid, "Temperature", datbuf)
Retrieve Information About Data Field in a Grid

GDfieldinfo

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

gridID      IN:  Grid id returned by GDcreate or GDattach
fieldname    IN:  Fieldname
rank         OUT:  Pointer to rank of the field
dims         OUT:  Array containing the dimension sizes of the field
numbertype   OUT:  Data type class ID of the field
dimlist      OUT:  Dimension list

Purpose     Retrieve information about a specific geolocation or data field in the grid.
Return value Returns SUCCEED(0) if successful 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 = GDfieldinfo(gridID, "Spectra", &rank, dims, &numbertype, dimlist);

The return parameters will have the following values:

rank=3, numbertype= 1, dims[3]={15,200,120} and

dimlist="Bands,YDim,XDim"
FORTRAN  integer function gdfldinfo(gridid, fieldname, rank, dims, numbertype, dimlist)

ingeger gridid
character*(*) fieldname
integer*4 rank
integer*4 dims(*)
integer numbertype
character*(*) dimlist

The equivalent FORTRAN code for the example above is:

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

The return parameters will have the following values:

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

dimlist="XDim,YDim,Bands"

Note that the dimensions array and the dimension list are in FORTRAN order.
Get Fill Value for Specified Field

GDgetfillvalue

herr_t GDgetfillvalue(hid_t gridID, char *fieldname, void *fillvalue)

gridID IN: Grid id returned by GDcreate or GDattach
fieldname IN: Fieldname
fillvalue OUT: Space allocated to store the fill value

Purpose Retrieves fill value for the specified field.

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

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

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

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

FORTRAN

integer function gdgetfill(gridid,fieldname,fillvalue)

integer gridid
classer*(*) Fieldname
<valid type> fillvalue(*)

The equivalent FORTRAN code for the example above is:

status = gdgetfill(gridid, "Temperature", tempfill)
Get Row/Columns for Specified Longitude/Latitude Pairs

GDgetpixels

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

gridID IN: Grid id returned by GDcreate or GDattach
nLonLat IN: Number of longitude/latitude pairs
lonVal IN: Longitude values in degrees
latVal IN: Latitude values in degrees
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 GDdefboxregion.

Example To convert two pairs of longitude/latitude values to rows and columns, make the following call:

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

The row and column of the two pairs will be returned in the rowArr and colArr arrays.
FORTRAN  integer function gdgetpix(gridid, nlonlat, lonval, latval, pixrow, pixcol)

  integer  gridid
  integer*4  nlonlat
  real*8  lonval
  real*8  latval
  integer*4  pixrow(*)
  integer*4  pixcol(*)

The equivalent FORTRAN code for the example above is:

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

Note that the row and columns values will be 1 - based.
Return Information About a Grid Structure

GDgridinfo

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

gridID IN: Grid id returned by GDcreate or GDattach

xdimsize OUT: Number of columns in grid

ydimsize OUT: Number of rows in grid

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

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

Purpose Returns position and size of grid

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

Description This routine returns the number of rows, columns and the location, in meters, of the upper left and lower right corners of the grid image.

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

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

FORTRAN integer function gdgridinfo(gridid, xdimsize, ydimsize, upleft, lowright)

integer gridid

integer*4 xdimsize

integer*4 ydimsize

real*8 upleft(*)

real*8 lowright(*)

The equivalent FORTRAN code for the example above is:

status = gdgridinfo(gridid, xdimsize, ydimsize, upleft, lowrgt);
Retrieve Information About Grid Attributes

GDinqattrs

int32_t GDinqattrs(hid_t gridID, char *attrlist, int32 *strbufsize)

gridID IN: Grid id returned by GDcreate or GDattach
attrlist OUT: Attribute list (entries separated by commas)
strbufsize OUT: String length of attribute list
Purpose Retrieve information about attributes defined in grid.
Return value Number of attributes found 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 grid structure. We assume that there are two attributes stored, attrOne and attr_2:
nattr = GDinqattrs(gridID, NULL, strbufsize);
The parameter, nattr, will have the value 2 and strbufsize will have value 14.
nattr = GDinqattrs(gridID, attrlist, strbufsize);
The variable, attrlist, will be set to: "attrOne,attr_2".

FORTRAN integer*4 function gdinqattrs(gridid,attrlist,strbufsize)
integer gridid
character(*) attrlist
integer*4 strbufsize
The equivalent FORTRAN code for the example above is:
nattr = gdinqattrs(gridid, attrlist, strbufsize)
Retrieve Information About Data Type of a Data Field Defined in Grid

GDinqdatatype

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

gridID IN: Grid id returned by GDcreate or GDattach
fieldname IN:: String containing the name of a field
fieldgroup IN: Group (Data Field ) the data field belongs to
datatype OUT: Data type ID of a data field dataset
classid OUT: Data type class ID
order OUT: Byte order of an atomic datatype
size OUT: Size of a datatype (in bytes)

Purpose Retrieve information about the data type of a specified data field.

Return value Status variable (0 if successful or -1 otherwise).

Description

Example

FORTRAN
Retrieve Information About Dimensions Defined in Grid

**GDinqdims**

```c
int32_t GDinqdims(hid_t gridID, char *dimname, int32_t dims[])
```

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

- **dimname**
  - **OUT:** Dimension list (entries separated by commas)

- **dims**
  - **OUT:** Array containing size of each dimension

**Purpose**
Retrieve information about dimensions defined in grid.

**Return value**
Number of dimension entries found if 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 = GDinqdims(gridID, dimname, dims);
```

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

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

**FORTRAN**

```fortran
integer*4 function gdinqdims(gridid, dimname, dims)
  integer      gridid
  character(*) dimname
  integer*4   dims(*)
```

The equivalent FORTRAN code for the example above is:

```fortran
ndim = gdinqdims(gridid, dimname, dims)
```
Retrieve Information About Data Fields Defined in Grid

GDinqfields

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

- **gridID**: IN: Grid id returned by GDcreate or GDattach
- **fieldlist**: OUT: Listing of data fields (entries separated by commas)
- **rank**: OUT: Array containing the rank of each data field
- **numbertype**: OUT: Array containing data type class ID of each data field

**Purpose**: Retrieve information about the data fields defined in grid.

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

**Description**: The field list is returned as a string with each data field separated by commas. The `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:

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

The parameter, `fieldlist`, will have the value: "Temperature,Spectra" with `rank[2]={2,3}`, `numbertype[2]={1,1}`

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

```fortran
integer gridid
character(*) fieldlist
integer*4 rank(*)
integer numbertype(*)
```

The equivalent FORTRAN code for the example above is:

```fortran
nfld = gdinqdflds(gridID, fieldlist, rank, numbertype)
```

The parameter, `fieldlist`, will have the value: "Spectra,Temperature" with `rank[2]={3,2}`, `numbertype[2]={1,1}`
Retrieve Grid Structures Defined in HDF-EOS File

GDinqgrid

int32_t GDinqgrid(char * filename, char *gridlist, int32_t *strbufsize)

filename  IN:  HDF-EOS filename
gridlist  OUT:  Grid list (entries separated by commas)
strbufsize  OUT:  String length of grid list

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

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

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

Example  In this example, we retrieve information about the grids defined in an HDF-EOS file, HDFEOS.h5. We assume that there are two grids stored, GridOne and Grid_2:

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

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

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

The variable, gridlist, will be set to:

“GridOne,Grid_2”.

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

character(*) filename
character(*) gridlist
integer*4 strbufsize

The equivalent FORTRAN code for the example above is:

ngrid = gdinqgrid('HDFEOS.h5', gridlist, strbufsize)
Return Number of Specified Objects in a Grid

GDnentries

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

gridID IN: Grid id returned by GDcreate or GDattach

entrycode IN: Entrycode

strbufsize OUT: String buffer size

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

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

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

The entry codes are: HDFE_NENTDIM (0) - Dimensions

HDFE_NENTDFLD (4) - Data Fields

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

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

FORTRAN integer*4 function gdnentries(gridid,entrycode, bufsize)

integer gridid

integer*4 entrycode

integer*4 bufsize

The equivalent FORTRAN code for the example above is:

ndims = gdnentries(gridid, 4, bufsize)
Open HDF-EOS File

GDopen

hid_t GDopen(char *filename, uintn access)

filename IN: Complete path and filename for the file to be opened
access IN: H5F_ACC_RDONLY, H5F_ACC_RDWR, H5F_ACC_TRUNC,
H5F_ACC_EXCL, H5F_ACC_DEBUG, H5P_DEFAULT

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

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

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

Access codes:

H5F_ACC_RDONLY Open for read only. If file does not exist, error
H5F_ACC_RDWR Open for read/write. If file does not exist, error
H5F_ACC_TRUNC If file exists, delete it, then open a new file for read/write
H5F_ACC_EXCL Fail if file already exists
H5F_ACC_DEBUG Print debug information
H5P_DEFAULT Apply default file access and creation properties

Example In this example, we create a new grid file named, GridFile.h5. It returns the file handle, fid.

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

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

The access codes should be defined as parameters:
parameter (DFACC_RDONLY=0)
parameter (DFACC_RDWR=1)
parameter (DFACC_TRUNC=2)
The equivalent *FORTRAN* code for the example above is:

```fortran
fid = gdopen("GridFile.h5", DFACC_TRUNC)
```
Return Grid Origin Information

GDorigininfo

herr_t GDorigininfo(hid_t gridID, int32_t *origincode)

gridID IN: Grid id returned by GDcreate or GDattach
origincode IN: Origin code
Purpose Retrieve origin code.
Return value Origin code if successful or FAIL (-1) otherwise.
Description This routine retrieves the origin code.
Example In this example, we retrieve the origin code defined in GDdeforigin.

status = GDorigininfo(gridID, &origincode);

The return value, origincode, will be equal to 3

FORTRAN integer function gdorginfo(gridid,origincde)

integer gridid
integer*4 origincode

The equivalent FORTRAN code for the above example is:

status = gdorginfo(gridid, origincode)
Return Pixel Registration Information

GDpixreginfo

herr_t GDpixreginfo(hid_t gridID, int32_t *pixregcode)

gridID IN: Grid id returned by GDcreate or GDattach
pixregcode IN: Pixel registration code

Purpose Retrieve pixel registration code.

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

Description This routine retrieves the pixel registration code.

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

status = GDpixreginfo(gridID, &pixregcode);

The return value, pixregcode, will be equal to 1

FORTRAN integer function gdpreginfo(gridid,pixregcode)

integer gridid
integer*4 pixregcode

The equivalent FORTRAN code for the above example is :

status = gdpreginfo(gridid, pixregcode)
Retrieve Grid Projection Information

GDprojinfo

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

gridID IN: Grid id returned by GDcreate or GDattach
projcode OUT: GCTP projection code
zonecode OUT: GCTP zone code used by UTM projection
spherecode OUT: GCTP spheroid code
projparm OUT: GCTP projection parameter array

Purpose Retrieves projection information of grid
Return Value Returns SUCCEED(0) if successful or FAIL(-1) otherwise

Description Retrieves the GCTP projection code, zone code, spheroid code and the projection parameters of the grid

Example In this example, we are retrieving the projection information from a grid attached to with GDattached:

status = GDprojinfo(gridID, projcode, zonecode, spherecode, projparm);

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

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

The equivalent FORTRAN code for the example above is:

status = gdprojinfo(gridid, projcode, zonecode, spherecode, projparm)
Read Grid Attribute

GDreadattr

herr_t GDreadattr(hid_t gridID, char *attrname, void *datbuf)

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

*attrname* IN: Attribute name

*datbuf* OUT: Buffer allocated to hold attribute values

Purpose Reads attribute from a grid.

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

Description The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.

Example In this example, we read a single precision (32 bit) floating point attribute with the name "ScalarFloat":

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

FORTRAN integer function gdrdattr(gridid, attrname, datbuf)

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

The equivalent FORTRAN code for the example above is:

```fortran
status = gdrdattr(gridid, "ScalarFloat", f32)
```
GDreadfield

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

gridID IN: Grid id returned by GDcreate or 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 IN: Buffer to store the data read from the field

Purpose Reads data from a grid field.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are improper grid id 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 value for stride is 1 and is used if this parameter is set to NULL.

Example In this example, we read data from the 10th row (0-based) of the Temperature field.

float32 row[120];
hssize_t start[2]={10,1};
hsize_t edge[2]={1,120};

status = GDreadfield(gridID, "Temperature", start, NULL, edge, row);
integer function gdrdflg(gridid,fieldname,start,stride,edge,buffer)

integer gridid
classer*(*) fieldName
integer*4 start(*)
integer*4 stride(*)
integer*4 edge(*)
<valid type> buffer(*)

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

The equivalent FORTRAN code for the example above is:

real*4 row(2000)
integer*4 start(2), stride(2), edge(2)
start(1) = 10
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 2000
edge(2) = 1

status = gdrdflg(gridid, "Temperature", start, stride, edge, row);
Return Information About a Region

GDregioninfo

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

gridID    IN: Grid id returned by GDcreate or GDattach
regionID  IN: Region (period) id returned by GDdefboxregion
           (GDdeftimeperiod)
fieldname IN: Field to subset
ntype     OUT: Data type class ID 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 GDdefboxregion for the Temperature field. We use this to allocate space for data in the subsetted region.

status = GDregioninfo(GDid, regionID, "Temperature", &ntype,
       &rank, dims, &size, upleft, lowright);
FORTRAN  integer function gdreginfo(gridid, regionid, fieldname, ntype, rank, dims, size, upleftpt, lowrightpt)

integer      gridid
integer      regionid
character(*)  fieldname
integer      ntype
integer*4    rank
integer*4    dims(*)
integer*4    size
real*8       upleftpt(*)
real*8       lowrightpt(*)

The equivalent FORTRAN code for the example above is:

status = gdreginfo(gridid, regid, "Spectra", ntype, rank, dims, size, upleftpt, lowrightpt)
Set Fill Value for a Specified Field

GDsetfillvalue

herr_t GDsetfillvalue(hid_t  
gridID, char  
*fieldname, void *fillvalue)

gridID IN: Grid id returned by GDcreate or GDattach
fieldname IN: Fieldname
fillvalue IN: Pointer to the fill value to be used

Purpose Sets fill value for the specified field.
Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are an improper grid id or number type.
Description The fill value is placed in all elements of the field which have not been explicitly defined.
Example In this example, we set a fill value for the "Temperature" field:

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

FORTRAN integer function

gdsetfill(gridid,fieldname,fillvalue)

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

The equivalent FORTRAN code for the example above is:

status = gdsetfill(gridid, "Temperature", -999.0)
Write/Update Grid Attribute

GDwriteattr

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

gridID IN: Grid id returned by GDcreate or 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 attribute in a grid.

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

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

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

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

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

```c
f32 = 3.14159;
status = GDwriteattr(gridid, "ScalarFloat",
H5T_NATIVE_FLOAT, 1, &f32);
```
integer function gdwrattr(gridid, attrname, ntype, count, datbuf)

integer gridid
character(*) attrname
integer ntype
integer*4 count(*)
<valid type> datbuf(*)

The equivalent FORTRAN code for the first example above is:

parameter (DFNT_FLOAT32=5)
count(1) = 1
f32 = 3.14
status = gdwrattr(gridid, "ScalarFloat", DFNT_FLOAT32, count, f32)
Write Data to a Grid Field

GDwritefield

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

gridID IN: Grid id returned by GDcreate or 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

glade 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 value for stride is 1 and is used if this parameter is set to NULL. 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 GDdeftile for further information.

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

float32 temperature [200][120];
hsize start[2]=(0,0);
hsize edge[2]=(200,120);

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

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

float32 newrow[2000];
start[2]=(0,10);
edge[2]=(2000,1);
/* Define elements of newrow array */
status = GDwritefield(gridID, "Temperature", start, NULL, edge, newrow);

FORTRAN

integer function
gdwrfld(gridid,fieldname,start,stride,edge,data)

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

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

The equivalent FORTRAN code for the example above is:

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

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

real*4 newrow(2000)
integer*4 start(2), stride(2), edge(2)
start(1) = 10
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 2000
edge(2) = 1
status = gdwrfld(gridid, "Temperature", start, stride, edge, newrow)
2.1.3 HDF-EOS Utility Routines

This section contains an alphabetical list of the utility functions. The functions are alphabetized on their C-language names.
Convert Among Angular Units

**EHconvAng**

float64 EHconvAng(float64 \(inAngle\), intn \(code\))

\(inAngle\) \hspace{1cm} \text{IN: Input angle}

\(code\) \hspace{1cm} \text{IN: Conversion code}

Purpose Convert among various angular units.

Return value Returns angle in desired units if successful or FAIL (-1) otherwise.

Description This routine converts angles between three units, decimal degrees, radians, and packed degrees-minutes-seconds. In the later unit, an angle is expressed as an integral number of degrees and minutes and a float point value of seconds packed as a single float64 number as follows: DDDMMMMSSS.SS. The six conversion codes are: HDFE_RAD_DEG (0), HDFE_DEG_RAD (1), HDFE_DMS_DEG (2), HDFE_DEG_DMS (3), HDFE_RAD_DMS (0), and HDFE_DMS_RAD (1), where the first three letter code (RAD - radians, DEG - decimal degrees, DMS - packed degrees-minutes-seconds) corresponds to the input angle and the second to the desired output angular unit.

Example To convert 27.5 degrees to packed format:

\[
\begin{align*}
\text{inAng} &= 27.5; \\
\text{outAng} &= \text{EHconvAng}(\text{inAng}, \text{HDFE_DEG_DMS}); \\
\text{“outAng”} &\text{ will contain the value: 27030000.00.}
\end{align*}
\]

**FORTRAN**

real*8 function ehconvang(inangle, code)

real*8 \hspace{1cm} \text{inangle}

integer \hspace{1cm} \text{code}

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

\[
\begin{align*}
inangle &= 27.5 \\
\text{outangle} &= \text{ehconvang}(\text{inangle}, 3)
\end{align*}
\]
Get HDF-EOS Version String

EHgetversion

herr_t EHgetversion(hid_t *fid, char *version)

fid IN: File id returned by SWopen, GDopen, or PTopen.
version OUT: HDF-EOS version string
Purpose Get HDF-EOS version string.
Return value Returns SUCCEED(0) if successful 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 2.0) used to create the HDF-EOS file: “SwathFile.h5”:

char version[16];
fid = SWopen("SwathFile.h5", H5F_ACC_RDONLY);
status = EHgetversion(fid, version);
“version” will contain the string: “HDFEOS_3.alpha”.

FORTRAN integer function ehgetver(fid, version)
integer fid
character*(*) version
The equivalent FORTRAN code for the example above is:
character*16 version
fid = swopen("SwathFile.h5",1)
status = ehgetver(fid, version)
Get HDF File ids

EHidinfo

herr_t EHidinfo(hid_t fid, hid_t *HDFfid, hid_t *groupID)

\( \text{id} \) IN: File id returned by \textit{SWopen}, \textit{GDopen}, or \textit{PTopen}.

\( \text{HDF fid} \) OUT: HDF file ID (returned by \textit{EHopen})

\( \text{groupID} \) OUT: HDF group ID (returned by \textit{XXcreate})

Purpose Get HDF file IDs.

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

Description This routine returns the HDF file ids corresponding to the HDF-EOS file id returned by \textit{SWopen}, \textit{GDopen}, or \textit{PTopen}. These ids can then be used to create or access native HDF structure such as groups, datasets, or HDF attributes within an HDF-EOS file.

Example

FORTRAN integer function ehidinfo(fid,hdffid,sdid)

integer fid
integer hdffid
integer grid

To retrieve the HDF file id and Group interface id:

\( \text{fid} = \text{swopen(“SwathFile.h5”,1)} \)

\( \text{status} = \text{ehidinfo(fid, hdffid, grid)} \)
### Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI&amp;T</td>
<td>Algorithm Integration &amp; Test</td>
</tr>
<tr>
<td>AIRS</td>
<td>Atmospheric Infrared Sounder</td>
</tr>
<tr>
<td>API</td>
<td>application program interface</td>
</tr>
<tr>
<td>ASTER</td>
<td>Advanced Spaceborne Thermal Emission and Reflection Radiometer</td>
</tr>
<tr>
<td>CCSDS</td>
<td>Consultative Committee on Space Data Systems</td>
</tr>
<tr>
<td>CDRL</td>
<td>Contract Data Requirements List</td>
</tr>
<tr>
<td>CDS</td>
<td>CCSDS day segmented time code</td>
</tr>
<tr>
<td>CERES</td>
<td>Clouds and Earth Radiant Energy System</td>
</tr>
<tr>
<td>CM</td>
<td>configuration management</td>
</tr>
<tr>
<td>COTS</td>
<td>commercial off–the–shelf software</td>
</tr>
<tr>
<td>CUC</td>
<td>constant and unit conversions</td>
</tr>
<tr>
<td>CUC</td>
<td>CCSDS unsegmented time code</td>
</tr>
<tr>
<td>DAAC</td>
<td>distributed active archive center</td>
</tr>
<tr>
<td>DBMS</td>
<td>database management system</td>
</tr>
<tr>
<td>DCE</td>
<td>distributed computing environment</td>
</tr>
<tr>
<td>DCW</td>
<td>Digital Chart of the World</td>
</tr>
<tr>
<td>DEM</td>
<td>digital elevation model</td>
</tr>
<tr>
<td>DTM</td>
<td>digital terrain model</td>
</tr>
<tr>
<td>ECR</td>
<td>Earth centered rotating</td>
</tr>
<tr>
<td>ECS</td>
<td>EOSDIS Core System</td>
</tr>
<tr>
<td>EDC</td>
<td>Earth Resources Observation Systems (EROS) Data Center</td>
</tr>
<tr>
<td>EDHS</td>
<td>ECS Data Handling System</td>
</tr>
<tr>
<td>EDOS</td>
<td>EOSDIS Data and Operations System</td>
</tr>
<tr>
<td>EOS</td>
<td>Earth Observing System</td>
</tr>
<tr>
<td>EOSAM</td>
<td>EOS AM Project (morning spacecraft series)</td>
</tr>
<tr>
<td>EOSDIS</td>
<td>Earth Observing System Data and Information System</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>EOSPM</td>
<td>EOS PM Project (afternoon spacecraft series)</td>
</tr>
<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>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>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NGDC</td>
<td>National Geophysical Data Center</td>
</tr>
<tr>
<td>NMC</td>
<td>National Meteorological Center (NOAA)</td>
</tr>
<tr>
<td>ODL</td>
<td>object description language</td>
</tr>
<tr>
<td>PC</td>
<td>process control</td>
</tr>
<tr>
<td>PCF</td>
<td>process control file</td>
</tr>
<tr>
<td>PDPS</td>
<td>planning &amp; data production system</td>
</tr>
<tr>
<td>PGE</td>
<td>product generation executive (formerly product generation executable)</td>
</tr>
<tr>
<td>POSIX</td>
<td>Portable Operating System Interface for Computer Environments</td>
</tr>
<tr>
<td>PT</td>
<td>point</td>
</tr>
<tr>
<td>QA</td>
<td>quality assurance</td>
</tr>
<tr>
<td>RDBMS</td>
<td>relational data base management system</td>
</tr>
<tr>
<td>RPC</td>
<td>remote procedure call</td>
</tr>
<tr>
<td>RRDB</td>
<td>recommended requirements database</td>
</tr>
<tr>
<td>SCF</td>
<td>Science Computing Facility</td>
</tr>
<tr>
<td>SDP</td>
<td>science data production</td>
</tr>
<tr>
<td>SDPF</td>
<td>science data processing facility</td>
</tr>
<tr>
<td>SGI</td>
<td>Silicon Graphics Incorporated</td>
</tr>
<tr>
<td>SMF</td>
<td>status message file</td>
</tr>
<tr>
<td>SMP</td>
<td>Symmetric Multi–Processing</td>
</tr>
<tr>
<td>SOM</td>
<td>Space Oblique Mercator</td>
</tr>
<tr>
<td>SPSO</td>
<td>Science Processing Support Office</td>
</tr>
<tr>
<td>SSM/I</td>
<td>Special Sensor for Microwave/Imaging</td>
</tr>
<tr>
<td>SW</td>
<td>swath</td>
</tr>
<tr>
<td>TAI</td>
<td>International Atomic Time</td>
</tr>
<tr>
<td>TBD</td>
<td>to be determined</td>
</tr>
<tr>
<td>TDRSS</td>
<td>Tracking and Data Relay Satellite System</td>
</tr>
<tr>
<td>TRMM</td>
<td>Tropical Rainfall Measuring Mission (joint US – Japan)</td>
</tr>
<tr>
<td>UARS</td>
<td>Upper Atmosphere Research Satellite</td>
</tr>
<tr>
<td>UCAR</td>
<td>University Corporation for Atmospheric Research</td>
</tr>
</tbody>
</table>

**Notes:**
- SPSO: Science Processing Support Office
- SSM/I: Special Sensor for Microwave/Imaging
- TAI: International Atomic Time
- TBD: to be determined
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>universal reference locator</td>
</tr>
<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>
</tbody>
</table>