HDF-EOS Interface Based on HDF5, Volume 1: Overview and Examples

Technical Paper

January 2012

Prepared Under Contract # NNG10HP02C

RESPONSIBLE ENGINEER

E. Moghaddam-Taaheri 1/17/2012
Abe Taaheri Date
EOSDIS Evolution and Development (EED) Contract

SUBMITTED BY

Derrick Clarke 1/23/2012
Derrick Clarke, Custom Code Maintenance Date
EOSDIS Evolution and Development (EED) Contract

Raytheon Company
Riverdale, Maryland
Preface

This document is a Users Guide for HDF-EOS (Hierarchical Data Format - Earth Observing System) library tools. The version described in this document is HDF-EOS Version 5.1.14. The software is based on HDF5, a new version of HDF provided by The HDF Group. HDF5 is a complete rewrite of the earlier HDF4 version, containing a different data model and user interface. HDF-EOS V5.1.14 incorporates HDF5, and keeps the familiar HDF4-based interface. There are a few exceptions and these exceptions are described in this document. Note that the major functional difference is that Version 5.1.14 of the HDF-EOS library is a thread-safe.

HDF is the scientific data format standard selected by NASA as the baseline standard for EOS. This Users Guide accompanies Version 5.1.14 software, which is available to the user community on the EDHS1 server. This library is aimed at EOS data producers and consumers, who will develop their data into increasingly higher order products. These products range from calibrated Level 1 to Level 4 model data. The primary use of the HDF-EOS library will be to create structures for associating geolocation data with their associated science data. This association is specified by producers through use of the supplied library. Most EOS data products which have been identified, fall into categories of point, grid, swath or zonal average structures, the latter two of which are implemented in the current version of the library. Services based on geolocation information will be built on HDF-EOS structures. Producers of products not covered by these structures, e.g. non-geolocated data, can use the standard HDF libraries.

In the ECS (EOS Core System) production system, the HDF-EOS library will be used in conjunction with SDP (Science Data Processing) Toolkit software. The primary tools used in conjunction with HDF-EOS library will be those for metadata handling, process control and status message handling. Metadata tools will be used to write ECS inventory and granule specific metadata into HDF-EOS files, while the process control tools will be used to access physical file handles used by the HDF tools (SDP Toolkit Users Guide for the EOSDIS Evolution and Development (EED) Contract, January 2012, 333-EED-001, Revision 01.)

HDF-EOS is an extension of The HDF Group (THG) HDF5 and uses HDF5 library calls as an underlying basis. Version 5-1.8.8 of HDF5 is used. The library tools are written in the C language and a FORTRAN interface is provided. The current version contains software for creating, accessing and manipulating Grid, Point, Swath and Zonal Average structures. This document includes overviews of the interfaces, and code examples. The HDF-EOS plug-in for HDFView, the HDF-EOS viewing tool, will be revised to accommodate the current version of the library.

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

   Abe Taaheri, Abe_Taaheri@raytheon.com

An email address has been provided for user help:

   Landover_PGSTLKIT@raytheon.com

Any questions should be addressed to:

Data Management Office
EOSDIS Evolution and Development (EED) Contract
Raytheon Company
5700 Rivertech Court
Riverdale, MD 20737
Abstract

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

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

### Preface

### Abstract

### 1. Introduction

1.1 Identification ................................................................. 1-1  
1.2 Scope ................................................................. 1-1  
1.3 Purpose and Objectives ................................................................. 1-1  
1.4 Status and Schedule ................................................................. 1-1  
1.5 Document Organization ................................................................. 1-2  

### 2. Related Documentation

2.1 Parent Documents ................................................................. 2-1  
2.2 Related Documents ................................................................. 2-1  

### 3. Overview of HDF-EOS

3.1 Background ................................................................. 3-1  
3.2 Design Philosophy ................................................................. 3-1  
3.3 Packaging ................................................................. 3-2  
3.4 Operations Concept for the HDF5 Based Library ................................................................. 3-2  
3.5 Differences Between HDF-EOS V2.18 (HDF4 based) and HDF-EOS V5.1.14 (HDF5 based) ................................................................. 3-3  
3.6 Different Types of Attributes in HDF-EOS5 ................................................................. 3-6  

vii
4. Point Data

4.1 Introduction ................................................................................................................... 4-1
4.2 Applicability .................................................................................................................. 4-3
4.3 The Point Data Interface .............................................................................................. 4-3
  4.3.1 PT API Routines ...................................................................................................... 4-3
  4.3.2 File Identifiers ......................................................................................................... 4-5
  4.3.3 Point Identifiers ....................................................................................................... 4-5
4.4 Programming Model ..................................................................................................... 4-5

5. Swath Data

5.1 Introduction ................................................................................................................... 5-1
  5.1.1 Data Fields ................................................................................................................. 5-2
  5.1.2 Geolocation Fields .................................................................................................... 5-3
  5.1.3 Dimensions .............................................................................................................. 5-3
  5.1.4 Dimension Maps ...................................................................................................... 5-3
  5.1.5 Index .......................................................................................................................... 5-5
5.2 Applicability .................................................................................................................. 5-5
5.3 The Swath Data Interface ............................................................................................. 5-5
  5.3.1 SW API Routines .................................................................................................... 5-5
  5.3.2 File Identifiers ......................................................................................................... 5-8
  5.3.3 Swath Identifiers ...................................................................................................... 5-8
5.4 Programming Model ..................................................................................................... 5-8

6. Grid Data

6.1 Introduction ................................................................................................................... 6-1
  6.1.1 Data Fields ................................................................................................................. 6-2
  6.1.2 Dimensions .............................................................................................................. 6-2
  6.1.3 Projections ............................................................................................................... 6-2
6.2 Applicability .................................................................................................................. 6-3
6.3 The Grid Data Interface ............................................................................................... 6-3
6.3.1 GD API Routines .......................................................................................... 6-3
6.3.2 File Identifiers ............................................................................................ 6-5
6.3.3 Grid Identifiers ........................................................................................... 6-6
6.4 Programming Model ....................................................................................... 6-6
6.5 GCTP Usage ................................................................................................... 6-7
6.5.1 GCTP Projection Codes .............................................................................. 6-7
6.5.2 UTM Zone Codes ......................................................................................... 6-8
6.5.3 GCTP Spheroid Codes ............................................................................... 6-10
6.5.4 Projection Parameters ................................................................................ 6-11

7. Examples of HDF-EOS Library Usage

7.1 Point Examples ............................................................................................... 7-1
  7.1.1 Creating a Simple Point ............................................................................ 7-1
7.2 Swath Examples .............................................................................................. 7-38
  7.2.1 Creating a Simple Swath .......................................................................... 7-38
7.3 Grid Examples .................................................................................................. 7-68
  7.3.1 Creating a Simple Grid ............................................................................ 7-69
7.4 Zonal Average Examples ................................................................................ 7-94
  7.4.1 Creating a Simple Zonal Average .............................................................. 7-94

8. Writing ODL Metadata into HDF-EOS

8.1 Introduction ..................................................................................................... 8-1
8.2 Coding Examples of Metadata Writes to HDF-EOS Files .............................. 8-2
  8.2.1 C Code for MTD TOOLKIT Users .............................................................. 8-2
  8.2.2 C Code for SDP TOOLKIT Users .............................................................. 8-8
8.3 The Metadata Configuration File (MCF) for Codes in Section 8.2 ............... 8-15
8.4 The ODL Output File Which Results from Running Code in Section 8.2 .......... 8-22
8.5 The files filetable.temp and PCF file ........................................................... 8-31
  8.5.1 The file filetable.temp used for example in Section 8.2.1 ....................... 8-32
  8.5.2 The PCF file used for example in Section 8.2.2 ....................................... 8-32
9. Zonal Average Data

9.1 Introduction .................................................................................................................. 9-1
  9.1.1 Data Fields .............................................................................................................. 9-1
  9.1.2 Dimensions ............................................................................................................ 9-1

9.2 Applicability ............................................................................................................... 9-1

9.3 The Zonal Average Data Interface ............................................................................. 9-1
  9.3.1 ZA API Routines .................................................................................................... 9-2
  9.3.2 File Identifiers ....................................................................................................... 9-3
  9.3.3 Zonal Average Identifiers ...................................................................................... 9-3

9.4 Programming Model .................................................................................................. 9-3

List of Figures

4-1 A Simple Point Data Example .................................................................................... 4-1
4-2 Recording Points Over Time ....................................................................................... 4-2
4-3 Point Data from a Moving Platform ........................................................................... 4-2
5-1 A Typical Satellite Swath: Scanning Instrument ........................................................ 5-1
5-2 A Swath Derived from a Profiling Instrument ............................................................. 5-2
5-3 A “Normal” Dimension Map ..................................................................................... 5-4
5-4 A “Backwards” Dimension Map ................................................................................ 5-4
6-1 A Data Field in a Mercator-projected Grid ................................................................. 6-1
6-2 A Data Field in an Interrupted Goode’s Homolosine-Projected Grid ....................... 6-2

List of Tables

3-1 HDF-EOS 5.1.14 Current Modifications to the HDF4-Based HDF-EOS Library 2.18 3-4
4-1 Summary of the Point Interface .................................................................................. 4-4
5-1 Summary of the Swath Interface ................................................................................. 5-6
6-1 Summary of the Grid Interface ................................................................................... 6-4
6-2 Projection Transformation Package Projection Parameters ....................................... 6-11
6-3 Projection Transformation Package Projection Parameters Elements ....................... 6-12
9-1 Summary of the Zonal Average Interface .................................................................... 9-2
Appendix A. Installation and Maintenance

Abbreviations and Acronyms
This page intentionally left blank.
1. Introduction

1.1 Identification
The HDF-EOS Interface Based on HDF5, Volume 1: Overview and Examples was prepared under the EOSDIS Evolution and Development (EED) Contract (NNG10HP02C).

1.2 Scope
This document is intended for use by anyone who wishes to write software to create or read EOS data products. Users of this document will likely 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.

1.3 Purpose and Objectives
This document will serve as a user’s guide for the HDF-EOS file access library developed for ECS. Upon reading this document, the reader should have a thorough understanding of each data model and corresponding programming interface provided as part of HDF-EOS. Specifically, this user’s guide contains an overview of each data model, a complete function-by-function reference for each software interface, and sample programs illustrating the basic features of each interface.

The reader should note that this paper will not discuss the HDF structures underlying HDF-EOS nor the specific conventions employed. For more information on HDF, its design philosophy, and its logical and physical formats, the reader is referred to the HDF5 documentation listed in Section 2.2 Applicable Documents. For more information on the conventions employed by HDF-EOS, the reader is referred to the various design White Papers listed in Section 2.2.

*Important Note*:
The FORTRAN-literate reader is cautioned that dimension ordering is row-major in C (last dimension varying fastest), whereas FORTRAN uses column-major ordering (first dimension varying fastest). Therefore, FORTRAN programmers should take care to use dimensions in the reverse order to that shown in the text of this document. (FORTRAN code examples are correct as written.)

1.4 Status and Schedule
December, 1999, Prototype HDF5 based Library Available
January, 2001, SCF version including both HDF4 and HDF5 support available
June, 2001, SCF version with additional FORTRAN support available
February, 2002, this SCF version was integrated into ECS
1.5 Document Organization

This document is organized as follows:

- Section 1 Introduction - Presents Scope and Purpose of this document
- Section 2 Related Documentation
- Section 3 Overview of HDF-EOS - Background and design features of the library
- Section 4 Point Data - Design features and listing of the HDF-EOS Point Library
- Section 5 Swath Data - Design features and listing of the HDF-EOS Swath Library
- Section 6 Grid Data - Design features and listing of the HDF-EOS Grid Library
- Section 7 Examples - A selection of programming examples
- Section 8 Examples of SDP Toolkit Usage - How to use the HDF-EOS Library in conjunction with the SDP Toolkit
- Section 9 Zonal Average Data - Design features and listing of the HDF-EOS Zonal Average Library
- Appendix A Installation Instructions, Test Drivers, User Feedback
- Acronyms

The accompanying Function Reference Guide is organized as follows:

- Section 1 Introduction
- Section 2 Reference - Specification of the HDF-EOS Point, Swath, Grid and Zonal Average
- APIs Function
- Acronyms
2. Related Documentation

2.1 Parent Documents

The following documents are the parents from which this document’s scope and content derive:

175-EED-001  HDF-EOS Interface Based on HDF5, Volume 1: Overview and Examples
175-EED-002  HDF-EOS Interface Based on HDF5, Volume 2: Function Reference Guide
456-TP-013  The HDF-EOS Design Document for the ECS Project
170-WP-002  Thoughts on HDF-EOS Metadata, A White Paper for the ECS Project
170-WP-003  The HDF-EOS Swath Concept, A White Paper for the ECS Project
170-WP-011  The HDF-EOS Grid Concept, A White Paper for the ECS Project

2.2 Related Documents

The following documents are referenced within this technical paper, or are directly applicable, or contain policies or other directive matters that are binding upon the content of this document.

170-EMD-003  A Data Formatting Toolkit for Extended Data Providers to NASA’s Earth Observing System Data and Information System (V5.0)
175-WP-001  An HDF-EOS and Data Formatting Primer for the ECS Project, A White Paper for the ECS Project

none  Introduction to HDF5 Release 1.2, University of Illinois, October, 1999
none Getting Started with HDF, Version 3.2, University of Illinois, May 1993
none An Album of Map Projections, USGS Professional Paper 1453, Snyder and Voxland, 1989
3. Overview of HDF-EOS

3.1 Background

The Hierarchical Data Format (HDF) has been selected by the EOSDIS Project as the format of choice for standard product distribution. HDF is a function library that was originally developed by The HDF Group (THG) at Urbana-Champaign to provide a portable storage mechanism for supercomputer simulation results. Although this user’s guide does not attempt to explain the inner workings of HDF5, a cursory knowledge of HDF5 may help the reader to understand the basic workings of HDF-EOS.

HDF5 files consist of a directory and a collection of data objects. Every data object has a directory entry, containing a pointer to the data object location, and information defining the datatype (much more information about HDF5 can be found in the HDF5 documentation referenced in Section 2.2 of this Guide). Unlike HDF4, there are only two fundamental data objects in HDF5. These objects are groups and dataspaces. The HDF4 data types such as vdatas and scientific data sets are mapped into the more general class of dataspaces.

To bridge the gap between the needs of EOS data products and the capabilities of HDF, three new EOS specific datatypes – point, swath, grid, and zonal average – have been defined within the HDF framework. Each of these new datatypes is constructed using conventions for combining standard HDF datatypes and is supported by a special application programming interface (API) which aids the data product user or producer in the application of the conventions. The APIs allow data products to be created and manipulated in ways appropriate to each datatype, without regard to the actual HDF objects and conventions underlying them.

The sum of these new APIs comprise the HDF-EOS library. The Point interface is designed to support data that has associated geolocation information, but is not organized in any well defined spatial or temporal way. The Swath 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). The Grid interface is designed to support data that has been stored in a rectilinear array based on a well defined and explicitly supported projection. The Zonal Average interface is designed to support data that has not associated with specific geolocation information.

3.2 Design Philosophy

Since the HDF-EOS library is intended to support end users of EOS data as well as EOS data producers, it is essential that HDF-EOS be available separately from other ECS software. For this reason, HDF-EOS does not rely on any other ECS software, including the SDP Toolkit. It is treated as an extension to the HDF5 library and, as such, it follows the general design philosophy and coding style of HDF5. For more information on the design of HDF5, please refer to the appropriate HDF5 documentation listed in Section 2.2.
3.3 Packaging

Because of the functional overlap of HDF, HDF-EOS, and the SDP Toolkit, it is important to understand what each one contains and how they are related. THE HDF GROUP HDF is a subroutine library freely available as source code from the The HDF Group (THG). The basic HDF library has its own documentation, and comes with a selection of simple utilities.

HDF-EOS is a higher level library available from the ECS project as an add-on to the basic HDF library. It requires THE HDF GROUP HDF for successful compiling and linking and will be widely available (at no charge) to all interested parties.

The SDP Toolkit is a large, complex library of functions for use by EOS data producers. It presents a standard interface to Distributed Active Archive Center (DAAC) services for data processing, job scheduling, and error handling. The Toolkit distribution includes source code for both HDF and HDF-EOS.

EOS instrument data producers will use the SDP Toolkit in conjunction with the HDF-EOS and HDF libraries. Of primary importance will be process control and metadata handling tools. The former will be used to access physical file handles required by the HDF library. The SDP Toolkit uses logical file handles to access data, while HDF (HDF-EOS) requires physical handles. Users will be required to make one additional call, using the SDP toolkit to access the physical handles. Please refer to the SDP Toolkit Users Guide for the EOSDIS Evolution and Development Contract, January 2012, 333-EED-001, Revision 01, Section 6.2.1.2 for an example). Section 7 of this document gives examples of HDF-EOS usage in conjunction with the SDP Toolkit.

Metadata tools will be used to access and write inventory and granule specific metadata into their designated HDF structures. Please refer to Section 6.2.1.4 of the SDP Toolkit Users Guide.

We make an important distinction between granule metadata and the structural metadata referred to in the software description below. Structural metadata specifies the internal HDF-EOS file structure and the relationship between geolocation data and the data itself. Structural metadata is created and then accessed by calling the HDF-EOS functions. Granule metadata will be used by ECS to perform archival services on the data. A copy will attached to HDF-EOS files by SDP toolkit calls and another copy is placed in the ECS archives. The two sets of metadata are not dynamically linked. However, the data producer should use consistent naming conventions when writing granule metadata when calling the HDF-EOS API. Please refer to the examples in Section 7, below.

3.4 Operations Concept for the HDF5 Based Library

HDF5 is a nearly complete rewrite of HDF4 and contains a different user API and underlying data model. An HDF-EOS library written in conjunction with HDF5 and which uses HDF5 functionality, will necessarily be a rewrite of the HDF4 - based version. The new HDF5 - based library will support the same Grid/Point/Swath/ZA functionality and to the extent possible and will be built with the same calling sequences as the original library. We will refer to the newer library as HDF-EOS 5.1.14. The former library is currently designated HDF-EOS 2.18. (The HDF-EOS Library Users Guide for the EED Contract, Volume 1 and Volume 2).
The following future uses for HDF-EOS are anticipated:

A. Product developers reading and writing HDF4-based files.

B. ECS subsystems reading and writing HDF4-based files.

C. Product developers reading and writing HDF5-based files. This will include both users retrofitting HDF4-based software and users starting from scratch with HDF5.

D. ECS subsystems reading and writing HDF5-based files.

E. Data migration applications, i.e. read HDF4 and write HDF5.

HDF-EOS 5.1.14 support for HDF5 will have the same function calls as the current HDF-EOS 2.18 support for HDF4. The names have been modified to add “HE5_” as a prefix. The parameters in the V5.0 function calls will be the same, to the extent possible. This implementation will entail the fewest possible modifications with respect to retrofitting code, i.e. removing HDF4 file access in favor of HDF5 file access. Users retrofitting old code, should anticipate changing variable types to accommodate new HDF5 variable types. The API will make the underlying HDF5 data model transparent otherwise.

Support will be provided for the current suite of UNIX and Windows operating systems. F77, F90, C and C++ interfaces will continue to be supported.

### 3.5 Differences Between HDF-EOS V2.18 (HDF4 based) and HDF-EOS V5.1.14 (HDF5 based)

There are several important differences between the Versions 2.18 and 5.1.14 of the HDF-EOS library that are briefly summarized in this section. An overview of modifications to the HDF4 based library is listed in Table 3-1.
<table>
<thead>
<tr>
<th>Function</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point Access</td>
<td>Modify to implement HDF5 Compound data type.</td>
</tr>
<tr>
<td>Point Definition</td>
<td>Modify to implement HDF5 Compound data type.</td>
</tr>
<tr>
<td>Point Input/output</td>
<td>Add an additional parameter, 'size', a dataset size. Add an additional parameter – data structure in HE5_Pthreadlevel() to pass information about data fields to read.</td>
</tr>
<tr>
<td>Swath Access</td>
<td>Modify to implement new HDF5 file structure. Create a swath with a specified name.</td>
</tr>
<tr>
<td>Swath Definition</td>
<td>Add an additional parameter, 'maxdimlist', a list of maximum sizes of dimensions. This will account for feature of appendibility of each dimension in a dataset. Add a new function for chunking in HDF5 (HE5_SWdefchunk())</td>
</tr>
<tr>
<td>Swath Inquiry</td>
<td>Add a new function HE5_SWchunkinfo() to retrieve chunking information. Add two new functions HE5_SWIngqfildalias() and HE5_SWingqfildalias() to retrieve information about data or geolocation fields and aliases in swath</td>
</tr>
<tr>
<td>Swath Input/output</td>
<td>Add unsigned long type to store large number of elements in attributes. Change 'start,stride,edge' parameter type to allow for very large numbers.</td>
</tr>
<tr>
<td>Swath Subset</td>
<td>Add a new function HE5_SWindexinfo() to retrieve the indices information about a subsetted region.</td>
</tr>
<tr>
<td>New interfaces</td>
<td>HE5_SWIngdatatype() retrieve information about data type of a data field HE5_PRdefine(),HE5_PRread(),HE5_PRwrite(),HE5_PRinquire(), HE5_PRinfo(),HE5_PRreclaimsment() - support profile structures. HE5_SWsetextdata(),HE5_SWgetextdata() – support external data files.</td>
</tr>
<tr>
<td>Grid Access</td>
<td>Modify to implement new HDF5 file structure. Create a grid with a specified name.</td>
</tr>
<tr>
<td>Grid Definition</td>
<td>HE5_GDdeffield() modified to add 'maxdimlist', a comma separated list of maximum dimensions - same as for Swath Modify functions to support HDF5 chunking: HE5_GDdeftile() and HE5_GDdefcompi() and HE5_GDdefcomti().</td>
</tr>
<tr>
<td>New interfaces</td>
<td>HE5_GDsetextdata(),HE5_GDgetextdata() – support external data files.</td>
</tr>
<tr>
<td>Grid Input/Output</td>
<td>Add unsigned long type to store large number of elements in attributes.</td>
</tr>
<tr>
<td>Grid Inquiry</td>
<td>Add a new function HE5_GDinqfildalias() to retrieve information about data fields and aliases in grid.</td>
</tr>
<tr>
<td>Grid Subset</td>
<td>No new functionality or interface changes</td>
</tr>
<tr>
<td>Grid Tiling</td>
<td>Add a new function HE5_GDtileinfo() to retrieve tiling information for fields.</td>
</tr>
<tr>
<td>Utilities</td>
<td>Added a function for determining whether a file is HDF4 or HDF5 based.  Added checking for illegal characters (&quot;,&quot;&quot;,&quot;&quot;,&quot;&quot; ) in the HDF-EOS object names. Added interfaces for the global &quot;File Attributes&quot;: HE5_EHwriteglbattr(),HE5_EHreadglbattr(), HE5_EHglbattrinfo(),HE5_EHinqglbattr(),HE5_EHinqglbdatatye().</td>
</tr>
<tr>
<td>HE5View</td>
<td>Modified EOS View to accommodate new HDF5-based HDF-EOS library.</td>
</tr>
</tbody>
</table>
Version 5.1.14 of the HDF-EOS library is a thread-safe.

Selected user considerations are listed below.

The routines HE5_SWopen(), HE5_SWcreate(), HE5_GDopen(), HE5_GDcreate() implement a new file structure designed for the V5.1.14 of the HDF-EOS library.

The field definition routines, HE5_SWdefgeofield(), HE5_SWdefdatafield(), and HE5_GDdeffield(), have an additional parameter 'maxdimlist', a coma separated list of maximum sizes of dimensions. The parameter 'maxdimlist' is reserved for future use. Since in HDF5 each dimension of a dataset can be appendable (extendible), the definition of a dataset should include the maximum size (or unlimited size) the corresponding dimension can be expanded to. Passing a NULL as a 'maxdimlist' means that the dimension is not appendable, and its maximum size is the same as its actual size. The unlimited dimension can be specified e.g. by a call to HE5_SWdefdim(sw_id, "Unlim", H5S_UNLIMITED). Then, in the call to e.g. HE5_SWdefdatafield() we should use for the 'maxdimlist' parameter the value "Unlim".

HDF5 requires the user to use chunking in order to define extendible datasets. Chunking makes it possible to extend datasets efficiently, without having to reorganize storage excessively. The corresponding (new) calls are HE5_SWdefchunk() and HE5_GDdeftile(). These calls should be used before the familiar old-library call to HE5_SWdefcomp() and HE5_GDdefcomp(), respectively. The latter set the field compression for all subsequent field definitions.

In the input/output routines HE5_PTwriteattr(), HE5_SWwriteattr() and HE5_GDwriteattr(), the data type of the fourth parameter, number of values to store in attribute, is now such that it allows to store arbitrary big number of elements in attribute. Also, the routines HE5_SWwritefield(), HE5_SWreadfield(), HE5_GDwritefield(), and HE5_GDreadfield() allow for the parameters 'start', 'stride', and 'edge' to use very big numbers.

There are four new inquiry routines HE5_EHinqglobdatatype(), HE5_PTinqdatatype(), HE5_SWinqdatatype() and HE5_GDinqdatatype() that, for a specified field, retrieve explicit information about data type, including number of bytes allocated for each element of the corresponding dataset.

There are two new inquiry routines HE5_SWinqdfldalias() and HE5_SWinqgfldalias() that retrieve information about the list of data or geolocation fields and aliases, and the length of the list.

A new data type HE5T_CHARSTRING for a character string is defined in the header file HE5_HdfEosDef.h.

The new routine HE5_SWchunkinfo() retrieves chunking information about a field.

The new routine HE5_GDtileinfo() retrieves tiling information about a field.

The four new routines HE5_SWwritegeograttr(), HE5_SWreadgeograttr(), HE5_SWgeograttrinfo(), and HE5_SWinqgeograttr() to write/read/retrieve an attribute associated with the "Geolocation Fields" group are added into SWapi.c.

The new routine HE5_SWindexinfo() retrieves the indices information about a subsetted region.
The three new routines HE5_GDsetalias(), HE5_GDdropalias(), and HE5_GDaliasinfo() to define/remove/retrieve aliases are added into GDapi.c.

The new inquiry routine HE5_GDinqfldalias() retrieves information about the list of data fields and aliases, and the length of the list.

The following profile routines have been added for the swath interface:

- HE5_PRdefine() - sets up the profile within the swath under the "Profile Fields" group
- HE5_PRwrite() - writes in the data to a specified profile
- HE5_PRread() - reads out the data from a specified profile
- HE5_PRinquire() - retrieves information about profiles
- HE5_PRinfo() - return information about specified profile
- HE5_PRreclaimspace() - reclaims memory used by data buffer in HE5_PRread() call
- HE5_PRwritegrpattr() - writes/updates the “Profile Fields” group attribute
- HE5_PRreadgrpattr() - reads the “Profile Fields” group attribute
- HE5_PRgrpattrinfo() - returns information about “Profile Fields” group attribute
- HE5_PRinqgrpattrs() - retrieves information about “Profile Fields” group attribute

The new routine HE5_EHset_error_on() to set a flag for suppressing HDF5 error messages is added into EHapi.c.

### 3.6 Different Types of Attributes in HDF-EOS5

Unlike HDF-EOS2 there are four different types of attributes in HDF-EOS5– global attributes, object attributes, group attributes, and local attributes. The following sample shows where each type of attribute is written in the HDF-EOS file Swath.he5.

The “GlobalAttribute_FLOAT” and the “GlobalAttribute_CHAR” attributes are global attributes. The Swath1_Attribute is an object attribute for the swath named “Swath1”. The “DF_GroupAttribute”, “GF_GroupAttribute”, and “PR_GroupAttribute” attributes are group attributes associated with the ”Data Fields”, “Geolocation Fields”, and “Profile Fields” groups, respectively. The “LocalAttribute” attribute is a field attribute associated with the “Longitude” geolocation field.

Similar attributes can be written with Grid, Point, and Zonal Average APIs.

```hdf5
HDF5 "Swath.he5" {
 GROUP "/" {
  GROUP "HDFEOS" {
    GROUP "ADDITIONAL" {
      GROUP "FILE_ATTRIBUTES" {
        ATTRIBUTE "GlobalAttribute_FLOAT" {
          DATATYPE H5T_IEEE_F32BE
          DATASPACE SIMPLE {(4) / (4)}
          DATA {
            1.11111, 2.22222, 3.33333, 4.44444
          }
        }
      }
      ATTRIBUTE "GlobalAttribute_CHAR" {
    }
  }
```

3-6 175-EED-001, Revision 01
DATATYPE H5T_STRING {
    STRSIZE 6;
    STRPAD H5T_STR_NULLTERM;
    CSET H5T_CSET_ASCII;
    CTYPE H5T_C_S1;
}
DATASPACE SCALAR
DATA {
    "AAAAAA"
}
}
GROUP "SWATHS" {
GROUP "Swath1" {
    ATTRIBUTE "Swath1_Attribute" {
        DATATYPE H5T_STD_I32BE
        DATASPACE SIMPLE { ( 4 ) / ( 4 ) }
        DATA {
            1, 2, 3, 4
        }
    }
    GROUP "Data Fields" {
        ATTRIBUTE "DF_GroupAttribute" {
            DATATYPE H5T_STD_I32BE
            DATASPACE SIMPLE { ( 4 ) / ( 4 ) }
            DATA {
                10, 20, 30, 40
            }
        }
        DATASET "Count" {
            DATATYPE H5T_STD_I32BE
            DATASPACE SIMPLE { ( 20 ) / ( 40 ) }
            DATA {
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            }
            ATTRIBUTE "_FillValue" {
                DATATYPE H5T_STD_I32BE
                DATASPACE SIMPLE { ( 1 ) / ( 1 ) }
                DATA {
                    0
                }
            }
        }
    }
    GROUP "Geolocation Fields" {
        ATTRIBUTE "GF_GroupAttribute" {
            DATATYPE H5T_STD_I32BE
            DATASPACE SIMPLE { ( 4 ) / ( 4 ) }
            DATA {
                10, 20, 30, 40
            }
        }
        DATASET "Longitude" {
            DATATYPE H5T_IEEE_F32BE
            DATASPACE SIMPLE { ( 4 ) / ( 4 ) }
            DATA {
                10, 20, 30, 40
            }
        }
    }
}
}
DATASPACE SIMPLE { (3, 10) / (3, 10) }
DATA {
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
}
ATTRIBUTE "_FillValue" {
    DATATYPE H5T_IEEE_F32BE
    DATASPACE SIMPLE { (1) / (1) }
    DATA {
        0
    }
}
ATTRIBUTE "LocalAttribute" {
    DATATYPE H5T_IEEE_F32BE
    DATASPACE SIMPLE { (4) / (4) }
    DATA {
        1.11111, 2.22222, 3.33333, 4.44444
    }
}

GROUP "Profile Fields" {
    ATTRIBUTE "PR_GroupAttribute" {
        DATATYPE H5T_STD_I32BE
        DATASPACE SIMPLE { (4) / (4) }
        DATA {
            10, 20, 30, 40
        }
    }
    DATASET "Profile-2000" {
        DATATYPE H5T_VLEN { H5T_STD_U32BE }
        DATASPACE SIMPLE { (4) / (4) }
        DATA {
            (1000, 1001, 1002, 1003, 1004, 1005, 1006, .........,
             1020, 1021, 1022, 1023, 1024),
             2036, 2037, 2038, 2039, 2040, 2041, .........),
            (3000, 3001, 3002, 3003, 3004, 3005, 3006, .........,
             3056, 3057, 3058, 3059, 3060, 3061, .........),
            (4000, 4001, 4002, 4003, 4004, 4005, 4006, .........,
             4086, 4087, 4088, 4089, 4090, 4091, .........)
        }
    }
}

GROUP "HDFEOS INFORMATION" {
    ATTRIBUTE "HDFEOSVersion" {
        DATATYPE H5T_STRING
        STRSIZE 32;
        STRPAD H5T_STR_NULLTERM;
        CSET H5T_CSET_ASCII;
    }
}
CTYPE H5T_C_S1;
}
DATASPACE SCALAR
DATA {
"HDFEOS_5.1.8"
}
DATASET "StructMetadata.0" {
DATATYPE H5T_STRING {
STRSIZE 32000;
STRPAD H5T_STR_NULLTERM;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE SCALAR
DATA {
........
........
}
}
}
}
}

To inquire about specific attribute lists (such as object, group, or local), whether the object is swath, grid, or point, it is recommended that user call the same inquiry routine twice; in the first call user passes NULL for attribute list and gets the string size for the attribute list. Then after allocating enough memory for the attribute list user calls inquiry routine for the second time, getting the attribute list. For example if we want to obtain the attribute list for the “Data Fields” group in a grid, we may do the followings:

```c
char *attrlist;
hid_t gridID;
long strbufsize, nattr;
gridID = HE5_GDopen("GRID_FILE.he5", H5F_ACC_RDWR);
nattr = HE5_GDinqgrpattrs(gridID, NULL, &strbufsize);
attrlist = (char *)malloc((strbufsize+1) * sizeof(char));
nattr = HE5_GDinqgrpattrs(gridID, attrlist, &strbufsize);
........
........
free(attrlist);
```
This page intentionally left blank.
4. Point Data

4.1 Introduction

This section will describe the routines available for storing and retrieving HDF-EOS Point Data. A Point Data set is made up of a series of data records taken at [possibly] irregular time intervals and at scattered geographic locations. Point Data is the most loosely organized form of geolocated data supported by HDF-EOS. Simply put, each data record consists of a set of one or more data values representing, in some sense, the state of a point in time and/or space.

Figure 4-1 shows an example of a simple point data set. In this example, each star on the map represents a reporting station. Each record in the data table contains the location of the point on the Earth and the measurements of the temperature and dew point at that location. This sort of point data set might represent a snapshot in time of a network of stationary weather reporting facilities.

<table>
<thead>
<tr>
<th>Lat</th>
<th>Lon</th>
<th>Temp (C)</th>
<th>Dewpt</th>
</tr>
</thead>
<tbody>
<tr>
<td>61.12</td>
<td>-149.48</td>
<td>15.00</td>
<td>5.00</td>
</tr>
<tr>
<td>45.31</td>
<td>-122.41</td>
<td>17.00</td>
<td>5.00</td>
</tr>
<tr>
<td>38.50</td>
<td>-77.00</td>
<td>24.00</td>
<td>7.00</td>
</tr>
<tr>
<td>38.39</td>
<td>-90.15</td>
<td>27.00</td>
<td>11.00</td>
</tr>
<tr>
<td>30.00</td>
<td>-90.05</td>
<td>22.00</td>
<td>7.00</td>
</tr>
<tr>
<td>37.45</td>
<td>-122.26</td>
<td>25.00</td>
<td>10.00</td>
</tr>
<tr>
<td>18.00</td>
<td>-76.45</td>
<td>27.00</td>
<td>4.00</td>
</tr>
<tr>
<td>43.40</td>
<td>-79.23</td>
<td>30.00</td>
<td>14.00</td>
</tr>
<tr>
<td>34.03</td>
<td>-118.14</td>
<td>25.00</td>
<td>4.00</td>
</tr>
<tr>
<td>32.45</td>
<td>-96.48</td>
<td>32.00</td>
<td>8.00</td>
</tr>
<tr>
<td>33.30</td>
<td>-112.00</td>
<td>30.00</td>
<td>10.00</td>
</tr>
<tr>
<td>42.15</td>
<td>-71.07</td>
<td>28.00</td>
<td>7.00</td>
</tr>
<tr>
<td>35.05</td>
<td>-106.40</td>
<td>30.00</td>
<td>9.00</td>
</tr>
<tr>
<td>34.12</td>
<td>-77.56</td>
<td>28.00</td>
<td>9.00</td>
</tr>
<tr>
<td>46.32</td>
<td>-87.25</td>
<td>30.00</td>
<td>8.00</td>
</tr>
<tr>
<td>47.36</td>
<td>-122.20</td>
<td>32.00</td>
<td>15.00</td>
</tr>
<tr>
<td>39.44</td>
<td>-104.59</td>
<td>31.00</td>
<td>16.00</td>
</tr>
<tr>
<td>21.25</td>
<td>-78.00</td>
<td>28.00</td>
<td>7.00</td>
</tr>
<tr>
<td>44.58</td>
<td>-93.15</td>
<td>32.00</td>
<td>13.00</td>
</tr>
<tr>
<td>41.49</td>
<td>-87.37</td>
<td>28.00</td>
<td>9.00</td>
</tr>
<tr>
<td>25.45</td>
<td>-80.11</td>
<td>19.00</td>
<td>3.00</td>
</tr>
</tbody>
</table>

A more realistic example might record the changes in the parameters over time by including multiple values of the parameters for each location. In this case, the identity and location of the reporting stations would remain constant, while the values of the measured parameters would vary. This sort of set up naturally leads to a hierarchical table arrangement where a second table
is used to record the static information about each reporting station, thereby removing the redundant information that would be required by a single “flat” table and acting as an index for quick access to the main data table. Such an arrangement is depicted in Figure 4-2.

An even more complex point data set may represent data taken at various times aboard a moving ship. Here, the only thing that remains constant is the identity of the reporting ship. Its location varies with each data reading and is therefore treated similarly to the data. Although this example seems more complicated than the static example cited above, its implementation is nearly identical. Figure 4-3 shows the tables resulting from this example. Note that the station location information has been moved from the static table to the data table.
In fact, the hierarchical arrangement of the tables in the last two examples can be expanded upon to include up to seven indexing levels (a total of eight levels, including the bottom level data table). A normal data access on a multi-level hierarchical point data set would involve starting at the top (first) level and following successive pointers down the structure until the desired information is found. As each level is traversed, more and more specific information is gained about the data.

In rare cases, it may be advantageous to access a point data set from the bottom up. The point data model implemented in HDF-EOS provides for backward (or upward) pointers which facilitate bottom-up access.

4.2 Applicability

The Point data model is very flexible and can be used for data at almost any level of processing. It is expected that point structure will be used for data for which there is no spatial or temporal organization, although lack of those characteristics do not preclude the use of a point structure. For example, profile data which is accumulated in sparsely located spatial averages may be most useful in a point structure.

4.3 The Point Data Interface

The Point interface consists of routines for storing, retrieving, and manipulating data in point data sets.

4.3.1 PT API Routines

All C routine names in the point data interface have the prefix “HE5_PT” and the equivalent FORTRAN routine names are prefixed by “he5_pt.” The HE5_PT routines are classified into the following categories:

- **Access routines** initialize and terminate access to the HE5_PT interface and point data sets (including opening and closing files).
- **Definition** routines allow the user to set key features of a point data set.
- **Basic I/O** routines read and write data and metadata to a point data set.
- **Index I/O** routines read and write information which links two tables in a point data set.
- **Inquiry** routines return information about data contained in a point data set.
- **Subset** routines allow reading of data from a specified geographic region.

The supported HE5_PT function calls are listed in Table 4-1 and are described in detail in the Software Reference Guide Vol. 2, User’s Guide that accompanies this document. The page number column in the following table refers to the Software Reference Guide.
<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>FORTRAN</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>HE5_PTopen</td>
<td>he5_ptopen</td>
<td>Creates a new file or opens an existing one</td>
<td>2-32</td>
</tr>
<tr>
<td></td>
<td>HE5_PTCREATE</td>
<td>he5_ptcreate</td>
<td>Creates a new point data set and returns a handle</td>
<td>2-06</td>
</tr>
<tr>
<td></td>
<td>HE5_PTADETACH</td>
<td>he5_ptdetach</td>
<td>Attaches to an existing point data set</td>
<td>2-02</td>
</tr>
<tr>
<td></td>
<td>HE5_PTCLOSE</td>
<td>he5_ptclose</td>
<td>Releases a point data set and frees memory</td>
<td>2-14</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_PTDEFLEVEL</td>
<td>he5_ptdeflevel</td>
<td>Defines a level within the point data set</td>
<td>2-07</td>
</tr>
<tr>
<td></td>
<td>HE5_PTDLINKAGE</td>
<td>he5_ptdlinkage</td>
<td>Defines link field to use between two levels</td>
<td>2-13</td>
</tr>
<tr>
<td>Basic I/O</td>
<td>HE5_PTWRITELEVEL</td>
<td>he5_ptwritelevel</td>
<td>Writes (appends) full records to a level</td>
<td>2-42</td>
</tr>
<tr>
<td></td>
<td>HE5_PUPDATELEVEL</td>
<td>he5_ptupdatelevel</td>
<td>Updates the specified fields and records of a level</td>
<td>2-37</td>
</tr>
<tr>
<td></td>
<td>HE5_PWRITEATTR</td>
<td>he5_ptwriteattr</td>
<td>Creates or updates an attribute of the point data set</td>
<td>2-38</td>
</tr>
<tr>
<td></td>
<td>HE5_PWRITEGRPATTR</td>
<td>he5_ptwritegrpattr</td>
<td>Creates or updates group attribute</td>
<td>2-40</td>
</tr>
<tr>
<td></td>
<td>HE5_PWRITELOCATTR</td>
<td>he5_ptwritelocattr</td>
<td>Creates or updates local attribute</td>
<td>2-43</td>
</tr>
<tr>
<td></td>
<td>HE5_PREADATTR</td>
<td>he5_ptreadattr</td>
<td>Reads existing attribute of point data set</td>
<td>2-33</td>
</tr>
<tr>
<td>Inquiry</td>
<td>HE5_PNTLEVELS</td>
<td>he5_ptnlevels</td>
<td>Returns the number of levels in a point data set</td>
<td>2-30</td>
</tr>
<tr>
<td></td>
<td>HE5_PTNRECS</td>
<td>he5_ptnrecs</td>
<td>Returns the number of records in a level</td>
<td>2-31</td>
</tr>
<tr>
<td></td>
<td>HE5_PTNFIELDS</td>
<td>he5_ptnfields</td>
<td>Returns number of fields defined in a level</td>
<td>2-29</td>
</tr>
<tr>
<td></td>
<td>HE5_PLEVELINFO</td>
<td>he5_ptlevelinfo</td>
<td>Returns information about a given level</td>
<td>2-27</td>
</tr>
<tr>
<td></td>
<td>HE5_PLEVELINDEX</td>
<td>he5_ptlevelindex</td>
<td>Returns index number for a named level</td>
<td>2-26</td>
</tr>
<tr>
<td></td>
<td>HE5_PTBACKLINKINFO</td>
<td>he5_ptbacklinkinfo</td>
<td>Returns link field to previous level</td>
<td>2-04</td>
</tr>
<tr>
<td></td>
<td>HE5_PTFLDLINKINFO</td>
<td>he5_ptfdlinkinfo</td>
<td>Returns link field to following level</td>
<td>2-15</td>
</tr>
<tr>
<td></td>
<td>HE5_PGETLEVELNAME</td>
<td>he5_ptgetlevelname</td>
<td>Returns level name given level number</td>
<td>2-16</td>
</tr>
<tr>
<td></td>
<td>HE5_PGETRECNUMS</td>
<td>None</td>
<td>Retrieves number of records in one level corresponding to a group of records in a different level</td>
<td>2-17</td>
</tr>
<tr>
<td></td>
<td>HE5_PATTRINFO</td>
<td>he5_ptattrinfo</td>
<td>Returns information about point attributes</td>
<td>2-03</td>
</tr>
<tr>
<td></td>
<td>HE5_PGROUPATTRINFO</td>
<td>he5_pgroupattrinfo</td>
<td>Returns information about group attribute</td>
<td>2-18</td>
</tr>
<tr>
<td></td>
<td>HE5_PLOCATTRINFO</td>
<td>he5_ptlocattrinfo</td>
<td>Returns information about local (level) attribute</td>
<td>2-28</td>
</tr>
<tr>
<td></td>
<td>HE5_PINTQATTRS</td>
<td>he5_ptinqattrs</td>
<td>Retrieves number and names of attributes defined</td>
<td>2-19</td>
</tr>
</tbody>
</table>
Table 4-1 Summary of the Point Interface (2 of 2)

<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>C Routine Name</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inquiry</td>
<td>HE5_Ptinggrpattrs</td>
<td>he5_ptinggrpattrs</td>
<td>Retrieves information about group attributes</td>
<td>2-22</td>
</tr>
<tr>
<td></td>
<td>HE5_Ptinglocattrs</td>
<td>he5_ptinglocattrs</td>
<td>Retrieves information about local (level) attributes</td>
<td>2-23</td>
</tr>
<tr>
<td></td>
<td>HE5_Ptingpoint</td>
<td>he5_ptingpoint</td>
<td>Retrieves number and names of points in file</td>
<td>2-25</td>
</tr>
<tr>
<td></td>
<td>HE5_Ptingdatatype</td>
<td>he5_ptingdatatype</td>
<td>Returns data type information about specified level in point</td>
<td>2-20</td>
</tr>
</tbody>
</table>

4.3.2 File Identifiers

As with all HDF-EOS interfaces, file identifiers in the HE5_PT interface are of hid_t HDF5 type, each uniquely identifying one open data file. They are not interchangeable with other file identifiers created with other interfaces.

4.3.3 Point Identifiers

Before a point data set is accessed, it is identified by a name which is assigned to it upon its creation. The name is used to obtain a point identifier. After a point data set has been opened for access, it is uniquely identified by its point identifier.

4.4 Programming Model

The programming model for accessing a point object through the HE5_PT interface is as follows:

1. Open the file and initialize the HE5_PT interface by obtaining a file ID from a file name.
2. Open or create a point object by obtaining a point ID from a point name.
3. Perform desired operations on the data set.
4. Close the point object by disposing of the point ID.
5. Terminate point access to the file by disposing of the file ID.

In this example we open the HDF-EOS point file, "Point.he5". Assuming that this file may not exist, we are using the "H5F_ACC_TRUNC" access code.

The "HE5_PTopen" function returns the point file ID, ptfid, which is used to identify the file in subsequent calls to the HDF-EOS library functions.

```c
ptfid = HE5_PTopen("Point.he5", H5F_ACC_TRUNC);
/* Set up the point structures */
PTid1 = HE5_PTcreate(ptfid, "Simple Point");
PTid2 = HE5_PTcreate(ptfid, "FixedBuoy Point");
PTid3 = HE5_PTcreate(ptfid, "FloatBuoy Point");
/* Close the point interface */
status = HE5_PTdetach(PTid1);
status = HE5_PTdetach(PTid2);
```
status = HE5_PTdetach(PTid3);

/* Close the point file */
status = HE5_PTClose(ptfid);

To access several files at the same time, a calling program must obtain a separate ID for each file to be opened. Similarly, to access more than one point object, a calling program must obtain a separate point ID for each object. For example, to open two objects stored in two files, a program would execute the following series of C function calls:

ptfid_1 = HE5_POpen(filename_1, access_mode);
ptid_1 = HE5_PAttach(ptfid_1, point_name_1);
ptfid_2 = HE5_POpen(filename_2, access_mode);
ptid_2 = HE5_PAttach(ptfid_2, point_name_2);
<Optional operations>
status = HE5_PTdetach(ptid_1);
status = HE5_PTClose(ptfid_1);
status = HE5_PTdetach(ptid_2);
status = HE5_PTClose(ptfid_2);

Because each file and point object is assigned its own identifier, the order in which files and objects are accessed is very flexible. However, it is very important that the calling program individually discard each identifier before terminating. Failure to do so can result in empty or, even worse, invalid files being produced.
5. Swath Data

5.1 Introduction

The Swath concept for HDF-EOS is based on a typical satellite swath, where an instrument takes a series of scans perpendicular to the ground track of the satellite as it moves along that ground track. Figure 5-1 below shows this traditional view of a swath.

![Figure 5-1. A Typical Satellite Swath: Scanning Instrument](image)

Another type of data that the Swath is equally well suited to arises from a sensor that measures a vertical profile, instead of scanning across the ground track. The resulting data resembles a standard Swath tipped up on its edge. Figure 5-2 shows how such a Swath might look.

In fact, the two approaches shown in Figures 5-1 and 5-2 can be combined to manage a profiling instrument that scans across the ground track. The result would be a three dimensional array of measurements where two of the dimensions correspond to the standard scanning dimensions (along the ground track and across the ground track), and the third dimension represents a height above the Earth or a range from the sensor. The "horizontal" dimensions can be handled as normal geographic dimensions, while the third dimension can be handled as a special "vertical" dimension.
A standard Swath is made up of four primary parts: data fields, geolocation fields, dimensions, and dimension maps. An optional fifth part called an index can be added to support certain kinds of access to Swath data. Each of the parts of a Swath is described in detail in the following subsections.

### 5.1.1 Data Fields

Data fields are the main part of a Swath from a science perspective. Data fields usually contain the raw data (often as *counts*) taken by the sensor or parameters derived from that data on a value-for-value basis. All the other parts of the Swath exist to provide information about the data fields or to support particular types of access to them. Data fields typically are two-dimensional arrays, but can have as few as one dimension or as many as eight, in the current library implementation. They can have any valid C data type.
5.1.2 Geolocation Fields

Geolocation fields allow the Swath to be accurately tied to particular points on the Earth’s surface. To do this, the Swath interface requires the presence of at least a time field (“Time”) or a latitude/longitude field pair (“Latitude” and “Longitude”). Geolocation fields must be either one- or two-dimensional and can have any data type.

In addition to the Geolocation fields “Latitude”, “Longitude”, and “Time”, one can define other geolocation fields related to the third or higher dims in the datafields. For example if the third dim in a datafield is altitude, then one can define a geofield called “Altitude” and map the third dimension in the data field to the dimension of this field like the first 2 dimensions in the swath.

5.1.3 Dimensions

Dimensions define the axes of the data and geolocation fields by giving them names and sizes. In using the library, dimensions must be defined before they can be used to describe data or geolocation fields.

Every axis of every data or geolocation field, then, must have a dimension associated with it. However, there is no requirement that they all be unique. In other words, different data and geolocation fields may share the same named dimension. In fact, sharing dimension names allows the Swath interface to make some assumptions about the data and geolocation fields involved which can reduce the complexity of the file and simplify the program creating or reading the file.

5.1.4 Dimension Maps

Dimension maps are the glue that holds the Swath together. They define the relationship between data fields and geolocation fields by defining, one-by-one, the relationship of each dimension of each geolocation field with the corresponding dimension in each data field. In cases where a data field and a geolocation field share a named dimension, no explicit dimension map is needed. In cases where a data field has more dimensions than the geolocation fields, the “extra” dimensions are left unmapped.

In many cases, the size of a geolocation dimension will be different from the size of the corresponding data dimension. To take care of such occurrences, there are two pieces of information that must be supplied when defining a dimension map: the offset and the increment. The offset tells how far along a data dimension you must travel to find the first point to have a corresponding entry along the geolocation dimension. The increment tells how many points to travel along the data dimension before the next point is found for which there is a corresponding entry along the geolocation dimension. Figure 5-3 depicts a dimension map.

---

1 “Co-latitude” may be substituted for “Latitude.”
The “data skipping” method described above works quite well if there are fewer regularly spaced geolocation points than data points along a particular pair of mapped dimensions of a Swath. It is conceivable, however, that the reverse is true – that there are more regularly spaced geolocation points than data points. In that event, both the offset and increment should be expressed as negative values to indicate the reversed relationship. The result is shown in Figure 5-4. Note that in the reversed relationship, the offset and increment are applied to the geolocation dimension rather than the data dimension.
5.1.5 Index

The index was designed specifically for Landsat 7 data products. These products require geolocation information that does not repeat at regular intervals throughout the Swath. The index allows the Swath to be broken into unequal length scenes which can be individually geolocated.

For this version of the HDF-EOS library, there is no particular content required for the index. It is quite likely that a later version of the library will impose content requirements on the index in an effort to standardize its use.

5.2 Applicability

The Swath data model is most useful for satellite [or similar] data at a low level of processing. The Swath model is best suited to data at EOS processing levels 1A, 1B, and 2.

5.3 The Swath Data Interface

The SW interface consists of routines for storing, retrieving, and manipulating data in swath data sets.

5.3.1 SW API Routines

All C routine names in the swath data interface have the prefix “HE5_SW” and the equivalent FORTRAN routine names are prefixed by “he5_sw”. The 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.

The SW function calls are listed in Table 5-1 and are described in detail in the Software Reference Guide that accompanies this document. The page number column in the following table refers to the Software Reference Guide.
<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>HE5_SWopen</td>
<td>he5_swopen Opens or creates HDF file in order to create, read, or write a swath</td>
<td>2-122</td>
</tr>
<tr>
<td></td>
<td>HE5_SWcreate</td>
<td>he5_swcreate Creates a swath within the file</td>
<td>2-53</td>
</tr>
<tr>
<td></td>
<td>HE5_SWattach</td>
<td>he5_swattach Attaches to an existing swath within the file</td>
<td>2-47</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdetach</td>
<td>he5_swdetach Detaches from swath interface</td>
<td>2-76</td>
</tr>
<tr>
<td></td>
<td>HE5_SWclose</td>
<td>he5_swclose Closes file</td>
<td>2-51</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefdim</td>
<td>he5_swdefdim Defines a new dimension within the swath</td>
<td>2-64</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_SWdefdimmap</td>
<td>he5_swdefmap Defines the mapping between the geolocation and data dimensions</td>
<td>2-66</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefidxmap</td>
<td>he5_swdefmap Defines a non-regular mapping between the geolocation and data dimension</td>
<td>2-70</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefgeofield</td>
<td>he5_swdefgeofield Defines a new geolocation field within the swath</td>
<td>2-68</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefdatafield</td>
<td>he5_swdefdatafield Defines a new data field within the swath</td>
<td>2-62</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefcomp</td>
<td>he5_swdefcomp Defines a field compression scheme</td>
<td>2-59</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefchunk</td>
<td>he5_swdefchunk Define chunking parameters</td>
<td>2-56</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdefcomchunk</td>
<td>he5_swdefcomchunk Defines compression with automatic chunking</td>
<td>2-57</td>
</tr>
<tr>
<td></td>
<td>HE5_Wsetalias</td>
<td>he5_wsetalias Defines alias for data field</td>
<td>2-139</td>
</tr>
<tr>
<td></td>
<td>HE5_Wdropalias</td>
<td>he5_wdropalias Removes alias from the list of field aliases</td>
<td>2-78</td>
</tr>
<tr>
<td></td>
<td>HE5_SWfldrename</td>
<td>he5_swfldrename Changes the field name</td>
<td>2-86</td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritefield</td>
<td>he5_swwritefiel Writes data to a swath field</td>
<td>2-154</td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritegeometra</td>
<td>he5_swwritegeometa Writes field metadata for an existing swath geolocation field</td>
<td>2-159</td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritedatameta</td>
<td>he5_swwritedatameta Writes field metadata for an existing swath data field</td>
<td>2-150</td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadfield</td>
<td>he5_swreadfield Reads data from a swath field</td>
<td>2-130</td>
</tr>
<tr>
<td></td>
<td>HE5_SWwriteattr</td>
<td>he5_swwriteattr Writes/updates attribute in a swath</td>
<td>2-148</td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadattr</td>
<td>he5_swreadattr Reads attribute from a swath</td>
<td>2-125</td>
</tr>
<tr>
<td>Basic I/O</td>
<td>HE5_SWwritegeogrppattr</td>
<td>he5_swwritegeogrppattr Writes/updates group Geolocation Fields attribute in a swath</td>
<td>2-157</td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritegrppattr</td>
<td>he5_swwritegrppattr Writes/updates group Data Fields attribute in a swath</td>
<td>2-161</td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritelocattr</td>
<td>he5_swwritelocattr Write/updates local attribute in a swath</td>
<td>2-163</td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadgeogrppattr</td>
<td>he5_swreadgeogrppattr Reads attribute in Geolocation Fields from swath</td>
<td>2-132</td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadgrppattr</td>
<td>he5_swreadgrppattr Reads attribute in Data Fields from a swath</td>
<td>2-133</td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadlocattr</td>
<td>he5_swreadlocattr Reads attribute from a swath</td>
<td>2-134</td>
</tr>
<tr>
<td></td>
<td>HE5_Wsetfieldvalue</td>
<td>he5_wsetfieldvalue Sets fill value for the specified field</td>
<td>2-143</td>
</tr>
<tr>
<td></td>
<td>HE5_Wgetfieldvalue</td>
<td>he5_wgetfieldvalue Retrieves fill value for the specified field</td>
<td>2-93</td>
</tr>
<tr>
<td>Inquiry</td>
<td>HE5_SWaliasinfo</td>
<td>he5_swaliasinfo Retrieves information about field aliases</td>
<td>2-46</td>
</tr>
<tr>
<td></td>
<td>HE5_SWgetaliaslist</td>
<td>he5_swgetaliaslist Retrieves list and number of aliases in a geo or data group</td>
<td>2-89</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqdims</td>
<td>he5_swinqdims Retrieves information about dimensions defined in swath</td>
<td>2-104</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqmaps</td>
<td>he5_swinqmaps Retrieves information about the geolocation relations defined</td>
<td>2-116</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqidxmaps</td>
<td>he5_swinqidxmaps Retrieves information about the indexed geolocation/data mappings defined</td>
<td>2-113</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqgeofields</td>
<td>he5_swinqgeofields Retrieves information about the geolocation fields defined</td>
<td>2-107</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqdatafields</td>
<td>he5_swinqdatafields Retrieves information about the data fields defined</td>
<td>2-99</td>
</tr>
<tr>
<td>Category</td>
<td>Routine Name</td>
<td>FORTRAN</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Inquiry</strong></td>
<td><strong>HE5_SWinqattrs</strong></td>
<td>he5_swinqattrs</td>
<td>Retrieves number and names of attributes defined in swath</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWinqdatatype</strong></td>
<td>he5_swidtype</td>
<td>Returns data type information about specified fields in swath</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWinqdfldalias</strong></td>
<td>he5_swdfldalias</td>
<td>Returns information about data fields &amp; aliases defined in swath</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWinqgeogrpattrs</strong></td>
<td>he5_swgeogrpattrs</td>
<td>Retrieve information about group Geolocation Fields attributes defined in swath</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWinqgrpattrs</strong></td>
<td>he5_swgrpattrs</td>
<td>Retrieve information about group Data Fields attributes defined in swath</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWlocattrs</strong></td>
<td>he5_swlocattrs</td>
<td>Retrieve information about local attributes defined in swath</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWlocattrinfo</strong></td>
<td>he5_swlocattrinfo</td>
<td>Retrieve information about a data field’s local attribute(s)</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWnentries</strong></td>
<td>he5_swnentries</td>
<td>Returns number of entries and descriptive string buffer size for a specified entity</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWdiminfo</strong></td>
<td>he5_swdiminfo</td>
<td>Retrieve size of specified dimension</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWmapinfo</strong></td>
<td>he5_swmapinfo</td>
<td>Retrieve offset and increment of specified geolocation mapping</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWidxmapinfo</strong></td>
<td>he5_swidxmapinfo</td>
<td>Retrieve offset and increment of specified geolocation mapping</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWattrinfo</strong></td>
<td>he5_swattrinfo</td>
<td>Retrieve compression information about a file</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWgeomapinfo</strong></td>
<td>he5_swgeomapinfo</td>
<td>Retrieve type of dimension mapping for a dimension</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWregioninfo</strong></td>
<td>he5_swrreginfo</td>
<td>Returns information about a specific geolocation or data field</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWdefboxregion</strong></td>
<td>he5_swdefboxregion</td>
<td>Define region of interest by latitude/longitude</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWregionindex</strong></td>
<td>he5_swrregidx</td>
<td>Returns information about the swath region ID</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWextractregion</strong></td>
<td>he5_swextractregion</td>
<td>Read a region of interest from a field</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWextracttimeperiod</strong></td>
<td>he5_swextracttimeperiod</td>
<td>Read a region of interest from a field</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWduplicate</strong></td>
<td>he5_swduplicate</td>
<td>Define a region of interest by vertical field</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWindexinfo</strong></td>
<td>he5_swindexinfo</td>
<td>Returns the indices about a subsetted region</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_SWdupregion</strong></td>
<td>he5_swdupregion</td>
<td>Duplicate a region or time period</td>
</tr>
<tr>
<td><strong>Subset</strong></td>
<td><strong>HE5_PRdefine</strong></td>
<td>he5_prdefine</td>
<td>Defines profile data structure</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_PRread</strong></td>
<td>he5_prread</td>
<td>Reads profile data</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_PRwrite</strong></td>
<td>he5_prwrite</td>
<td>Writes profile data</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_PRinquire</strong></td>
<td>he5_prinquire</td>
<td>Retrieves information about profiles</td>
</tr>
<tr>
<td></td>
<td><strong>HE5_PRinfo</strong></td>
<td>he5_prinfo</td>
<td>Return information about profile</td>
</tr>
</tbody>
</table>
### Table 5-1. Summary of the Swath Interface (3 of 3)

<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><strong>Profile</strong></td>
<td>HE5_PRreclaimspace</td>
<td>Not available</td>
<td>Reclaims memory used by data buffer in HE5_PRRead()call</td>
<td>2-174</td>
</tr>
<tr>
<td></td>
<td>HE5_PRwritegrpattr</td>
<td>he5_prwrgattr</td>
<td>Writes/updates group Profile Fields attribute in a swath</td>
<td>2-177</td>
</tr>
<tr>
<td></td>
<td>HE5_PRreadgrpattr</td>
<td>he5_prrdgattr</td>
<td>Reads attribute in group Profile Fields from a swath</td>
<td>2-173</td>
</tr>
<tr>
<td></td>
<td>HE5_PRinggrpattrs</td>
<td>he5_prinqgattrs</td>
<td>Retrieves information about group Profile Fields attributes defined in swath</td>
<td>2-169</td>
</tr>
<tr>
<td></td>
<td>HE5_PRgrpattrinfo</td>
<td>he5_prgrattrinfo</td>
<td>Returns information about a group Profile Fields attribute</td>
<td>2-166</td>
</tr>
<tr>
<td><strong>External Files</strong></td>
<td>HE5_SWmountexternal</td>
<td>Not available</td>
<td>Mount external data file</td>
<td>2-120</td>
</tr>
<tr>
<td></td>
<td>HE5_SWreadexternal</td>
<td>Not available</td>
<td>Read external data set</td>
<td>2-129</td>
</tr>
<tr>
<td></td>
<td>HE5_SWunmount</td>
<td>Not available</td>
<td>Dismount external data file</td>
<td>2-145</td>
</tr>
<tr>
<td><strong>External Data Sets</strong></td>
<td>HE5_SWsetextdata</td>
<td>he5_swsetxdat</td>
<td>Set external data set</td>
<td>2-142</td>
</tr>
<tr>
<td></td>
<td>HE5_SWgetextdata</td>
<td>he5_swgetxdat</td>
<td>Get external data set</td>
<td>2-92</td>
</tr>
<tr>
<td><strong>Dimension Scale</strong></td>
<td>HE5_SWsetdimscale</td>
<td>he5_swsetdimscale</td>
<td>Sets dimension scale for a field dimension within the swath</td>
<td>2-140</td>
</tr>
<tr>
<td></td>
<td>HE5_SWgetdimscale</td>
<td>he5_swgetdimscale</td>
<td>Gets dimension scale for a field dimension within the swath</td>
<td>2-90</td>
</tr>
<tr>
<td></td>
<td>HE5_SWwritedscaleattr</td>
<td>he5_swwritedscaleattr</td>
<td>Writes/Updates a dimension scale attribute in a specific swath</td>
<td>2-151</td>
</tr>
<tr>
<td></td>
<td>HE5_SWreaddscaleattr</td>
<td>he5_swreaddscaleattr</td>
<td>Reads a dimension scale attribute from a specific dimension</td>
<td>2-126</td>
</tr>
<tr>
<td></td>
<td>HE5_SWinqdscaleattrs</td>
<td>he5_swinqdscaleattrs</td>
<td>Retrieve information about the attributes defined for a specific dimension scale</td>
<td>2-105</td>
</tr>
<tr>
<td></td>
<td>HE5_SWdscaleattrinfo</td>
<td>he5_swdscaleattrinfo</td>
<td>Returns information about attribute(s) in a specific dimension scale</td>
<td>2-79</td>
</tr>
</tbody>
</table>

#### 5.3.2 File Identifiers

As with all HDF-EOS interfaces, file identifiers in the HE5_SW interface are of hid_t HDF5 type, each uniquely identifying one open data file. They are not interchangeable with other file identifiers created with other interfaces.

#### 5.3.3 Swath Identifiers

Before a swath data set is accessed, it is identified by a name which is assigned to it upon its creation. The name is used to obtain a *swath identifier*. After a swath data set has been opened for access, it is uniquely identified by its swath identifier.

#### 5.4 Programming Model

The programming model for accessing a swath data set through the HE5_SW interface is as follows:

1. Open the file and initialize the HE5_SW interface by obtaining a file ID from a file name.
2. Open or create a swath object by obtaining a swath ID from a swath name.
3. Perform desired operations on the data set.
4. Close the swath data set by disposing of the swath ID.
5. Terminate swath access to the file by disposing of the file ID.

/* In this example we open an HDF-EOS file, (2) create the swath object within the file, and define the swath field dimensions. */

Open a new HDF-EOS swath file, "Swath.he5". Assuming that this file may not exist, we are using "H5F_ACC_TRUNC" access code. The "HE5_SWopen" function returns the swath file ID, swfid, which is used to identify the file in subsequent calls to the HDF-EOS library functions. */

swfid = HE5_SWopen("Swath.he5", H5F_ACC_TRUNC);

/* Create the swath, "Swath1", within the file */ SWid = HE5_SWcreate(swfid, "Swath1");

/* Define dimensions and specify their sizes */
status = HE5_SWdefdim(SWid, "GeoTrack", 20);
status = HE5_SWdefdim(SWid, "GeoXtrack", 10);
status = HE5_SWdefdim(SWid, "Res2tr", 40);
status = HE5_SWdefdim(SWid, "Res2xtr", 20);
status = HE5_SWdefdim(SWid, "Bands", 15);
status = HE5_SWdefdim(SWid, "ProfDim", 4);

/* Define "Unlimited" Dimension */
status = HE5_SWdefdim(SWid, "Unlim", H5S_UNLIMITED);

/* Once the dimensions are defined, the relationship (mapping) between the geolocation dimensions, such as track and cross track, and the data dimensions, must be established. This is done through the "HE5_SWdefdimmap" function. It takes as input the swath id, the names of the dimensions designating the geolocation and data dimensions, respectively, and the offset and increment defining the relation. */

In the first example we relate the "GeoTrack" and "Res2tr" dimensions with an offset of 0 and an increment of 2. Thus the ith element of "Geotrack" corresponds to the 2 * ith element of "Res2tr".

In the second example, the ith element of "GeoXtrack" corresponds to the 2 * ith + 1 element of "Res2xtr".

Note that there is no relationship between the geolocation dimensions and the "Bands" dimension. */

/* Define Dimension_mappings */
status = HE5_SWdefdimmap(SWid, "GeoTrack", "Res2tr", 0, 2);
status = HE5_SWdefdimmap(SWid, "GeoXtrack", "Res2xtr", 1, 2);

/* Define Indexed Mapping */
status = HE5_SWdefidxmap(SWid, "IndxTrack", "Res2tr", indx);

/* Close the swath interface */
status = HE5_SWdetach(SWid);

/* Close the swath file */
status = HE5_SWclose(swfid);
6. Grid Data

6.1 Introduction

This section will describe the routines available for storing and retrieving HDF-EOS Grid Data. A Grid data set is similar to a swath in that it contains a series of data fields of two or more dimensions. The main difference between a Grid and a Swath is in the character of their geolocation information.

As described in Section 4, swaths carry geolocation information as a series of individually located points (tie points or ground control points). Grids, though, carry their geolocation in a much more compact form. A grid merely contains a set of projection equations (or references to them) along with their relevant parameters. Together, these relatively few pieces of information define the location of all points in the grid. The equations and parameters can then be used to compute the latitude and longitude for any point in the grid.

![Figure 6-1. A Data Field in a Mercator-projected Grid](image)

In loose terms, each data field constitutes a map in a given standard projection. Although there may be many independent Grids in a single HDF-EOS file, within each Grid only one projection may be chosen for application to all data fields. Figures 6-1 and 6-2 show how a single data field may look in a Grid using two common projections.

There are three important features of a Grid data set: the data fields, the dimensions, and the projection. Each of these is discussed in detail in the following subsections.
6.1.1 Data Fields

The data fields are, of course, the most important part of the Grid. Data fields in a Grid data set are rectilinear arrays of two or more dimensions. Most commonly, they are simply two-dimensional rectangular arrays. Generally, each field contains data of similar scientific nature which must share the same data type. The data fields are related to each other by common geolocation. That is, a single set of geolocation information is used for all data fields within one Grid data set.

6.1.2 Dimensions

Dimensions are used to relate data fields to each other and to the geolocation information. To be interpreted properly, each data field must make use of the two predefined dimensions: “XDim” and “YDim”. These two dimensions are defined when the grid is created and are used to refer to the X and Y dimensions of the chosen projection (see 6.1.3 below). Although there is no practical limit on the number of dimensions a data field in a Grid data set may have, it is not likely that many fields will need more than three: the predefined dimensions “XDim” and “YDim” and a third dimension for depth or height.

6.1.3 Projections

The projection is really the heart of the Grid. Without the use of a projection, the Grid would not be substantially different from a Swath. The projection provides a convenient way to encode geolocation information as a set of mathematical equations which are capable of transforming Earth coordinates (latitude and longitude) to X-Y coordinates on a sheet of paper.
The choice of a projection to be used for a Grid is a critical decision for a data product designer. There is a large number of projections that have been used throughout history. In fact, some projections date back to ancient Greece. For the purposes of this release of HDF-EOS, however, only six families of projections are supported: Geographic, Interrupted Goode’s Homolosine, Polar Stereographic, Universal Transverse Mercator, Space Oblique, and Lambert Azimuthal Equal Area. These projections coincide with those supported by the SDP Toolkit.

The producer’s choice of a projection should be governed by knowledge of the specific properties of each projection and a thorough understanding of the requirements of the data set’s users. Two excellent resources for information on projections and their properties are the USGS Professional Papers cited in Section 2.2 “Related Documents”.

This release of HDF-EOS assumes that the data producer will use to create the data the General Coordinate Transformation Package (GCTP), a library of projection software available from the U.S. Geological Survey. This manual will not attempt to explain the use of GCTP. Adequate documentation accompanies the GCTP source code. For the purposes of this Grid interface, the data are assumed to have already been projected. The Grid interface allows the data producer to specify the exact GCTP parameters used to perform the projection and will provide for basic subsetting of the data fields by latitude/longitude bounding box.

See section below for further details on the usage of the GCTP package.

6.2 Applicability

The Grid data model is intended for data processed at a high level. It is most applicable to data at EOS processing levels 3 and 4.

6.3 The Grid Data Interface

The GD interface consists of routines for storing, retrieving, and manipulating data in grid data sets.

6.3.1 GD API Routines

All C routine names in the grid data interface have the prefix “HE5_GD” and the equivalent FORTRAN routine names are prefixed by “he5_gd”. The GD routines are classified into the following categories:

- **Access routines** initialize and terminate access to the 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.
The GD function calls are listed in Table 6-1 and are described in detail in the Software Reference Guide that accompanies this document. The page number column in the following table refers to the Software Reference Guide.

<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>FORTRAN</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>HE5_GDopen</td>
<td>he5_gdopen</td>
<td>Creates a new file or opens an existing one</td>
<td>2-244</td>
</tr>
<tr>
<td></td>
<td>HE5_GDcreate</td>
<td>he5_gdcreate</td>
<td>Creates a new grid in the file</td>
<td>2-187</td>
</tr>
<tr>
<td></td>
<td>HE5_GDattach</td>
<td>he5_gdattach</td>
<td>Attaches to a grid</td>
<td>2-181</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdetach</td>
<td>he5_gddetach</td>
<td>Detaches from grid interface</td>
<td>2-209</td>
</tr>
<tr>
<td></td>
<td>HE5_GDclose</td>
<td>he5_gdclose</td>
<td>Closes file</td>
<td>2-185</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_GDdeforigin</td>
<td>he5_gddeforigin</td>
<td>Defines origin of grid pixel</td>
<td>2-198</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdefdim</td>
<td>he5_gddefdim</td>
<td>Defines dimensions for a grid</td>
<td>2-195</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdefproj</td>
<td>he5_gddefproj</td>
<td>Defines projection of grid</td>
<td>2-200</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdefpixreg</td>
<td>he5_gddefpixreg</td>
<td>Defines pixel registration within grid cell</td>
<td>2-199</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdeffield</td>
<td>he5_gddeffield</td>
<td>Defines data fields to be stored in a grid</td>
<td>2-196</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdefcomp</td>
<td>he5_gddefcomp</td>
<td>Defines a field compression scheme</td>
<td>2-191</td>
</tr>
<tr>
<td></td>
<td>HE5_GBlkSOMoffset</td>
<td>None</td>
<td>This is a special function for SOM MISR data. Write block SOM offset values.</td>
<td>2-183</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdefcomtile</td>
<td>he5_gddefcomtile</td>
<td>Defines compression with automatic tiling</td>
<td>2-194</td>
</tr>
<tr>
<td></td>
<td>HE5_GDsetalias</td>
<td>he5_gdsetalias</td>
<td>Defines alias for data field</td>
<td>2-259</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdropalias</td>
<td>he5_gddropalias</td>
<td>Removes alias from a list of field aliases</td>
<td>2-211</td>
</tr>
<tr>
<td>Basic I/O</td>
<td>HE5_GDwritefieldmeta</td>
<td>he5_gdwrmeta</td>
<td>Writes metadata for field already existing in file</td>
<td>2-274</td>
</tr>
<tr>
<td></td>
<td>HE5_GDreadfield</td>
<td>he5_gdrdfield</td>
<td>Writes data to a grid field</td>
<td>2-271</td>
</tr>
<tr>
<td></td>
<td>HE5_GDwriteattr</td>
<td>he5_gdwattr</td>
<td>Writes/updates attribute in a grid</td>
<td>2-266</td>
</tr>
<tr>
<td></td>
<td>HE5_GDwritelocattr</td>
<td>he5_gdwlocattr</td>
<td>Writes/updates local attribute in a grid</td>
<td>2-277</td>
</tr>
<tr>
<td></td>
<td>HE5_GDwritegrppattr</td>
<td>he5_gdgppattr</td>
<td>Writes/updates group attribute in a grid</td>
<td>2-275</td>
</tr>
<tr>
<td></td>
<td>HE5_GDreadattr</td>
<td>he5_grdattr</td>
<td>Reads attribute from a grid</td>
<td>2-249</td>
</tr>
<tr>
<td></td>
<td>HE5_GDreadgrppattr</td>
<td>he5_grdgrppattr</td>
<td>Reads group attribute from a grid</td>
<td>2-255</td>
</tr>
<tr>
<td></td>
<td>HE5_GDreadlocattr</td>
<td>he5_grdlocattr</td>
<td>Reads local attribute from a grid</td>
<td>2-256</td>
</tr>
<tr>
<td></td>
<td>HE5_GDsetfillvalue</td>
<td>he5_gdsetfill</td>
<td>Sets fill value for the specified field</td>
<td>2-263</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgetfillvalue</td>
<td>he5_gdgetfill</td>
<td>Retrieves fill value for the specified field</td>
<td>2-221</td>
</tr>
<tr>
<td>Inquiry</td>
<td>HE5_GDgetaliaslist</td>
<td>he5_gdgetaliaslist</td>
<td>Retrieves list and number of aliases in a data group</td>
<td>2-217</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqdims</td>
<td>he5_gdinqdims</td>
<td>Retrieves information about dimensions defined in grid</td>
<td>2-231</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqfields</td>
<td>he5_gdinqfields</td>
<td>Retrieves information about the data fields defined in grid</td>
<td>2-234</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqattrs</td>
<td>he5_gdinqattrs</td>
<td>Retrieves number and names of attributes defined</td>
<td>2-228</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqlocattrs</td>
<td>he5_gdinqlocattrs</td>
<td>Retrieves information about local attributes defined for a field</td>
<td>2-238</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqgrppattrs</td>
<td>he5_gdinqgrppattrs</td>
<td>Retrieves information about group attributes defined for a field</td>
<td>2-237</td>
</tr>
<tr>
<td></td>
<td>HE5_GDnentries</td>
<td>he5_gdnentries</td>
<td>Returns number of entries and descriptive string buffer size for a specified entity</td>
<td>2-243</td>
</tr>
<tr>
<td></td>
<td>HE5_GDaliasinfo</td>
<td>he5_gdaliasinfo</td>
<td>Retrieves information about aliases</td>
<td>2-180</td>
</tr>
</tbody>
</table>
### Table 6-1. 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></td>
<td><strong>C</strong></td>
<td><strong>FORTRAN</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Inquiry</strong></td>
<td>HE5_GDgridinfo</td>
<td>Returns dimensions of grid and X-Y coordinates of corners</td>
<td>2-226</td>
</tr>
<tr>
<td></td>
<td>HE5_GDprojinfo</td>
<td>Returns all GCTP projection information</td>
<td>2-248</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdiminfo</td>
<td>Retrieves size of specified dimension.</td>
<td>2-210</td>
</tr>
<tr>
<td></td>
<td>HE5_GDcompinfo</td>
<td>Retrieves compression info about a field</td>
<td>2-186</td>
</tr>
<tr>
<td></td>
<td>HE5_GDfieldinfo</td>
<td>Retrieves information about a specific field in the grid</td>
<td>2-215</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqgrid</td>
<td>Retrieves number and names of grids in file</td>
<td>2-236</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqfidalias</td>
<td>Returns information about data fields &amp; aliases defined in grid</td>
<td>2-235</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqdatatype</td>
<td>Returns data type information about specified fields in grid</td>
<td>2-229</td>
</tr>
<tr>
<td></td>
<td>HE5_GDattrinfo</td>
<td>Returns information about grid attributes</td>
<td>2-182</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgrnpattrinfo</td>
<td>Returns info about a grid group attribute</td>
<td>2-227</td>
</tr>
<tr>
<td></td>
<td>HE5_GDlocattrinfo</td>
<td>Returns information about a Data Field’s local attribute(s)</td>
<td>2-242</td>
</tr>
<tr>
<td></td>
<td>HE5_GDorgininfo</td>
<td>Returns information about grid pixel origin</td>
<td>2-246</td>
</tr>
<tr>
<td></td>
<td>HE5_GDpixreginfo</td>
<td>Returns pixel registration info for given grid</td>
<td>2-247</td>
</tr>
<tr>
<td><strong>Subset</strong></td>
<td>HE5_GDdefboxregion</td>
<td>Defines region of interest by latitude/longitude</td>
<td>2-190</td>
</tr>
<tr>
<td></td>
<td>HE5_GDregioninfo</td>
<td>Returns information about a defined region</td>
<td>2-257</td>
</tr>
<tr>
<td></td>
<td>HE5_GDextractregion</td>
<td>Read a region of interest from a field</td>
<td>2-214</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdeftimeperiod</td>
<td>Define a time period of interest</td>
<td>2-205</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdefvrtregion</td>
<td>Define a region of interest by vertical field</td>
<td>2-207</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgetpixels</td>
<td>Get row/columns for lon/lat pairs</td>
<td>2-222</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgetpixvalues</td>
<td>Get field values for specified pixels</td>
<td>2-224</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinterpolate</td>
<td>Perform bilinear interpolation on a grid field</td>
<td>2-240</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdupregion</td>
<td>Duplicate a region or time period</td>
<td>2-213</td>
</tr>
<tr>
<td><strong>Tiling</strong></td>
<td>HE5_GDdeffile</td>
<td>Define a tiling scheme</td>
<td>2-202</td>
</tr>
<tr>
<td></td>
<td>HE5_GDtileinfo</td>
<td>Retrieve tiling information</td>
<td>2-264</td>
</tr>
<tr>
<td><strong>Utility</strong></td>
<td>HE5_GDij2ll</td>
<td>convert (i,j) coordinates to (lon,lat) for a grid</td>
<td>2-364</td>
</tr>
<tr>
<td></td>
<td>HE5_GDij2lji</td>
<td>convert (lon,lat) coordinates to (i,j) for a grid</td>
<td>2-367</td>
</tr>
<tr>
<td></td>
<td>HE5_GDrs2ll</td>
<td>Convert (r,s) coordinates to (lon,lat) for EASE grid</td>
<td>2-370</td>
</tr>
<tr>
<td><strong>External Data Sets</strong></td>
<td>HE5_GDsetextdata</td>
<td>Set external data set</td>
<td>2-262</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgetextdata</td>
<td>Get external data set</td>
<td>2-220</td>
</tr>
<tr>
<td><strong>Dimension Scale</strong></td>
<td>HE5_GDsetdimscale</td>
<td>Sets dimension scale for a field dimension within the grid</td>
<td>2-260</td>
</tr>
<tr>
<td></td>
<td>HE5_GDgetdimscale</td>
<td>Gets dimension scale for a field dimension within the grid</td>
<td>2-218</td>
</tr>
<tr>
<td></td>
<td>HE5_GDwritedscaleattr</td>
<td>Writes/Updates a dimension scale attribute in a specific grid</td>
<td>2-268</td>
</tr>
<tr>
<td></td>
<td>HE5_GDreaddscaleattr</td>
<td>Reads a dimension scale attribute from a specific dimension</td>
<td>2-250</td>
</tr>
<tr>
<td></td>
<td>HE5_GDinqdscaleattr</td>
<td>Retrieves information about the attributes defined for a specific dimension scale</td>
<td>2-232</td>
</tr>
<tr>
<td></td>
<td>HE5_GDdscaleattrinfo</td>
<td>Returns information about attribute(s) in a specific dimension scale</td>
<td>2-212</td>
</tr>
</tbody>
</table>

### 6.3.2 File Identifiers

As with all HDF-EOS interfaces, file identifiers in the GD interface are of hid_t HDF5 type, each uniquely identifying one open data file. They are not interchangeable with other file identifiers created with other interfaces.
6.3.3 Grid Identifiers

Before a grid data set is accessed, it is identified by a name which is assigned to it upon its creation. The name is used to obtain a grid identifier. After a grid data set has been opened for access, it is uniquely identified by its grid identifier.

6.4 Programming Model

The programming model for accessing a grid object through the GD interface is as follows:

1. Open the file and initialize the GD interface by obtaining a file ID from a file name.
2. Open OR create a grid object by obtaining a grid ID from a grid name.
3. Perform desired operations on the data set.
4. Close the grid object by disposing of the grid ID.
5. Terminate grid access to the file by disposing of the file ID.

In this example we open the HDF-EOS grid file, "Grid.he5". Assuming that this file may not exist, we are using the H5F_ACC_TRUNC access code. The "HE5_GDopen" function returns the grid file ID, gdfid which is used to identify the file in subsequent calls to the HDF-EOS library functions.

```
gdfid = HE5_GDopen("Grid.he5", H5F_ACC_TRUNC);
/* Create "UTM" Grid
Use default spheroid (Clarke 1866 - spherecode = 0) */
GDid   = HE5_GDcreate(gdfid, "UTMGrid", xdim, ydim, uplft, lowrgt);
/* Define projection */
status = HE5_GDdefproj(GDid, HE5_GCTP_UTM, zonecode, spherecode, projparm);
/* Define "Time" Dimension */
status = HE5_GDdefdim(GDid, "Time", 10);
/* Define "Unlimited" Dimension */
status = HE5_GDdefdim(GDid, "Unlim", H5S_UNLIMITED);
/* Close the grid interface */
status = HE5_GDdetach(GDid);
/* Close the grid file */
status = HE5_GDclose(gdfid);
```

To access several files at the same time, a calling program must obtain a separate ID for each file to be opened. Similarly, to access more than one grid object, a calling program must obtain a
separate grid ID for each object. For example, to open two objects stored in two files, a program would execute the following series of C function calls:

```c
gdfid_1 = HE5_GDopen(filename_1, access_mode);
gdid_1 = HE5_GDattach(gdfid_1, grid_name_1);
gdfid_2 = HE5_GDopen(filename_2, access_mode);
gdid_2 = HE5_GDattach(gdfid_2, grid_name_2);
Optional operations>
status = HE5_GDdetach(gdid_1);
status = HE5_GDclose(gdfid_1);
status = HE5_GDdetach(gdid_2);
status = HE5_GDclose(gdfid_2);
```

Because each file and grid object is assigned its own identifier, the order in which files and objects are accessed is very flexible. However, it is very important that the calling program individually discard each identifier before terminating. Failure to do so can result in empty or, even worse, invalid files being produced.

### 6.5 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 describes codes used by the package.

#### 6.5.1 GCTP Projection Codes

The following GCTP projection codes are used in the grid API described in Section 7 below:

- **GCTP_GEO** (0) Geographic
- **GCTP_UTM** (1) Universal Transverse Mercator
- **GCTP_LAMCC** (4) Lambert Conformal Conic
- **GCTP_PS** (6) Polar Stereographic
- **GCTP_POLYC** (7) Polyconic
- **GCTP_TM** (9) Transverse Mercator
- **GCTP_LAMAZ** (11) Lambert Azimuthal Equal Area
- **GCTP_HOM** (20) Hotine Oblique Mercator
- **GCTP_SOM** (22) Space Oblique Mercator
- **GCTP_GOOD** (24) Interrupted Goode Homolosine
- **GCTP_ISINUS** (99/31) Integerized Sinusoidal Projection*
- **GCTP_CEA** (97) Cylindrical Equal-Area (for EASE grid with corners in meters)**
- **GCTP_BCEA** (98) Cylindrical Equal-Area (for EASE grid with grid corners in packed degrees, DMS)**
* The Integerized Sinusoidal Projection is not part of the original GCTP package. It has been added by ECS. See *Level-3 SeaWiFS Data Products: Spatial and Temporal Binning Algorithms*. Additional references are provided in Section 2.

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

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

Note that other projections supported by GCTP will be adapted for HDF-EOS Version 5 new user requirements are surfaced. For further details on the GCTP projection package, please refer to Section 6.3.5 and Appendix G of the EOSDIS Evolution and Development Project Toolkit Users Guide, January 2012, (333-EED-001, Revision 01.)

### 6.5.2 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-180W</td>
</tr>
</tbody>
</table>
### 6.5.3 GCTP Spheroid Codes

<table>
<thead>
<tr>
<th>Spheroid Code</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>
<tr>
<td>Modified Everest</td>
<td>(11)</td>
</tr>
<tr>
<td>WGS 84</td>
<td>(12)</td>
</tr>
<tr>
<td>Southeast Asia</td>
<td>(13)</td>
</tr>
<tr>
<td>Australian National</td>
<td>(14)</td>
</tr>
<tr>
<td>Krassovsky</td>
<td>(15)</td>
</tr>
<tr>
<td>Hough</td>
<td>(16)</td>
</tr>
<tr>
<td>Mercury 1960</td>
<td>(17)</td>
</tr>
<tr>
<td>Modified Mercury 1968</td>
<td>(18)</td>
</tr>
<tr>
<td>Sphere of Radius 6370997m</td>
<td>(19)</td>
</tr>
<tr>
<td>Sphere of Radius 6371228m</td>
<td>(20)</td>
</tr>
<tr>
<td>Sphere of Radius 6371007.181m</td>
<td>(21)</td>
</tr>
</tbody>
</table>
### 6.5.4 Projection Parameters

**Table 6-2. Projection Transformation Package Projection Parameters**

<table>
<thead>
<tr>
<th>Code &amp; Projection Id</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>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>CentMer</td>
<td>OriginLat</td>
<td>FE</td>
<td>FN</td>
<td></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></td>
<td>OriginLat</td>
<td>FE</td>
<td>FN</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></td>
<td>FE</td>
<td>FN</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></td>
<td>FE</td>
<td>FN</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>97 CEA utilized by EASE grid (see notes)</td>
<td>SMajor</td>
<td>SMinor</td>
<td>CentMer</td>
<td>TrueScale</td>
<td>FE</td>
<td>FN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98 BCEA utilized by EASE grid (see notes)</td>
<td>SMajor</td>
<td>SMinor</td>
<td>CentMer</td>
<td>TrueScale</td>
<td>FE</td>
<td>FN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99 Integerized Sinusoidal</td>
<td>Sphere</td>
<td></td>
<td>CentMer</td>
<td></td>
<td>FE</td>
<td>FN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Table 6-3. Projection Transformation Package Projection Parameters Elements**

<table>
<thead>
<tr>
<th>Code &amp; Projection Id</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Geographic</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>
</tr>
<tr>
<td>4 Lambert Conformal C</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>
</tr>
<tr>
<td>7 Polyconic</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>
</tr>
<tr>
<td>11 Lambert Azimuthal</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>
</tr>
<tr>
<td>20 Hotin Oblique Merc B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>one</td>
</tr>
<tr>
<td>22 Space Oblique Merc A</td>
<td>PSRev</td>
<td>Srat</td>
<td>PFlag</td>
<td></td>
<td>zero</td>
</tr>
<tr>
<td>22 Space Oblique Merc B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>one</td>
</tr>
<tr>
<td>24 Interrupted Goode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97 CEA utilized by EASE grid (see Notes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98 BCEA utilized by EASE grid (see Notes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31 &amp; 99 Integerized Sinusoidal</td>
<td>NZone</td>
<td>RFlag</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. It is recommended that explicit value, rather than zero, is used for Smajor.

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

**Sphere** Radius of reference sphere. If zero, 6370997 meters is used. It is recommended that explicit value, rather than zero, is used for Sphere.

**STDPR1** Latitude of the first standard parallel.
STDPR2  Latitude of the second standard parallel.
CentMer  Longitude of the central meridian.
OriginLat Latitude of the projection origin.
FE       False easting in the same units as the semi-major axis.
FN       False northing in the same units as the semi-major axis.
TrueScale Latitude of true scale.
LongPol  Longitude down below pole of map.
Factor   Scale factor at central meridian (Transverse Mercator) or center of projection (Hotine Oblique Mercator).
CentLon  Longitude of center of projection.
CenterLat Latitude of center of projection.
Long1    Longitude of first point on center line (Hotine Oblique Mercator, format A).
Long2    Longitude of second point on center line (Hotine Oblique Mercator, format A).
Lat1     Latitude of first point on center line (Hotine Oblique Mercator, format A).
Lat2     Latitude of second point on center line (Hotine Oblique Mercator, format A).
AziAng   Azimuth angle east of north of center line (Hotine Oblique Mercator, format B).
AzmthPt  Longitude of point on central meridian where azimuth occurs (Hotine Oblique Mercator, format B).
IncAng   Inclination of orbit at ascending node, counter-clockwise from equator (SOM, format A).
AscLong  Longitude of ascending orbit at equator (SOM, format A).
PSRev    Period of satellite revolution in minutes (SOM, format A).
SRat     Satellite ratio to specify the start and end point of x,y values on earth surface (SOM, format A -- for Landsat use 0.5201613).
PFlag    End of path flag for Landsat:  0 = start of path, 1 = end of path (SOM, format A).
Satnum   Landsat Satellite Number (SOM, format B).
Path     Landsat Path Number (Use WRS-1 for Landsat 1, 2 and 3 and WRS-2 for Landsat 4 and 5.) (SOM, format B).
Nzone    Number of equally spaced latitudinal zones (rows); must be two or larger and even.
RFlag    Right justify columns flag is used to indicate what to do in zones with an odd number of columns. If it has a value of 0 or 1, it indicates the extra column is on the right (zero) 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 (DDDMMmSSS.SS) format.

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

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

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

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

Grid Dimensions:

Width 1383
Height 586

Map Origin:

Column (r0) 691.0
Row (s0) 292.5
Latitude 0.0
Longitude 0.0

Grid Extent:
Minimum Latitude 86.72S
Maximum Latitude 86.72N
Minimum Longitude 180.00W
Maximum Longitude 180.00E
Actual grid cell size 25.067525km

Grid coordinates (r,s) start in the upper left corner at cell (0,0), with r increasing to the right and s increasing downward.

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

Grid Dimensions:
Width 1383
Height 586

Grid Extent:
UpLeft Latitude 86.72
LowRight Latitude –86.72
UpLeft Longitude –180.00
LowRight Longitude 180.00

Projection Parameters:
1) 6371.2280/25.067525 = 254.16263
2) 6371.2280/25.067525 = 254.16263
5) 0.0
6) 30000000.0
7) 691.0
Also one may create **12.5 km global EASE grid** using:

Grid Dimensions:
- Width 2766
- Height 1171

Grid Extent:
- UpLeft Latitude 85.95
- LowRight Latitude -85.95
- UpLeft :Longitude -179.93
- LowRight Longitude 180.07

Projection Parameters:
1) \( \frac{6371.2280}{(25.067525/2)} = 508.325253 \)
2) \( \frac{6371.2280}{(25.067525/2)} = 508.325253 \)
5) 0.0
6) 30000000.0
7) 1382.0
8) -585.0

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

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

This Section contains code examples of usage of the HDF-EOS Library specified in Sections 4, 5 and 6 of this document. These examples assume that the user is not using the SDP Toolkit and is writing applications for use outside of ECS. Examples of SDP Toolkit usage in conjunction with HDF-EOS is presented in Section 8.

Note: The examples in this document are code fragments, designed to show users how to create HDF-EOS data structures. Some of the examples in this version have not yet undergone thorough inspections and checks for ECS software standard compliance.

7.1 Point Examples

This section contains several examples in C and FORTRAN which demonstrate the usage of most of the functions in the Point interface.

7.1.1 Creating a Simple Point

The following C and FORTRAN programs each create, define, and write a simple Point data set (level) to an HDF-EOS file using the HDF-EOS Point interface.

7.1.1.1 A C Example of Creating a Simple Point

Example 1

/*
 * In this example we will (1) open an HDF-EOS file, and (2) create
 * three point objects within the file.
 */
#include <HE5_HdfEosDef.h>

/* he5_pt_setup */

main()
{
    herr_t status;
    hid_t ptfid, PTid1, PTid2, PTid3;

    /* Open the HDF-EOS point file, "Point.he5". Assuming that this file
     * may not exist, we are using the "H5F_ACC_TRUNC" access code.
     * The "HE5_PTopen" function returns the point file ID, ptfid,
     * which is used to identify the file in subsequent calls to the
     */
HDF-EOS library functions.

/*
 ptfid = HE5_PTop("Point.he5", H5F_ACC_TRUNC);

/* Set up the point structures */
 PTid1 = HE5_PTcreate(ptfid, "Simple Point");
 PTid2 = HE5_PTcreate(ptfid, "FixedBuoy Point");
 PTid3 = HE5_PTcreate(ptfid, "FloatBuoy Point");

/* Close the point interface */
 status = HE5_PTdetach(PTid1);
 status = HE5_PTdetach(PTid2);
 status = HE5_PTdetach(PTid3);

/* Close the point file */
 status = HE5_PTclose(ptfid);

 return;
}

Example 2

/*
  * In this example we will open the "Point.he5" HDF-EOS file
  * and define the point levels.
  */

#include <HE5_HdfEosDef.h>

/* he5_pt_definelevels */

main()
{
  herr_t            status = FAIL;
  int               i;
  hid_t             ptfid = FAIL, PTid = FAIL;
  HE5_CmpDTSinfo    dtsinfo;

typedef struct
  {
    double          time;
    float           concentr[4];
    char            spec[8];
  } InputData1;

typedef struct

typedef struct
{
    double time;
    float rain;
    float temp;
    char id[8];
} InputData3;

typedef struct
{
    char label[10];
    int date;
    int weight;
    char id[8];
} InputData4;

typedef struct
{
    double time;
    double lon;
    double lat;
    float rain;
    float temp;
    char id[8];
} InputData5;

/*
 * We first open the HDF-EOS point file, "Point.he5". Because this file
 * already exist and we wish to write to it, we use the H5F_ACC_RDWR access
 * code in the open statement. The PTopen routine returns the point file
 * id, ptfid, which is used to identify the file in subsequent routines.
 */
ptfid = HE5_PTopen("Point.he5", H5F_ACC_RDWR);
if (ptfid != FAIL)
{
    /* Simple Point */
    /* ------------------ */
    PTid = HE5_PTattach(ptfid, "Simple Point");

    /* Set up level data structure */
    /* ------------------------------- */
    dtsinfo.nfields = 3;
    dtsinfo.rank[0] = 1;
    dtsinfo.rank[1] = 1;
    dtsinfo.rank[2] = 1;
Here we use the HDF5 Macro “HOFFSET” to calculate
the data member offsets within the C data structure

```c
    dtsinfo.offset[0] = HOFFSET(InputData1, time);
    dtsinfo.offset[1] = HOFFSET(InputData1, concentr);
    dtsinfo.offset[2] = HOFFSET(InputData1, spec);

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

    for (i = 0; i < 3; i++)
        {  
            dtsinfo.fieldname[i] = (char *)calloc(64, sizeof(char));
        }

    strcpy(dtsinfo.fieldname[0],"Time");
    strcpy(dtsinfo.fieldname[1],"Concentration");
    strcpy(dtsinfo.fieldname[2],"Species");

    dtsinfo.dims[0][0] = 1;
    dtsinfo.dims[1][0] = 4;
    dtsinfo.dims[2][0] = 8;

    dtsinfo.datasize = (size_t)sizeof(InputData1);

    status = HE5_PTdeflevel(PTid, "Sensor", &dtsinfo);
    printf("Status returned by HE5_PTdeflevel() : %d \n", status);

    for (i = 0; i < 3; i++)
        free(dtsinfo.fieldname[i]);

    status = HE5_PTdetach(PTid);

    /* Fixed Buoy Point */
    /* ----------------- */
    PTid = HE5_PTattach(ptfid, "FixedBuoy Point");

    /* Set up level data structure */
    /* ----------------------------- */
    dtsinfo.nfields = 5;

    dtsinfo.rank[0] = 1;
    dtsinfo.rank[1] = 1;
    dtsinfo.rank[2] = 1;
    dtsinfo.rank[3] = 1;
    dtsinfo.rank[4] = 1;

    /* Here we use the HDF5 Macro “HOFFSET” to calculate */
    /* the data member offsets within the C data structure */

    dtsinfo.offset[0] = HOFFSET(InputData2, label);
    dtsinfo.offset[1] = HOFFSET(InputData2, lon);
    dtsinfo.offset[2] = HOFFSET(InputData2, lat);
    dtsinfo.offset[3] = HOFFSET(InputData2, date);
    dtsinfo.offset[4] = HOFFSET(InputData2, id);
```

dtsinfo.dtype[0] = H5T_NATIVE_CHAR;
dtsinfo.dtype[1] = H5T_NATIVE_DOUBLE;
dtsinfo.dtype[2] = H5T_NATIVE_DOUBLE;
dtsinfo.dtype[4] = H5T_NATIVE_CHAR;

for (i = 0; i < 5; i++)
{
    dtsinfo.fieldname[i] = (char *)calloc(64, sizeof(char));
}

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

dtsinfo.dims[0][0] = 8;
dtsinfo.dims[1][0] = 1;
dtsinfo.dims[2][0] = 1;
dtsinfo.dims[3][0] = 1;
dtsinfo.dims[4][0] = 8;

dtsinfo.datasize = (size_t)sizeof(InputData2);

status = HE5_PTdeflevel(PTid, "Desc-Loc", &dtsinfo);
printf("Status returned by HE5_PTdeflevel() : %d \n", status);

for (i = 0; i < 5; i++)
    free(dtsinfo.fieldname[i]);

    /* Set up level data structure */
    /* --------------------------- */
    dtsinfo.nfields = 4;
    dtsinfo.rank[0] = 1;
    dtsinfo.rank[1] = 1;
    dtsinfo.rank[2] = 1;
    dtsinfo.rank[3] = 1;

    /* Here we use the HDF5 Macro "HOFFSET" to calculate */
    /* the data member offsets withing the C data structure */
    dtsinfo.offset[0] = HOFFSET(InputData3, time);
    dtsinfo.offset[1] = HOFFSET(InputData3, rain);
    dtsinfo.offset[2] = HOFFSET(InputData3, temp);
    dtsinfo.offset[3] = HOFFSET(InputData3, id);

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

for (i = 0; i < 4; i++)
{
    dtsinfo.fieldname[i] = (char *)calloc(64, sizeof(char));
}
strcpy(dtsinfo.fieldname[0],"Time");
strcpy(dtsinfo.fieldname[1],"Rainfall");
strcpy(dtsinfo.fieldname[2],"Temperature");
strcpy(dtsinfo.fieldname[3],"ID");

dtsinfo.dims[0][0] = 1;
dtsinfo.dims[1][0] = 1;
dtsinfo.dims[2][0] = 1;
dtsinfo.dims[3][0] = 8;

dtsinfo.datasize = (size_t)sizeof(InputData3);

status = HE5_PTdeflevel(PTid, "Observations", &dtsinfo);
printf("Status returned by HE5_PTdeflevel() : %d\n", status);

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

status = HE5_PTdeflinkage(PTid, "Desc-Loc", "Observations", "ID");
printf("Status returned by HE5_PTdeflinkage() : %d\n", status);

status = HE5_PTdetach(PTid);
printf("Status returned by HE5_PTdetach() : %d\n", status);

/* Floating Buoy Point */
/* ------------------- */
PTid = HE5_PTattach(ptfid, "FloatBuoy Point");

/* Set up level data structure */
/* ----------------------------- */
dtsinfo.nfields = 4;

dtsinfo.rank[0] = 1;
dtsinfo.rank[1] = 1;
dtsinfo.rank[2] = 1;
dtsinfo.rank[3] = 1;

/* Here we use the HDF5 Macro "HOFFSET" to calculate */
/* the data member offsets within the C data structure */

dtsinfo.offset[0] = HOFFSET(InputData4, label);
dtsinfo.offset[1] = HOFFSET(InputData4, date);
dtsinfo.offset[2] = HOFFSET(InputData4, weight);
dtsinfo.offset[3] = HOFFSET(InputData4, id);

dtsinfo.dtype[0] = H5T_NATIVE_CHAR;
dtsinfo.dtype[1] = H5T_NATIVE_INT;
dtsinfo.dtype[3] = H5T_NATIVE_CHAR;

for (i = 0; i < 4; i++)
    { 
        dtsinfo.fieldname[i] = (char *)calloc(64, sizeof(char));
    }

strcpy(dtsinfo.fieldname[0],"Label");
strcpy(dtsinfo.fieldname[1],"DeployDate");
strcpy(dtsinfo.fieldname[2],"Weight");
strcpy(dtsinfo.fieldname[3],"ID");
dtsinfo.dims[0][0] = 8;
dtsinfo.dims[1][0] = 1;
dtsinfo.dims[2][0] = 1;
dtsinfo.dims[3][0] = 8;
dtsinfo.datasize = (size_t)sizeof(InputData4);
status = HE5_PTdeflevel(PTid, "Description", &dtsinfo);
printf("Status returned by HE5_PTdeflevel() : %d \n", status);
for (i = 0; i < 4; i++)
    free(dtsinfo.fieldname[i]);

/* Define Data Level */
/* Set up level data structure */
/* ------------------------ */
dtsinfo.nfields = 6;
dtsinfo.rank[0] = 1;
dtsinfo.rank[1] = 1;
dtsinfo.rank[2] = 1;
dtsinfo.rank[3] = 1;
dtsinfo.rank[4] = 1;
dtsinfo.rank[5] = 1;

/* Here we use the HDF5 Macro "HOFFSET" to calculate */
/* the data member offsets within the C data structure */
dtsinfo.offset[0] = HOFFSET(InputData5, time);
dtsinfo.offset[1] = HOFFSET(InputData5, lon);
dtsinfo.offset[2] = HOFFSET(InputData5, lat);
dtsinfo.offset[3] = HOFFSET(InputData5, rain);
dtsinfo.offset[4] = HOFFSET(InputData5, temp);
dtsinfo.offset[5] = HOFFSET(InputData5, id);
dtsinfo.dtype[0] = H5T_NATIVE_DOUBLE;
dtsinfo.dtype[1] = H5T_NATIVE_DOUBLE;
dtsinfo.dtype[2] = H5T_NATIVE_DOUBLE;
dtsinfo.dtype[3] = H5T_NATIVE_FLOAT;
dtsinfo.dtype[4] = H5T_NATIVE_FLOAT;
dtsinfo.dtype[5] = H5T_NATIVE_CHAR;
for (i = 0; i < 6; i++)
{
    dtsinfo.fieldname[i] = (char *)calloc(64, sizeof(char));
}
strcpy(dtsinfo.fieldname[0],"Time");
strcpy(dtsinfo.fieldname[1],"Longitude");
strcpy(dtsinfo.fieldname[2],"Latitude");
strcpy(dtsinfo.fieldname[3],"Rainfall");
strcpy(dtsinfo.fieldname[4],"Temperature");
strcpy(dtsinfo.fieldname[5],"ID");
dtsinfo.dims[0][0] = 1;
dtsinfo.dims[1][0] = 1;
Example 3

#include <HE5_HdfEosDef.h>

main()
{
    FILE            *fp;
    herr_t          status = FAIL;
    int             n, date, wgt, IntAttr = 9999;
    hid_t           ptfid = FAIL, PTid = FAIL;
    hsize_t         count[1];
    size_t          datasize = 0;

    dtsinfo.dims[2][0] = 1;
    dtsinfo.dims[3][0] = 1;
    dtsinfo.dims[4][0] = 1;
    dtsinfo.dims[5][0] = 8;

    dtsinfo.datasize = (size_t)sizeof(InputData5);

    status = HE5_PTdeflevel(PTid, "Measurements", &dtsinfo);
    printf("Status returned by HE5_PTdeflevel() : %d \n", status);

    for (i = 0; i < 6; i++)
        free(dtsinfo.fieldname[i]);

    status = HE5_PTdeflinkage(PTid, "Description", "Measurements", "ID");
    printf("Status returned by HE5_PTdeflinkage() : %d \n", status);

    status = HE5_PTdetach(PTid);
    status = HE5_PTclose(ptfid);
}

return 0;
}
float conc[4], rain, temp, flt = -7.5;
double time, lon, lat;
char spc[8], desc[16], id[2];
typedef struct
{
    double Time;
    float Conc[4];
    char Spc[8];
} CmpData_1;

typedef struct
{
    char Label[8];
    double Lon;
    double Lat;
    int Date;
    char Id[8];
} CmpData_2;

typedef struct
{
    double Time;
    float Rain;
    float Temp;
    char Id[8];
} CmpData_3;

typedef struct
{
    char Label[10];
    int Date;
    int Weight;
    char Id[8];
} CmpData_4;

typedef struct
{
    double Time;
    double Lon;
    double Lat;
    float Rain;
    float Temp;
    char Id[8];
} CmpData_5;

CmpData_1 datbuf_1[20];
CmpData_2 datbuf_2[5];
CmpData_3 datbuf_3[25];
CmpData_4 datbuf_4[5];
CmpData_5 datbuf_5[30];

/* Open the HDF-EOS file, "Point.he5" */
ptfid = HE5_PTopen("Point.he5", H5F_ACC_RDWR);

PTid = HE5_PTattach(ptfid, "Simple Point");

fp = fopen("simple.txt", "r");

n = 0;
while(fscanf(fp, "%lf %f %f %f %f %s", &time, &conc[0], &conc[1], &conc[2], &conc[3], spc) != -1)
{
    datbuf_1[n].Time    = time;
    datbuf_1[n].Conc[0] = conc[0];
    datbuf_1[n].Conc[1] = conc[1];
    datbuf_1[n].Conc[2] = conc[2];
    datbuf_1[n].Conc[3] = conc[3];
    datbuf_1[n].Spc = memmove(datbuf_1[n].Spc, spc, sizeof(char)*strlen(spc));
    datbuf_1[n].Spc[strlen(spc)] = 0;
    n++;
}
fclose(fp);

status = HE5_PTwritelevel(PTid, 0, count, &datasize, datbuf_1);
printf("Status returned by HE5_PTwritelevel() :   %d \n", status);

status = HE5_PTdetach(PTid);
printf("Status returned by HE5_PTdetach() :       %d \n", status);

PTid = HE5_PTattach(ptfid, "FixedBuoy Point");

fp = fopen("fixedBuoy0.txt", "r");

n = 0;
while(fscanf(fp, "%s %lf %lf %d %s", desc, &lon, &lat, &date, id) != -1)
{
    strcpy(datbuf_2[n].Label, desc);
    datbuf_2[n].Lon     = lon;
    datbuf_2[n].Lat     = lat;
    datbuf_2[n].Date    = date;
    datbuf_2[n].Id = memmove(datbuf_2[n].Id, id, sizeof(char)*strlen(id));
    datbuf_2[n].Id[strlen(id)] = 0;
    n++;
}
fclose(fp);

datasize = (size_t)sizeof(CmpData_2);
count[0] = n;

status = HE5_PTwritelevel(PTid, 0, count, &datasize, datbuf_2);
printf("Status returned by HE5_PTwritelevel() : %d \n", status);

/* Write Second (1st) Level */
/* ------------------------ */
fp = fopen("fixedBuoy1.txt", "r");

n = 0;
while(fscanf(fp, "%lf %f %f %s", &time, &rain, &temp, id) != -1)
{
    datbuf_3[n].Time = time;
    datbuf_3[n].Rain = rain;
    datbuf_3[n].Temp = temp;
    memmove(datbuf_3[n].Id, id, sizeof(char)*strlen(id));
    datbuf_3[n].Id[strlen(id)] = 0;
    n++;
}
fclose(fp);

datasize = (size_t)sizeof(CmpData_3);
count[0] = n;

status = HE5_PTwritelevel(PTid, 1, count, &datasize, datbuf_3);
printf("Status returned by HE5_PTwritelevel() : %d \n", status);

count[0] = 1;
status = HE5_PTwriteattr(PTid, "GlobalAttr_Integer", H5T_NATIVE_INT, count,
&IntAttr);
printf("Status returned by HE5_PTwriteattr() : %d \n", status);

status = HE5_PTdetach(PTid);
printf("Status returned by HE5_PTdetach() : %d \n", status);

/* Write to Floating Buoy Point */
/* ---------------------------- */
PTid = HE5_PTattach(ptfid, "FloatBuoy Point");

/* Write First (0th) Level */
/* ------------------------- */
fp = fopen("floatBuoy0.txt", "r");

n = 0;
while(fscanf(fp, "%s %d %d %s", desc, &date, &wgt, id) != -1)
{
    strcpy(datbuf_4[n].Label, desc);
    datbuf_4[n].Date = date;
    datbuf_4[n].Weight = wgt;
    memmove(datbuf_4[n].Id, id, sizeof(char)*strlen(id));
    datbuf_4[n].Id[strlen(id)] = 0;
    n++;
}
fclose(fp);

datasize = (size_t)sizeof(CmpData_4);
count[0] = n;

status = HE5_PTwritelevel(PTid, 0, count, &datasize, datbuf_4);
printf("Status returned by HE5_PTwritelevel() : %d \n", status);

/* Write Second (1th) Level */
/* ------------------------ */
fp = fopen("floatBuoy1.txt", "r");

n = 0;
while(fscanf(fp, "%lf %lf %lf %f %f %s", &time, &lon, &lat, &rain, &temp, id) != -1)
{
    datbuf_5[n].Time = time;
    datbuf_5[n].Lon = lon;
    datbuf_5[n].Lat = lat;
    datbuf_5[n].Rain = rain;
    datbuf_5[n].Temp = temp;
    memmove(datbuf_5[n].Id, id,sizeof(char)*strlen(id));
    datbuf_5[n].Id[strlen(id)] = 0;
    n++;
}
fclose(fp);

datasize = (size_t)sizeof(CmpData_5);
count[0] = n;

status = HE5_PTwritelevel(PTid, 1, count, &datasize, datbuf_5);
printf("Status returned by HE5_PTwritelevel() : %d \n", status);

status = HE5_PTwriteattr(PTid, "GlobalAttr", H5T_NATIVE_FLOAT, count, &flt);
printf("Status returned by HE5_PTwriteattr() : %d \n", status);

status = HE5_PTdetach(PTid);
printf("Status returned by HE5_PTdetach() : %d \n", status);

status = HE5_PTclose(ptfid);
printf("Status returned by HE5_PTclose() : %d \n", status);
return 0;
}

Example 4

/*
 * In this example we will read data from a specified level
 */
/* he5_pt_readdata */
#include <HE5_HdfEosDef.h>

main()
{
  herr_t status = FAIL; /* return status variable */
  hid_t ptfid = FAIL; /* HDFEOS Point file ID */
  hid_t PTid = FAIL; /* Point structure ID */
  int i, j; /* Loop indices */
  int nflds = FAIL; /* Number of level fields */
  int IntAttr; /* Integer attribute value */
  long nattr; /* Number of attributes */
  long strbufsize; /* Size of attribute list buffer */
  hsize_t nrecs = 0; /* Number of records in a level */
  hid_t *nt = (hid_t *)NULL; /* Data type class ID */
  size_t datasize = 0; /* Size (in bytes) of data to read */
  HE5_CmpDTSinfo level; /* Level information data structure */
  HE5_CmpDTSinfo inInfo; /* Input information data structure */

  /* User-defined structure to read level data to */
  typedef struct {
    double   time;
    float    con[4];
    char     spec[8];
  } Sensor;

  Sensor *s;

  /* Open the HDF-EOS file, "Point.he5" */
  /* ----------------------------- */
  ptfid = HE5_PTopen("Point.he5", H5F_ACC_RDONLY);
  printf("File ID returned by HE5_PTopen() : %d \n", ptfid);

  /* Read Simple Point */
  /* ----------------- */
  PTid = HE5_PTattach(ptfid, "Simple Point");
  printf("Point ID returned by HE5_PTattach() : %d \n", PTid);

  /* Get level information */
  /* ---------------------- */
  status = HE5_PTlevelinfo(PTid, 0, &level);
  printf("Status returned by HE5_PTlevelinfo() : %d \n", status);

  nflds = level.nfields;
  printf("Number of fields in level: %d \n", nflds);

  for (i = 0; i < nflds; i++)
    level.fieldname[i] = (char *)calloc(64, sizeof(char));

  status = HE5_PTlevelinfo(PTid, 0, &level);
  printf("Status returned by HE5_PTlevelinfo() : %d \n", status);
for (i = 0; i < nflds; i++)
{
    printf("Field name: %s \n", level.fieldname[i]);
    printf("Field rank: %d \n", level.rank[i]);
    for (j = 0; j < level.rank[i]; j++)
        printf("Field dims: %d \n", (int)level.dims[i][j]);
    printf("Field class: %d \n", level.dclass[i]);
}

/* Get the number of records in level */
/* ---------------------------------- */
nrecs = HE5_Ptnrecs(PTid, 0);
printf("Number of records in level: %lu \n", (unsigned long)nrecs);

/* Set the data size */
/* ------------------ */
datasize = (size_t)sizeof(Sensor);

/* Allocate memory for the output data structure */
/* --------------------------------------------- */
s = (Sensor *)calloc(nrecs, sizeof(Sensor));

/* Populate input information structure */
/* ------------------------------------ */
inInfo.nfields = nflds;
inInfo.datasize = (size_t)sizeof(Sensor);
inInfo.rank[0] = 1;
inInfo.rank[1] = 1;
inInfo.rank[2] = 1;

/* Here we use the HDF5 Macro "HOFFSET" to calculate */
/* the data member offsets withing the C data structure */
inInfo.offset[0] = HOFFSET(Sensor, time);
inInfo.offset[1] = HOFFSET(Sensor, con);
inInfo.offset[2] = HOFFSET(Sensor, spec);
inInfo.dtype[0] = H5T_NATIVE_DOUBLE;
inInfo.dtype[1] = H5T_NATIVE_FLOAT;
inInfo.dtype[2] = H5T_NATIVE_CHAR;
inInfo.dclass[0] = H5T_NO_CLASS;
inInfo.dclass[1] = H5T_NO_CLASS;
inInfo.dclass[2] = H5T_NO_CLASS;
inInfo.dims[0][0] = 1;
inInfo.dims[1][0] = 4;
inInfo.dims[2][0] = 8;

for( i = 0; i < nflds; i++)
{
    inInfo.fieldname[i] = (char *)calloc(HE5_HDFE_NAMBUFSIZE, sizeof(char));
    strcpy(inInfo.fieldname[i], level.fieldname[i]);
}

/* Read the level data */
/* ------------------ */
status = HE5_PTreadlevel(PTid, 0, &inInfo, &datasize, s);
printf("Status returned by HE5_PTreadlevel() : %d \n", status);
for (i = 0; i < nrecs; i++)
    printf("%f %f %f %f %s\n", s[i].time, s[i].con[0], s[i].con[1], s[i].con[2], s[i].con[3], s[i].spec);

/* Release memory */
/* ----------------*/
for (i = 0; i < nflds; i++)
    { free(level.fieldname[i]);
      free(inInfo.fieldname[i]);
    }
free(s);

status = HE5_PTdetach(PTid);
printf("Status returned by HE5_PTdetach() : %d \n", status);

PTid = HE5_PTattach(ptfid, "FixedBuoy Point");
printf("Point ID returned by HE5_PTattach() : %d \n", PTid);

/* Read Fixed Buoy Point Attributes */
/* -------------------------------*/
nt = (hid_t *)calloc(1, sizeof(hid_t));
nattr = HE5_PTinqattrs(PTid, NULL, &strbufsize);
status = HE5_PTreadattr(PTid, "GlobalAttr_Integer", &IntAttr);
printf("Status returned by HE5_PTreadattr() : %d \n", status);
printf("\n");
printf("Integer attribute value: %d\n", IntAttr);
free(nt);

status = HE5_PTdetach(PTid);
printf("Status returned by HE5_PTdetach() : %d \n", status);

status = HE5_PTclose(ptfid);
printf("Status returned by HE5_PTclose() : %d \n", status);
return 0;
}

Example 5

/*
 * In this example we will update a specified level
 */

/* he5_pt_updatelevels */
#include <HE5_HdfEosDef.h>

main()
{ }
herr_t  status = FAIL;
int     i, j;
hid_t   ptfid = FAIL;
hid_t   PTid1 = FAIL;
hssize_t recs[32];
hsize_t nrec;
typedef struct
{
    double   Time;
    float    Conc[4];
    char     Spc[8];
} CmpData_1;

CmpData_1 buf_1;
buf_1.Time     = 13131313.0;
buf_1.Conc[0]  = 1.11;
buf_1.Conc[1]  = 2.22;
strcpy(buf_1.Spc,"AM");

/* ---------------------------------------------- */
/* NOTE: To update all records, set "nrec" => 0 or "recs" => NULL */
/*       the data buffer should be properly populated */
/* ---------------------------------------------- */

/* Open the HDF-EOS file, "Point.he5" */
/* ------------------------------------- */
ptfid = HE5_PTopen("Point.he5", H5F_ACC_RDWR);
if (ptfid != FAIL)
{
    PTid1 = HE5_PTattach(ptfid, "Simple Point");
    if (PTid1 != FAIL)
    {
        nrec    = 1;
        recs[0] = 0;
        status = HE5_PTupdatelevel(PTid1, 0, "Concentration", nrec, recs, &buf_1);
        printf("Status returned by HE5_PTupdatelevel() : %d \n", status);
    }
    status = HE5_PTdetach(PTid1);
    printf("Status returned by HE5_PTdetach() : %d \n", status);
}

status = HE5_PTClose(ptfid);
return 0;
}
7.1.1.2 A FORTRAN Example of a Simple Point Creation

**Example 1**

```fortran
program he5_pt_setupF_32
implicit none
integer status
integer ptfid
integer ptid1, ptid2, ptid3
integer he5_ptopen
integer he5_ptcreate
integer he5_ptdetach
integer he5_ptclose
integer HE5F_ACC_TRUNC
parameter (HE5F_ACC_TRUNC=102)
```

We first open the HDF-EOS point file, "Point.he5". Because this file does not already exist, we use the HE5F_ACC_TRUNC access code in the open statement. The PTopen routine returns the point file id, ptfid, which is used to identify the file in subsequent routines in the library.

```fortran
ptfid = he5_ptopen('Point.he5', HE5F_ACC_TRUNC)
write(*,*) 'File ID returned by he5_ptopen(): ', ptfid
```

```fortran
ptid1 = he5_ptcreate(ptfid, "Simple Point")
write(*,*) 'Point ID returned by he5_ptcreate(): ', ptid1
ptid2 = he5_ptcreate(ptfid, "FixedBuoy Point")
write(*,*) 'Point ID returned by he5_ptcreate(): ', ptid2
ptid3 = he5_ptcreate(ptfid, "FloatBuoy Point")
write(*,*) 'Point ID returned by he5_ptcreate(): ', ptid3
```

We now close the point interface with the he5_ptdetach routine. This step is necessary to properly store the point information within the file.
status = he5_ptdetach(ptid1)
write(*,*) 'Status returned by he5_ptdetach():  ',status
status = he5_ptdetach(ptid2)
write(*,*) 'Status returned by he5_ptdetach():  ',status
status = he5_ptdetach(ptid3)
write(*,*) 'Status returned by he5_ptdetach():  ',status
status = he5_ptclose(ptfid)
write(*,*) 'Status returned by he5_ptclose():  ',status

stop
end

Example 2

program he5_pt_definelevelsF_32
implicit none
integer status
integer ptfid
integer ptid
integer he5_ptopen
integer he5_ptattach
integer he5_ptdeflevel
integer he5_ptdeflinkage
integer he5_ptdetach
integer he5_ptclose
integer rank_1(3)
integer rank_2(5)
integer rank_3(4)
integer rank_4(4)
integer rank_5(6)
integer dtype_1(3)
integer dtype_2(5)
integer dtype_3(4)
integer dtype_4(4)
integer dtype_5(6)
integer array_1(3)
integer array_2(5)
integer array_3(4)
integer array_4(4)
integer array_5(6)

integer*4 dims_1(3,1)
integer*4 dims_2(5,1)
integer*4 dims_3(4,1)
integer*4 dims_4(4,1)
integer*4 dims_5(6,1)

character*240 fieldlist1
character*240 fieldlist2
character*240 fieldlist3
Open the HDF point file, "Point.he5"
-----------------------------

cfid = he5_ptopen('Point.he5', HE5F_ACC_RDWR)
write(*,*) 'File ID returned by he5_ptopen(): ', cfid

Read Simple Point
----------------

cfid = HE5_PTattach(cfid, "Simple Point")
write(*,*) 'Point ID returned by he5_ptattach(): ', cfid

Populate input information structure
-------------------------------------

levelname = 'Sensor'
rk_1(1) = 1
rk_1(2) = 1
rk_1(3) = 1
fieldlist1 = 'Time, Concentration, Species'
dtype_1(1) = HE5F_NATIVE_DOUBLE
dtype_1(2) = HE5F_NATIVE_FLOAT
dtype_1(3) = HE5F_NATIVE_CHAR
array_1(1) = 0
array_1(2) = 1
array_1(3) = 1
dims_1(1,1) = 1
dims_1(2,1) = 4
dims_1(3,1) = 8

status = he5_ptdeflevel(cfid, levelname, rk_1, fieldlist1, 1, dtype_1, array_1)
write(*,*) 'Status returned by he5_ptdeflevel(): ', status
c.....Close out the point interface
  status = he5_ptdetach(ptid)
  write(*,*), 'Status returned by he5_ptdetach(): ', status

  Read Fixed Buoy Point
  ---------------------
  ptid = HE5_PTattach(ptfid, "FixedBuoy Point")
  write(*,*), 'Point ID returned by he5_ptattach(): ', ptid

  Populate input information structure
  ------------------------------------
  levelname = 'Desc-Loc'
  rank_2(1) = 1
  rank_2(2) = 1
  rank_2(3) = 1
  rank_2(4) = 1
  rank_2(5) = 1
  fieldlist2 = 'Label,Longitude,Latitude,DeployDate,ID'
  dtype_2(1) = HE5T_NATIVE_CHAR
  dtype_2(2) = HE5T_NATIVE_DOUBLE
  dtype_2(3) = HE5T_NATIVE_DOUBLE
  dtype_2(4) = HE5T_NATIVE_INT
  dtype_2(5) = HE5T_NATIVE_CHAR
  array_2(1) = 1
  array_2(2) = 0
  array_2(3) = 0
  array_2(4) = 0
  array_2(5) = 1
  dims_2(1,1) = 8
  dims_2(2,1) = 1
  dims_2(3,1) = 1
  dims_2(4,1) = 1
  dims_2(5,1) = 8
  status = he5_ptdeflevel(ptid, levelname, rank_2, fieldlist2, dims_2, dtype_2, array_2)
  write(*,*), 'Status returned by he5_ptdeflevel(): ', status

  Populate input information structure
  ------------------------------------
  levelname = 'Observations'
  rank_3(1) = 1
  rank_3(2) = 1
  rank_3(3) = 1
  rank_3(4) = 1
  fieldlist3 = 'Time, Rainfall, Temperature, ID'
  dtype_3(1) = HE5T_NATIVE_DOUBLE
```plaintext
dtype_3(2)  =  HE5T_NATIVE_FLOAT
dtype_3(3)  =  HE5T_NATIVE_FLOAT
dtype_3(4)  =  HE5T_NATIVE_CHAR

array_3(1)  =  0
array_3(2)  =  0
array_3(3)  =  0
array_3(4)  =  1

dims_3(1,1)  =  1
dims_3(2,1)  =  1
dims_3(3,1)  =  1
dims_3(4,1)  =  8

status = he5_ptdeflevel(ptid, levelname, rank_3, fieldlist3,
                        dims_3, dtype_3, array_3)
write(*,*) 'Status ID returned by he5_ptdeflevel(): ',status

parent    = 'Desc-Loc'
child     = 'Observations'
linkfield = 'ID'

status = he5_ptdeflinkage(ptid, parent, child, linkfield)
write(*,*) 'Status ID returned by he5_ptdeflinkage(): ',status

.....Close out the point interface
status = he5_ptdetach(ptid)
write(*,*) 'Status returned by he5_ptdetach(): ',status

c Read Floating Buoy Point
c  ---------------------
ptid = HE5 PTattach(ptfid, "FloatBuoy Point")
write(*,*) 'Point ID returned by he5_ptattach(): ',ptid

c Populate input information structure
c  -------------------------------

levelname = 'Description'
rank_4(1)  =  1
rank_4(2)  =  1
rank_4(3)  =  1
rank_4(4)  =  1

fieldlist4 = 'Label,DeployDate,Weight,ID'

dtype_4(1)  =  HE5T_NATIVE_CHAR
dtype_4(2)  =  HE5T_NATIVE_INT
dtype_4(3)  =  HE5T_NATIVE_INT
dtype_4(4)  =  HE5T_NATIVE_CHAR

array_4(1)  =  1
array_4(2)  =  0
array_4(3)  =  0
array_4(4)  =  1

dims_4(1,1)  =  8
```
dims_4(2,1) = 1
 dims_4(3,1) = 1
 dims_4(4,1) = 8

status = he5_ptdeflevel(ptid, levelname, rank_4, fieldlist4,
 dims_4, dtype_4, array_4)
write(*,*) 'Status returned by he5_ptdeflevel(): ',status

--- Populate input information structure ---

levelname = 'Measurements'
rank_5(1) = 1
rank_5(2) = 1
rank_5(3) = 1
rank_5(4) = 1
rank_5(5) = 1
rank_5(6) = 1
fieldlist5 = 'Time,Longitude,Latitude,Rainfall,Temperature,Id'
dtype_5(1) = HE5T_NATIVE_DOUBLE
dtype_5(2) = HE5T_NATIVE_DOUBLE
dtype_5(3) = HE5T_NATIVE_DOUBLE
dtype_5(4) = HE5T_NATIVE_FLOAT
dtype_5(5) = HE5T_NATIVE_FLOAT
dtype_5(6) = HE5T_NATIVE_CHAR
array_5(1) = 0
array_5(2) = 0
array_5(3) = 0
array_5(4) = 0
array_5(5) = 0
array_5(6) = 1
dims_5(1,1) = 1
dims_5(2,1) = 1
dims_5(3,1) = 1
dims_5(4,1) = 1
dims_5(5,1) = 1
dims_5(6,1) = 8

status = he5_ptdeflevel(ptid, levelname, rank_5, fieldlist5,
 dims_5, dtype_5, array_5)
write(*,*) 'Status returned by he5_ptdeflevel(): ',status

parent = 'Description'
child = 'Measurements'
linkfield = 'Id'

status = he5_ptdeflinkage(ptid, parent, child, linkfield)
write(*,*) 'Status ID returned by he5_ptdeflinkage(): ',status

--- Close out the point interface ---
status = he5_ptdetach(ptid)
write(*,*) 'Status returned by he5_ptdetach(): ',status
status = he5_ptclose(ptfid)
write(*,*) 'Status returned by he5_ptclose(): ', status

stop
dend

Example 3

program he5_pt_writedataF_32
implicit none
integer status
integer ptfid
integer ptid
integer he5_ptopen
integer he5_ptattach
integer he5_ptwritelevel
integer he5_ptfort2c
integer he5_ptwrbcckptr
integer he5_ptwrfdptr
integer he5_ptwriteattr
integer he5_ptdetach
integer he5_ptclose
integer i
integer rank, datatype
integer attr

integer*4 n
integer*4 count(1)
integer*4 dimens(2)
integer*4 fortcount(8), ntype
real*4 flt
character*80 fieldname, attrname
c.....used by Simple Point
real*8 time_tt
real*8 time(15)
real*4 concentration_tt(4)
real*4 conc(15,4)
real*4 outconc(4,15)
character*8 spc_tt
character*8 spc(15)
c.....used by FixedBuoy Point - Level 0
character*8 desc_tt
character*8 desc(3)
real*8 lon_tt
real*8 lon(3)
real*8 lat_tt
real*8 lat(3)
integer*4     date_tt
integer*4     date(3)
character*8   id_tt
character*8   id(3)

c.....used by FixedBuoy Point - Level 1
real*8        time3_tt
real*8        time3(20)
real*4        rain_tt
real*4        rain(20)
real*4        temp_tt
real*4        temp(20)
character*8   id3_tt
character*8   id3(20)

c.....used by FloatBuoy Point - Level 0
character*8   desc4_tt
character*8   desc4(3)
integer*4     date4_tt
integer*4     date4(3)
integer*4     wgt_tt
integer*4     wgt(3)
character*8   id4_tt
character*8   id4(3)

c.....used by FloatBuoy Point - Level 1
real*8        time5_tt
real*8        time5(25)
real*8        lon5_tt
real*8        lon5(25)
real*8        lat5_tt
real*8        lat5(25)
real*4        rain5_tt
real*4        rain5(25)
real*4        temp5_tt
real*4        temp5(25)
character*8   id5_tt
character*8   id5(25)

integer       HE5T_NATIVE_INT
parameter     (HE5T_NATIVE_INT=0)
integer       HE5T_NATIVE_FLOAT
parameter     (HE5T_NATIVE_FLOAT=10)
integer       HE5T_NATIVE_DOUBLE
parameter     (HE5T_NATIVE_DOUBLE=11)
integer       HE5T_NATIVE_CHAR
parameter     (HE5T_NATIVE_CHAR=56)

integer       HE5F_ACC_RDWR
parameter     (HE5F_ACC_RDWR=100)

c     Open the HDF point file, "Point.he5"
c------------------------------------
ptfid = he5_ptopen('Point.he5',HE5F_ACC_RDWR)
write(*,*) 'File ID returned by he5_ptopen(): ',ptfid

c     Do Simple Point
c----------------
ptid = he5_ptattach(ptfid, "Simple Point")
write(*,*) 'Point ID returned by he5_ptattach(): ',ptid

c Read Simple Point
-----------------
open(unit=1, file='simple.txt', status='OLD')
n = 0
do 10 i=1,1000
   read(1, 110, end=100) time_tt, concentration_tt(1),
   concentration_tt(2),
   concentration_tt(3),
   concentration_tt(4),
   spc_tt
   time(i) = time_tt
   conc(i,1) = concentration_tt(1)
   conc(i,2) = concentration_tt(2)
   conc(i,3) = concentration_tt(3)
   conc(i,4) = concentration_tt(4)
   spc(i) = spc_tt
   n = n + 1
10 continue
100 close(unit=1)
c.....Convert array to 'C' order
dimens(1) = 15
dimens(2) = 4
rank = 2
datatype = HE5T_NATIVE_FLOAT
c This is a call to utility routine reversing dimension order from
FORTRAN to C
status = he5_ptfort2c(dimens, rank, datatype, conc, outconc)
write(*,*) 'Status returned by he5_ptfort2c(): ',status

c.....Call to writelevel
fieldname = 'Time'
datatype = HE5T_NATIVE_DOUBLE
status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, time)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

c.....Call to writelevel
fieldname = 'Concentration'
datatype = HE5T_NATIVE_CHAR
status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, outconc)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

fieldname = 'Species'
datatype = HE5T_NATIVE_CHAR
status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, spc)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

Close out the point interface
status = he5_ptdetach(ptid)
write(*,*) 'Status returned by he5_ptdetach(): ',status

Do FixedBuoy Point
------------------
ptid = he5_ptattach(ptfid, "FixedBuoy Point")
write(*,*) 'Point ID returned by he5_ptattach(): ',ptid

Read FixedBuoy Point
--------------------
open(unit=1, file='fixedBuoy0.txt', status='OLD')
n = 0
do 20 i=1,1000
  read(1, 210, end=200) desc_tt, lon_tt, lat_tt, date_tt, id_tt
  desc(i) = desc_tt
  lon(i) = lon_tt
  lat(i) = lat_tt
  date(i) = date_tt
  id(i) = id_tt
  n = n + 1
20 continue
200 close(unit=1)
210 format(A8,F13.7,F13.7,I7,1X,A8)
count(1) = n
fieldname = 'Label'
datatype = HE5T_NATIVE_CHAR
status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, desc)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

fieldname = 'Longitude'
datatype = HE5T_NATIVE_DOUBLE
status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, lon)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

fieldname = 'Latitude'
datatype = HE5T_NATIVE_DOUBLE
status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, lat)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

fieldname = 'DeployDate'
datatype = HE5T_NATIVE_INT
status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, date)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

fieldname      = 'ID'
datatype       = HE5T_NATIVE_CHAR

status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, id)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

Read FixedBuoy Point - Level 1
----------------------------------
open(unit=1, file='fixedBuoy1.txt', status='OLD')
n = 0
do 30 i=1,1000
   read(1, 310, end=300) time3_tt, rain_tt, temp_tt, id3_tt
   time3(i)     = time3_tt
   rain(i)      = rain_tt
   temp(i)      = temp_tt
   id3(i)       = id3_tt
   n = n + 1
30 continue
300 close(unit=1)
310 format(F13.2,F8.1,F8.2,3X,A8)
count(1) = n

fieldname      = 'Time'
datatype       = HE5T_NATIVE_DOUBLE
status = he5_ptwritelevel(ptid, 1, count, fieldname, datatype, time3)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

fieldname      = 'Rainfall'
datatype       = HE5T_NATIVE_FLOAT
status = he5_ptwritelevel(ptid, 1, count, fieldname, datatype, rain)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

fieldname      = 'Temperature'
datatype       = HE5T_NATIVE_FLOAT
status = he5_ptwritelevel(ptid, 1, count, fieldname, datatype, temp)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status

fieldname      = 'ID'
datatype       = HE5T_NATIVE_CHAR
status = he5_ptwritelevel(ptid, 1, count, fieldname, datatype, id3)
write(*,*) 'Status returned by he5_ptwritelevel(): ',status
c.....Write forward and backward pointers
status = he5_ptwrbckptr(ptid,1)
write(*,*), 'Status returned by he5_ptwrbckptr(): ', status

status = he5_ptwrfwdptr(ptid,1)
write(*,*), 'Status returned by he5_ptwrfwdptr(): ', status

c Write attributes to "Fixed Buoy Point"
--------------------------------------
attrname       = 'GlobalAttribute_int'
type           = HE5T_NATIVE_INT
fortcount(1)   = 1
attr            = 9999

status = he5_ptwriteattr(ptid,attrname,type,fortcount,attr)
write(*,*), 'Status returned by he5_ptwriteattr(): ', status

c.....Close out the point interface
status = he5_ptdetach(ptid)
write(*,*), 'Status returned by he5_ptdetach(): ', status

c Do FloatBuoy Point
--------------------
ptid = he5_ptattach(ptfid, "FloatBuoy Point")
write(*,*), 'Point ID returned by he5_ptattach(): ', ptid

c Read FloatBuoy Point - Level 0
-----------------------------------
open(unit=1, file='floatBuoy0.txt', status='OLD')

n = 0
do 40 i=1,1000
   read(1, 410, end=400) desc4_tt, date4_tt, wgt_tt, id4_tt
   desc4(i)     = desc4_tt
   date4(i)     = date4_tt
   wgt(i)       = wgt_tt
   id4(i)       = id4_tt
   n = n + 1
40 continue

400 close(unit=1)
410 format(A8,I8,I7,2X,A8)

count(1) = n

fieldname      = 'Label'
datatype       = HE5T_NATIVE_CHAR

status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, desc4)
write(*,*), 'Status returned by he5_ptwritelevel(): ', status

fieldname      = 'DeployDate'
datatype       = HE5T_NATIVE_INT
status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, date4)
write(*,*),'Status returned by he5_ptwritelevel(): ',status

fieldname = 'Weight'
datatype = HE5T_NATIVE_INT

status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, wgt)
write(*,*),'Status returned by he5_ptwritelevel(): ',status

fieldname = 'ID'
datatype = HE5T_NATIVE_CHAR

status = he5_ptwritelevel(ptid, 0, count, fieldname, datatype, id4)
write(*,*),'Status returned by he5_ptwritelevel(): ',status

Read FixedBuoy Point – Level 1

open(unit=1, file='floatBuoy1.txt', status='OLD')

n = 0
do 50 i=1,1000
    read(1, 510, end=500) time5_tt, lon5_tt, lat5_tt, rain5_tt, temp5_tt, id5_tt
    time5(i) = time5_tt
    lon5(i) = lon5_tt
    lat5(i) = lat5_tt
    rain5(i) = rain5_tt
    temp5(i) = temp5_tt
    id5(i) = id5_tt
    n = n + 1
50 continue

500 close(unit=1)
510 format(F13.1,F13.6,F13.6,F8.1,F8.2,3X,A8)

count(1) = n

fieldname = 'Time'
datatype = HE5T_NATIVE_DOUBLE

status = he5_ptwritelevel(ptid, 1, count, fieldname, datatype, time5)
write(*,*),'Status returned by he5_ptwritelevel(): ',status

fieldname = 'Longitude'
datatype = HE5T_NATIVE_DOUBLE

status = he5_ptwritelevel(ptid, 1, count, fieldname, datatype, lon5)
write(*,*),'Status returned by he5_ptwritelevel(): ',status

fieldname = 'Latitude'
datatype = HE5T_NATIVE_DOUBLE

status = he5_ptwritelevel(ptid, 1, count, fieldname,
Example 4

```fortran
program he5_pt_readdataF_32

ldatatype, lat5)
write(*,*) 'Status returned by he5_ptwritelevel(): ', status

fieldname   = 'Rainfall'
datatype    = HE5T_NATIVE_FLOAT

status = he5_ptwritelevel(ptid, 1, count, fieldname, ldatatype, rain5)
write(*,*) 'Status returned by he5_ptwritelevel(): ', status

fieldname   = 'Temperature'
datatype    = HE5T_NATIVE_FLOAT

status = he5_ptwritelevel(ptid, 1, count, fieldname, ldatatype, temp5)
write(*,*) 'Status returned by he5_ptwritelevel(): ', status

fieldname   = 'ID'
datatype    = HE5T_NATIVE_CHAR

status = he5_ptwritelevel(ptid, 1, count, fieldname, ldatatype, id5)
write(*,*) 'Status returned by he5_ptwritelevel(): ', status

c.....Write forward and backward pointers
status = he5_ptwrbckptr(ptid,1)
write(*,*) 'Status returned by he5_ptwrbckptr(): ', status

status = he5_ptwrfwdptr(ptid,1)
write(*,*) 'Status returned by he5_ptwrfwdptr(): ', status

c Write attributes to "Float Buoy Point"
-----------------------------
attrname       = 'GlobalAttribute_float'
ntype          = HE5T_NATIVE_FLOAT
fortcount(1)   = 1
flt            = -7.5

status   = he5_ptwriteattr(ptid,attrname, lntype,fortcount,flt)
write(*,*) 'Status returned by he5_ptwriteattr(): ', status

c.....Close out the point interface
status = he5_ptdetach(ptid)
write(*,*) 'Status returned by he5_ptdetach(): ', status

status = he5_ptclose(ptfid)
write(*,*) 'Status returned by he5_ptclose(): ', status

stop
end
```
implicit none

integer status
integer ptfid
integer ptid
integer he5_ptopen
integer he5_ptattach
integer he5_ptreadlevel
integer he5_ptlevelinfo
integer he5_ptnrecs
integer he5_ptnlevels
integer he5_ptnfields
integer he5_ptc2fort
integer he5_ptingattrs
integer he5_ptreadattr
integer he5_ptdetach
integer he5_ptclose
integer i
integer nflds
integer level
integer arr_rank
integer datatype
integer dtype(3)
integer attr
integer nrecs
integer nlevels

integer*4 dimens(2)
integer*4 datasize
integer*4 rank_tt(3)
integer*4 offset_tt(3)
integer*4 dtype_tt(3)
integer*4 dim_sizes_tt(3)
integer*4 nattr
integer*4 strbufsize

character*80 levelname
character*80 fieldname(3)
character*80 attrname
character*240 fieldlist
character*240 attrlist
character*8 spec(15)

real*4 con(4,15)
real*4 outcon(15,4)
real*8 time(15)

integer HE5T_NATIVE_INT
parameter (HE5T_NATIVE_INT=0)
integer HE5T_NATIVE_FLOAT
parameter (HE5T_NATIVE_FLOAT=10)
integer HE5T_NATIVE_DOUBLE
parameter (HE5T_NATIVE_DOUBLE=11)
integer HE5T_NATIVE_CHAR
parameter (HE5T_NATIVE_CHAR=56)
integer HE5F_ACC_RDONLY
parameter (HE5F_ACC_RDONLY=101)
Open the HDF point file, "Point.he5"

ptfid = he5_ptopen('Point.he5', HE5F_ACC_RDONLY)
write(*,*) 'File ID returned by he5_ptopen(): ', ptfid

Read Simple Point

ptid = he5_ptattach(ptfid, "Simple Point")
write(*,*) 'Point ID returned by he5_ptattach(): ', ptid

Get level information

level = 0

status = he5_ptlevelinfo(ptid, level, levelname, rank_tt, 1fieldlist, dim_sizes_tt, datasize, offset_tt, dtype_tt)
write(*,*) 'Status returned by he5_ptlevelinfo(): ', status

Get the number of records in level

level = 0
nrecs = he5_ptnrecs(ptid, level)
print *, 'Number of records in level: ', nrecs

nlevels = he5_ptnlevels(ptid)
print *, 'Number of levels in Point data set: ', nlevels

nflds = he5_ptnfields(ptid, level, fieldlist, strbufsize)
print *, 'Number of fields in level: ', nflds

Populate input information structure

dtype(1) = HE5T_NATIVE_DOUBLE
dtype(2) = HE5T_NATIVE_FLOAT
dtype(3) = HE5T_NATIVE_CHAR

Read the level data and print out

fieldname(1) = 'Time'

status = he5_ptreadlevel(ptid, 0, fieldname(1), 1dtype(1), time)
write(*,*) 'Status returned by he5_ptreadlevel(): ', status
write(*,*) 'time array: '
    do i = 1, nrecs
        print *, time(i)
    end do

fieldname(2) = 'Concentration'

status = he5_ptreadlevel(ptid, 0, fieldname(2), 1dtype(2), con)
write(*,*) 'Status returned by he5_ptreadlevel(): ', status

Convert 'C' array to Fortran order
dimens(1) = 15
dimens(2) = 4
arr_rank = 2
datatype = HE5T_NATIVE_FLOAT

c This is a call to utility routine reversing dimension order from
C to FORTRAN

status = he5_ptc2fort(dimens, arr_rank, datatype, con,
1outcon)
write(*,*) 'Status returned by he5_ptc2fort(): ',status

write(*,*) 'outcon array: '
do i = 1,nrecs
print *,outcon(i,1),outcon(i,2),outcon(i,3),outcon(i,4)
end do

fieldname(3) = 'Species'

status = he5_ptreadlevel(ptid, 0, fieldname(3),
datatype(3), spec)
write(*,*) 'Status returned by he5_ptreadlevel(): ',status

write(*,*) 'spec array: '
do i = 1,nrecs
print *,spec(i)
end do

C.....Close out the point interface
status = he5_ptdetach(ptid)
write(*,*) 'Status returned by he5_ptdetach(): ',status

C Read FixedBuoy Point
--------------------
ptid = he5_ptattach(ptfid, "FixedBuoy Point")
write(*,*) 'Point ID returned by he5_ptattach(): ',ptid

C Global Attributes
-----------------
attrname = 'GlobalAttribute_int'

print *,
print *, 'Global Attribute: '
nattr = he5_ptinqattrs(ptid,attrlist,strbufsize)
print *, 'Number of attributes: ',nattr
print *, 'Attribute list: ',attrlist
print *, 'Size (in bytes) of attribute list: ',strbufsize

status = he5_ptreadattr(ptid,attrname,attr)
write(*,*) 'Status returned by he5_ptreadattr(): ',status
print *, 'Attribute value: ',attr

C.....Close out the point interface
status = he5_ptdetach(ptid)
write(*,*) 'Status returned by he5_ptdetach(): ',status

status = he5_ptclose(ptfid)
write(*,*) 'Status returned by he5_ptclose(): ',status
Example 5

program he5_pt_updatelevelsF_32
implicit none
integer status
integer ptfid
integer ptid
integer level
integer he5_ptopen
integer he5_ptattach
integer he5_ptupdatelevel
integer he5_ptdetach
integer he5_ptclose
integer dtype(3)
integer*4 recs(32)
integer*4 nrec
real*4 conc_tt(4)
real*8 time_tt
character*8 spc_tt
character*80 fieldname
integer HE5F_ACC_RDWR
parameter (HE5F_ACC_RDWR=100)
integer HE5T_NATIVE_DOUBLE
parameter (HE5T_NATIVE_DOUBLE=11)
integer HE5T_NATIVE_FLOAT
parameter (HE5T_NATIVE_FLOAT=10)
integer HE5T_NATIVE_CHAR
parameter (HE5T_NATIVE_CHAR=56)

open the HDF point file, "Point.he5"

ptfid = he5_ptopen('Point.he5',HE5F_ACC_RDWR)
write(*,*) 'File ID returned by he5_ptopen(): ',ptfid

Read Simple Point

ptid = he5_ptattach(ptfid, "Simple Point")
write(*,*) 'Point ID returned by he5_ptattach(): ',ptid
dtype(1) = HE5T_NATIVE_DOUBLE
dtype(2) = HE5T_NATIVE_FLOAT
Example 6

```plaintext
program he5_pt_datainfoF_32
implicit none
integer status
integer ptfid
integer ptid
```

```
dtype(3) = HE5T_NATIVE_CHAR
nrec = 1
recs(1) = 0
level = 0
fieldname = 'Concentration'
conc_tt(1) = 1.11
conc_tt(2) = 2.22
conc_tt(3) = 3.33
conc_tt(4) = 4.44
status = he5_ptupdatelevel(ptid, level, fieldname, nrec, lrecs, dtype(2), conc_tt)
write(*,*) 'Status returned by he5_ptupdatelevel(): ',status
fieldname = 'Time'
time_tt = 13131313.0
status = he5_ptupdatelevel(ptid, level, fieldname, nrec, lrecs, dtype(1), time_tt)
write(*,*) 'Status returned by he5_ptupdatelevel(): ',status
fieldname = 'Species'
spc_tt = 'AM'
status = he5_ptupdatelevel(ptid, level, fieldname, nrec, lrecs, dtype(3), spc_tt)
write(*,*) 'Status returned by he5_ptupdatelevel(): ',status

c......Close out the point interface
status = he5_ptdetach(ptid)
write(*,*) 'Status returned by he5_ptdetach(): ',status

status = he5_ptclose(ptfid)
write(*,*) 'Status returned by he5_ptclose(): ',status

stop
end
```
integer       he5_ptopen
integer       he5_ptattach
integer       he5_ptinqdatatype
integer       he5_ptdetach
integer       he5_ptclose
integer       dtype
classid
integer       order
integer       fieldgroup

define
integer*4     size
cchar*1       null_char_0
cchar*80      fieldname

cchar*80      attrname

integer       HE5F_ACC_RDONLY
parameter     (HE5F_ACC_RDONLY=101)

integer       HE5T_NATIVE_INT
parameter     (HE5T_NATIVE_INT=0)
integer       HE5T_NATIVE_DOUBLE
parameter     (HE5T_NATIVE_DOUBLE=11)
integer       HE5T_NATIVE_FLOAT
parameter     (HE5T_NATIVE_FLOAT=10)
integer       HE5T_NATIVE_CHAR
parameter     (HE5T_NATIVE_CHAR=56)

integer       HE5_HDFE_GEOGROUP
parameter     (HE5_HDFE_GEOGROUP=0)
integer       HE5_HDFE_DATAGROUP
parameter     (HE5_HDFE_DATAGROUP=1)
integer       HE5_HDFE_ATTRGROUP
parameter     (HE5_HDFE_ATTRGROUP=2)
integer       HE5_HDFE_GRPATTRGROUP
parameter     (HE5_HDFE_GRPATTRGROUP=3)
integer       HE5_HDFE_LOCATTRGROUP
parameter     (HE5_HDFE_LOCATTRGROUP=4)
integer       HE5_HDFE_PROFRGROUP
parameter     (HE5_HDFE_PROFRGROUP=5)

null_char_0  = '0'
c
Open the HDF point file, "Point.he5"
-------------------------------------
ptfid = he5_ptopen('Point.he5',HE5F_ACC_RDONLY)
write(*,*) 'File ID returned by he5_ptopen():  ',ptfid

c
Read Simple Point
-----------------
ptid = he5_ptattach(ptfid, "FixedBuoy Point")
write(*,*) 'Point ID returned by he5_ptattach():  ',ptid

fieldgroup = HE5_HDFE_DATAGROUP
fieldname  = 'Observations'

status = he5_ptinqdatatype(ptid,fieldname,null_char_0,fieldgroup,
dtype,classid,order,size)
print *, 'Status returned from he5_ptinqdatatype(): ', status
print *, 'datatype: ', dtype
print *, 'class ID: ', classid
print *, 'order: ', order
print *, 'size: ', size

fieldgroup = HE5_HDFE_ATTRGROUP
attrname   = 'GlobalAttribute_int'

status = he5_ptinqdatatype(ptid, null_char_0, attrname, fieldgroup,
                         ldtype, classid, order, size)
print *, 'Status returned from he5_ptinqdatatype(): ', status
print *, 'datatype: ', dtype
print *, 'class ID: ', classid
print *, 'order: ', order
print *, 'size: ', size

fieldgroup = HE5_HDFE_GRPATTRGROUP
attrname   = 'GroupAttribute'

status = he5_ptinqdatatype(ptid, null_char_0, attrname, fieldgroup,
                         ldtype, classid, order, size)
print *, 'Status returned from he5_ptinqdatatype(): ', status
print *, 'datatype: ', dtype
print *, 'class ID: ', classid
print *, 'order: ', order
print *, 'size: ', size

fieldname = 'Observations'
fieldgroup = HE5_HDFE_LOCATTRGROUP
attrname   = 'LocalAttribute'

status = he5_ptinqdatatype(ptid, fieldname, attrname, fieldgroup,
                         ldtype, classid, order, size)
print *, 'Status returned from he5_ptinqdatatype(): ', status
print *, 'datatype: ', dtype
print *, 'class ID: ', classid
print *, 'order: ', order
print *, 'size: ', size

c.....Close out the point interface
status = he5_ptdetach(ptid)
write(*,*) 'Status returned by he5_ptdetach(): ', status

status = he5_ptclose(ptfid)
write(*,*) 'Status returned by he5_ptclose(): ', status

stop
end
7.2 Swath Examples

This section contains several examples of the use of the Swath interface from both C and FORTRAN programs. First, there are simple examples in C and FORTRAN which demonstrate the use of most of the functions in the Swath interface.

7.2.1 Creating a Simple Swath

The following C and FORTRAN programs each create, define, and write a simple Swath data set to an HDF-EOS file using the HDF-EOS Swath interface.

7.2.1.1 A C Example of a Simple Swath Creation

Example 1

```c
/*
 * In this program we (1) open an HDF-EOS file, (2) create the swath
 * interface within the file, and (3) define the swath field dimensions
 */

/* he5_sw_setup */

#include <HE5_HdfEosDef.h>

main()
{
  herr_t status = FAIL;
  int i, j;
  hid_t swfid = FAIL;
  hid_t SWid = FAIL;
  long indx[12] = {0,1,3,6,7,8,11,12,14,24,32,39};

  /* Open a new HDF-EOS swath file, "Swath.he5" */
  /* ---------------------------------------------- */
  swfid = HE5_SWopen("Swath.he5", H5F_ACC_TRUNC);
  printf("File ID returned by HE5_SWopen(): %d \n", swfid);

  /* Create the swath, "Swath1", within the file */
  /* ------------------------------------------- */
  SWid = HE5_SWcreate(swfid, "Swath1");
  printf("Swath ID returned by HE5_SWcreate(): %d \n", SWid);

  /* Define dimensions and specify their sizes */
  /* ------------------------------------------ */
  status = HE5_SWdefdim(SWid, "GeoTrack", 20);
  printf("Status returned by HE5_SWdefdim(): %d \n", status);

  /* Other C examples go here */
}
```

status = HE5_SWdefdim(SWid, "GeoXtrack", 10);
printf("Status returned by HE5_SWdefdim(): %d \n", status);

status = HE5_SWdefdim(SWid, "Res2tr", 40);
printf("Status returned by HE5_SWdefdim(): %d \n", status);

status = HE5_SWdefdim(SWid, "Res2xtr", 20);
printf("Status returned by HE5_SWdefdim(): %d \n", status);

status = HE5_SWdefdim(SWid, "Bands", 15);
printf("Status returned by HE5_SWdefdim(): %d \n", status);

status = HE5_SWdefdim(SWid, "IndxTrack", 12);
printf("Status returned by HE5_SWdefdim(): %d \n", status);

status = HE5_SWdefdim(SWid, "ProfDim", 4);
printf("Status returned by HE5_SWdefdim(): %d \n", status);

status = HE5_SWdefdim(SWid, "Unlim", H5S_UNLIMITED);
printf("Status returned by HE5_SWdefdim(): %d \n", status);

/* Define Dimension Mapping */
*--------------------------*
status = HE5_SWdefdimmap(SWid, "GeoTrack", "Res2tr", 0, 2);
printf("Status returned by HE5_SWdefdimmap(): %d \n", status);

status = HE5_SWdefdimmap(SWid, "GeoXtrack", "Res2xtr", 1, 2);
printf("Status returned by HE5_SWdefdimmap(): %d \n", status);

/* Define Indexed Mapping */
*------------------------*
status = HE5_SWdefidxmap(SWid, "IndxTrack", "Res2tr", indx);
printf("Status returned by HE5_SWdefidxmap(): %d \n", status);

/* Close the swath interface */
*-------------------------*/
status = HE5_SWdetach(SWid);
printf("Status returned by HE5_SWdetach(): %d \n", status);
Example 2

/* Close the swath file */
/* ------------------- */
status = HE5_SWclose(swfid);
printf("Status returned by HE5_SWCLOSE(): %d \n", status);

return 0;
}

#include <HE5_HdfEosDef.h>
define RANK 1

class main(){
    herr_t status = FAIL;
    int comp_level[5] = {0,0,0,0,0};
    int comp_code;
    int i, j;
    hid_t swfid = FAIL;
    hid_t SWid = FAIL;
    hsize_t chunk_dims[2];

    /* Open the file, "Swath.he5", using the H5F_ACC_RDWR access code */
    /* --------------------------------------------------------------- */
    swfid = HE5_SWopen("Swath.he5", H5F_ACC_RDWR);
    if (swfid != FAIL)
    {
        SWid = HE5_SWattach(swfid, "Swath1");
        if (SWid != FAIL)
        {
            /* We define seven fields. The first three, "Time", "Longitude"
             * and "Latitude" are geolocation fields and thus we use the
             * geolocation dimensions "GeoTrack" and "GeoXtrack" in the
             * field definitions.
             * The next four fields are data fields. Note that either
             * geolocation or data dimensions can be used.
             */
            status = HE5_SWdefinefield(SWid, "Time", "GeoTrack", NULL,
                                        H5T_NATIVE_DOUBLE, 0);
        }
    }
}
printf("Status returned by HE5_SWdefgeofield(..."Time",...) : %d\n",status);

status = HE5_SWdefgeofield(SWid, "Longitude", "GeoTrack,GeoXtrack", NULL, H5T_NATIVE_FLOAT, 0);
printf("Status returned by HE5_SWdefgeofield(..."Longitude",...) : %d\n",status);

status = HE5_SWdefgeofield(SWid, "Latitude", "GeoTrack,GeoXtrack", NULL, H5T_NATIVE_FLOAT, 0);
printf("Status returned by HE5_SWdefgeofield(..."Latitude",...) : %d\n",status);

status = HE5_SWdefdatafield(SWid, "Density", "GeoTrack", NULL, H5T_NATIVE_FLOAT, 0);
printf("Status returned by HE5_SWdefdatafield(..."Density",...) : %d\n",status);

status = HE5_SWdefdatafield(SWid, "Temperature", "GeoTrack", NULL, H5T_NATIVE_FLOAT, 0);
printf("Status returned by HE5_SWdefdatafield(..."Temperature",...) : %d\n",status);

status = HE5_SWdefdatafield(SWid, "Pressure", "Res2tr,Res2xtr", NULL, H5T_NATIVE_DOUBLE, 0);
printf("Status returned by HE5_SWdefdatafield(..."Pressure",...) : %d\n",status);

status = HE5_SWdefdatafield(SWid, "Spectra", "Bands,Res2tr,Res2xtr", NULL, H5T_NATIVE_DOUBLE, 0);
printf("Status returned by HE5_SWdefdatafield(..."Spectra",...) : %d\n",status);

/* Define Profile field */
/* ------------------- */
status = HE5_PRdefine(SWid, "Profile-2000", "ProfDim", NULL, H5T_NATIVE_UINT);
printf("Status returned by HE5_PRdefine(..."Profile-2000",...) : %d\n",status);

/* Define Appendable Field */
/* ------------------------ */

/* First, define chunking */
/* (the appendable dataset must be chunked) */
/* ------------------------------------------- */
chunk_dims[0] = 20;
status = HE5_SWdefchunk(SWid, RANK, chunk_dims);
printf("Status returned by HE5_SWdefchunk() : %d\n",status);

/* Second, define compression scheme */
/* ---------------------------------- */

/* set the value of compression code: */
/* HDPE_COMP_NONE 0 */
/* HDPE_COMP_RLE 1 */
/* HDPE_COMP_NBIT 2 */
/* HDPE_COMP_SKPHUFF 3 */
/ * HDFE_COMP_DEFLATE               4 */
comp_code = 4;

/* Set compression level: value 0,1,2,3,4,5,6,7,8, or 9 */
/* ------------------------------------------------------ */
comp_level[0] = 6;

status = HE5_SWdefcomp(SWid, comp_code, comp_level);
printf("\tStatus returned by HE5_SWdefcomp() :\n%d\n", status);

status = HE5_SWdefdatafield(SWid, "Count", "GeoTrack", "Unlim",
H5T_NATIVE_INT, 0);
printf("Status returned by HE5_SWdefdatafield(...\"Count\",...) :\n%d\n", status);
}
}

status = HE5_SWdetach(SWid);
status = HE5_SWclose(swfid);

return 0;
}

Example 3

/* In this example we (1) open the "Swath.he5" file, (2) attach to the
 * "Swath1" swath, and (3) write data to the "Longitude", "Latitude", and
 * "Spectra" fields. Also, set up the global, group, and local attributes */
/* he5_sw_writedata */

#include <HE5_HdfEosDef.h>

main()
{
    herr_t          status = FAIL;
    int             i, j, k;
    int             track, xtrack;
    int             attr1[4] = {1, 2, 3, 4};         /* global attribute */
    int             attr2[4] = {10, 20, 30, 40};     /* group attribute */
    int             attr3[4] = {100, 200, 300, 400}; /* local attribute */
    hid_t           swfid = FAIL;
    hid_t           SWid  = FAIL;
    char            attr4[7]; /* Global 'char' attribute */
 'long' attribute */
    double          attr6[4] = {1.111111,2.222222,3.333333,4.444444};/* Global
 'double' attribute */
float attr7[4] = {1.111111, 2.222222, 3.333333, 4.444444}; /* Local 'float' attribute */

hssize_t start[3];
hsize_t count[3];
size_t datasize;
float lng[20][10], latcnt;
float lat[20][10], loncnt;
double plane[15][40][20], tme[20];
hvl_t buffer[4]; /* Data buffer for the profile */

/* Populate lon/lat data arrays */
/* -------------------------------- */
latcnt = 1.;
loncnt = 1.;
track = 0;
xtrack = 0;
while(track < 20) {
    while(xtrack < 10) {
        lat[track][xtrack] = latcnt;
        lng[track][xtrack] = loncnt;
        loncnt = loncnt + 1.;
        xtrack++;
    }
    latcnt = latcnt + 1.;
    loncnt = 1.;
    track++;
xtrack = 0;
}

/* Popolate spectra data array. Value = 100*(track index)+(band index) */
/* --------------------------------------------------------------- */
for (i = 0; i < 15; i++) {
    for (j = 0; j < 40; j++)
        for (k = 0; k < 20; k++)
            plane[i][j][k] = (double)(j*100 + i);
}

/* Allocate memory for and populate data buffer */
/* ----------------------------------------------- */
datasize = 0;
for (i = 0; i < 4; i++) {
    buffer[i].p = malloc(25 *(i+1)* sizeof(unsigned int));
    buffer[i].len = 25 * (i+1);
    /* calculate the data buffer size (bytes) */
    datasize += buffer[i].len * sizeof(unsigned int);
    for (j = 0; j < 25 * (i+1); j++)
        ((unsigned int *)buffer[i].p)[j] = (i+1)*1000 + j;
}
/* Open the HDF swath file, "Swath.he5" */
/* ------------------------------------ */
swfid = HE5_SWopen("Swath.he5", H5F_ACC_RDWR);
if (swfid != FAIL)
{
    /* Attach the "Swath1" swath */
    /* ------------------------ */
    SWid = HE5_SWattach(swfid, "Swath1");
    if (SWid != FAIL)
    {
        start[0] = 0;
        start[1] = 0;
        count[0] = 20;
        count[1] = 10;

        /* Write longitude field */
        /* --------------------- */
        status = HE5_SWwritefield(SWid, "Longitude", start, NULL, count, lng);
        printf("status returned by HE5_SWwritefield("Longitude"):
%dn", status);

        /* Write latitude field */
        /* --------------------- */
        status = HE5_SWwritefield(SWid, "Latitude", start, NULL, count, lat);
        printf("status returned by HE5_SWwritefield("Latitude"):
%dn", status);

        /* Write Time Field */
        /* ---------------- */
        for (i = 0; i < 20; i++)
        {
tme[i] = 34574087.3 + 84893.2*i;
        start[0] = 0;
        count[0] = 20;
        status = HE5_SWwritefield(SWid, "Time", start, NULL, count, tme);
        printf("status returned by HE5_SWwritefield("Time"):
%dn", status);

        /* Write Spectra Field */
        /* -------------------- */
        start[0] = 0; count[0] = 15;
        start[1] = 0; count[1] = 40;

        status = HE5_SWwritefield(SWid, "Spectra", start, NULL, count, plane);
        printf("status returned by HE5_SWwritefield("Spectra"):
%dn", status);

        /* Write data to the profile */
        /* -------------------------- */
        start[0] = 0; count[0] = 4;
        status = HE5_PRwrite(SWid, "Profile-2000", start, NULL, count, datasize, buffer);
    }
}
printf("Status returned by HE5_PRwrite("Profile-2000"):\n\%d \n", status);

    /* Write Global 'int' Attribute */
    /* -------------------------------- */
    count[0] = 4;
    status = HE5_SWwriteattr(SWid, "GlobalAttribute",
    H5T_NATIVE_INT, count, attr1);
    printf("status returned by HE5_SWwriteattr("GlobalAttribute"):\n\%d\n", status);

    /* Write Global 'char' Attribute */
    /* -------------------------------- */
    strcpy(attr4,"ABCDEFG");
    count[0] = 6;
    status = HE5_SWwriteattr(SWid, "GLOBAL_CHAR_ATTR",
    H5T_NATIVE_CHAR, count, attr4);
    printf("status returned by HE5_SWwriteattr("GLOBAL_CHAR_ATTR"):\n\%d\n", status);

    /* Write Global 'long' Attribute */
    /* -------------------------------- */
    count[0] = 4;
    status = HE5_SWwriteattr(SWid, "GLOBAL_LONG_ATTR",
    H5T_NATIVE_LONG, count, attr5);
    printf("status returned by HE5_SWwriteattr("GLOBAL_LONG_ATTR"):\n\%d\n", status);

    /* Write Global 'double' Attribute */
    /* ----------------------------------- */
    count[0] = 4;
    status = HE5_SWwriteattr(SWid, "GLOBAL_DOUBLE_ATTR",
    H5T_NATIVE_DOUBLE, count, attr6);
    printf("status returned by HE5_SWwriteattr("GLOBAL_DOUBLE_ATTR"):\n\%d\n", status);

    /* Write Group Attribute */
    /* -------------------- */
    status = HE5_SWwritegrpattr(SWid, "GroupAttribute",
    H5T_NATIVE_INT, count, attr2);
    printf("status returned by HE5_SWwritegrpattr("GroupAttribute"):\n\%d\n", status);

    /* Write Local Attribute */
    /* ---------------------- */
    status = HE5_SWwritelocattr(SWid, "Density", "LocalAttribute_1",
    H5T_NATIVE_INT, count, attr3);
    printf("status returned by HE5_SWwritelocattr("LocalAttribute_1"):\n\%d\n", status);

    /* Write Local Attribute */
    /* ---------------------- */
    status = HE5_SWwritelocattr(SWid, "Longitude", "LocalAttribute_2",
    H5T_NATIVE_FLOAT, count, attr7);
    printf("status returned by HE5_SWwritelocattr("LocalAttribute_2"):\n\%d\n", status);
}
\begin{verbatim}
status = HE5_SWdetach(SWid);
status = HE5_SWCLOSE(swfid);

return 0;
}

Example 4

\/*
* In this example we (1) open the "Swath.he5" HDF-EOS file, (2) attach to
* the "Swath1" swath, and (3) read data from the "Longitude" field. Also,
* we read the global/group/local attributes
*/

\/* he5_sw_readdata */

#include <HE5_HdfEosDef.h>

main()
{
  herr_t status = FAIL;
  int i, j, k;
  int attr1[4];    /* data buffer for global attribute */
  int attr2[4];    /* ... for group attribute */
  int attr3[4];    /* ... for local attribute */
  hid_t swfid = FAIL;
  hid_t SWid = FAIL;
  char attr4[10];  /* ... for global 'char' attribute */
  long attr5[4];  /* ... for global 'long' attribute */
  double attr6[4]; /* ... for global 'double' attribute */
  float attr7[4]; /* ... for local 'float' attribute */
  hssize_t start[2];
  hsize_t stride[2], count[2];
  float lng[20][10];
  hvl_t buffer_out[4]; /* Buffer to read out data from profile */

  /* Open the HDF-EOS swath file, "Swath.he5" */
  \/* ---------------------------------------- */
  swfid = HE5_SWopen("Swath.he5", H5F_ACC_RDONLY);
  if (swfid != FAIL)
  {
    /* Attach the "Swath1" swath */
    \/* -------------------------- */
    SWid = HE5_SWattach(swfid, "Swath1");
    if (SWid != FAIL)
    {
      /* Read data from the "Longitude" field */
      \/* ------------------------ */
      \/* Read global attributes */
      buffer_out[0] = attr4[0];
      buffer_out[1] = attr4[1];
      buffer_out[2] = attr4[2];
      buffer_out[3] = attr4[3];
      \/* Read group attributes */
      buffer_out[4] = attr5[0];
      buffer_out[5] = attr5[1];
      buffer_out[6] = attr5[2];
      buffer_out[7] = attr5[3];
      \/* Read local attributes */
      buffer_out[8] = attr6[0];
      buffer_out[9] = attr6[1];
      buffer_out[10] = attr6[2];
      buffer_out[11] = attr6[3];
    }
  }
}
\end{verbatim}
{ /* Read the entire longitude field */
  /* -------------------------------- */
  start[0] = 0;  start[1] = 0;
  count[0] = 20;  count[1] = 10;
  status = HE5_SWreadfield(SWid, "Longitude", start, NULL, count,
 lng);
  printf("Status returned by HE5_SWreadfield() : %d \n", status);

  /* Display longitude data */
  /* ----------------------- */
  for (i = 0; i < 20; i++)
    for (j = 0; j < 10; j++)
      printf("i j Longitude: %d %d %f \n", i, j, lng[i][j]);

  /* Read data from the Profile */
  /* -------------------------------- */
  start[0] = 0;  count[0] = 4;
  status = HE5_PRread(SWid, "Profile-2000", start, NULL, count,
 buffer_out);
  printf("Status returned by HE5_PRread() : %d \n", status);

  /* Display the profile data */
  /* ------------------------ */
  for (i = 0; i < 4; i++)
    { printf("The %d-th element length is %d \n", i,
              (unsigned)buffer_out[i].len);
      for (j = 0; j < buffer_out[i].len; j++)
        printf(" \t \t %d \n", ((unsigned int *)buffer_out[i].p)[j]);
    }

  /* Release IDs and memory */
  /* ------------------------ */
  status = HE5_PRreclaimspace(SWid, "Profile-2000", buffer_out);
  printf("Status returned by HE5_PRreclaimspace() : %d \n", status);

  /* Read Global 'int' Attribute */
  /* -------------------------------- */
  status = HE5_SWreadattr(SWid, "GlobalAttribute", attr1);
  printf("Status returned by HE5_SWreadattr() : %d \n", status);
  printf("Global attribute values:\n");
  for (i = 0; i < 4; i++)
    printf(" \t \t %d \n", attr1[i]);

  /* Read Group Attribute */
  /* ---------------------- */
  status = HE5_SWreadgrpattr(SWid, "GroupAttribute", attr2);
  printf("Status returned by HE5_SWreadgrpattr() : %d \n", status);
  printf("Group attribute values:\n");
  for (i = 0; i < 4; i++)
    printf(" \t \t %d \n", attr2[i]);
/* Read Local Attribute */
/* ------------------- */
status = HE5_SWreadlocattr(SWid, "Density", "LocalAttribute_1", attr3);
printf("Status returned by HE5_SWreadlocattr() : %d \n", status);
printf("Local attribute values: \n");
for (i = 0; i < 4; i++)
    printf("%d \n", attr3[i]);

/* Read Local Attribute */
/* ------------------- */
status = HE5_SWreadlocattr(SWid, "Longitude", "LocalAttribute_2", attr7);
printf("Status returned by HE5_SWreadlocattr() : %d \n", status);
printf("Local attribute values: \n");
for (i = 0; i < 4; i++)
    printf("%f \n", attr7[i]);

/* Read Global 'char' Attribute */
/* ----------------------------- */
status = HE5_SWreadattr(SWid, "GLOBAL_CHAR_ATTR", attr4);
printf("Status returned by HE5_SWreadattr() : %d \n", status);
printf("Global attribute values: \n");
printf("%s \n", attr4);

/* Read Global 'long' Attribute */
/* ----------------------------- */
status = HE5_SWreadattr(SWid, "GLOBAL_LONG_ATTR", attr5);
printf("Status returned by HE5_SWreadattr() : %d \n", status);
printf("Global attribute values: \n");
for (i = 0; i < 4; i++)
    printf("%li \n", attr5[i]);

/* Read Global 'double' Attribute */
/* ----------------------------- */
status = HE5_SWreadattr(SWid, "GLOBAL_DOUBLE_ATTR", attr6);
printf("Status returned by HE5_SWreadattr() : %d \n", status);
printf("Global attribute values: \n");
for (i = 0; i < 4; i++)
    printf("%f \n", attr6[i]);

status = HE5_SWdetach(SWid);
status = HE5_SWclose(swfid);
return 0;
}

Example 5
/*
* In this example we (1) open the "Swath.he5" HDF-EOS file, (2) attach to
* the "Swath1", and (3) read data from the "Spectra" and "Time" fields
*/

/* he5_sw_subset  */

#include <HE5_HdfEosDef.h>

main()
{
    herr_t          status = FAIL;
    int             i, j,  rank = 0;
    hid_t           swfid = FAIL, SWid = FAIL;
    hid_t           regionID = FAIL, periodID = FAIL;
    hid_t           *ntype;
    size_t          size = 0;
    hsize_t         dims[8];
    double          cornerlon[2], cornerlat[2];
    double          *datbuf, start_time, stop_time;

    /* Open the HDF-EOS swath file, "Swath.he5" */
    /* ------------------------------- */
    swfid = HE5_SWopen("Swath.he5", H5F_ACC_RDWR);
    if (swfid != FAIL)
    {
        /* Attach to the "Swath1" swath */
        /* ------------------- */
        SWid = HE5_SWattach(swfid, "Swath1");
        if (SWid != FAIL)
        {
            cornerlon[0] = 3. ;
            cornerlat[0] = 5. ;
            cornerlon[1] = 7. ;
            cornerlat[1] = 12.;

            regionID = HE5_SWdefboxregion(SWid, cornerlon, cornerlat,
HE5_HDFE_MIDPOINT);
            printf("\n");
            printf("Region ID returned by HE5_SWdefboxregion() : \
%d \n", regionID);
            ntype = (hid_t *)calloc(1, sizeof(hid_t ));
            status = HE5_SWregioninfo(SWid, regionID, "Longitude", ntype,
&rank, dims, &size);
            printf("Status returned by HE5_SWregioninfo("Longitude") : \
%d \n", status);
        }
    }
}
status = HE5_SWregioninfo(SWid, regionID, "Spectra", ntype, &rank, dims, &size);
    printf("Status returned by HE5_SWregioninfo("Spectra") : %d \n", status);
    datbuf = (double *)calloc(size, sizeof(double));
    status = HE5_SWextractregion(SWid, regionID, "Spectra", HE5_HDFE_INTERNAL, datbuf);
    printf("Status returned by HE5_SWextractregion() : %d \n", status);
    printf("\n");
    printf("=*=*=*=*=*=*=*  DATA  =*=*=*=*=*=*=* \n");
    printf("\n");
    for (i = 0; i < size / sizeof(double); i++)
        printf("%lf \n", datbuf[i]);
    free(datbuf);
    /* Time Subsetting */
    /* ----------------*/
    start_time = 35232487.2;
    stop_time = 36609898.1;
    periodID = HE5_SWdeftimeperiod(SWid, start_time, stop_time, HE5_HDFE_MIDPOINT);
    printf("\n");
    printf("Period ID returned by HE5_SWdeftimeperiod() : %d \n", periodID);
    status = HE5_SWperiodinfo(SWid, periodID, "Time", ntype, &rank, dims, &size);
    printf("Status returned by HE5_SWperiodinfo() : %d \n", status);
    datbuf = (double *)calloc(size, sizeof(double));
    status = HE5_SWextractperiod(SWid, periodID, "Time", HE5_HDFE_INTERNAL, datbuf);
    printf("Status returned by HE5_SWextractperiod() : %d \n", status);
    printf("\n");
    printf("=*=*=*=*=*=*=*  DATA  =*=*=*=*=*=*=* \n");
    printf("\n");
    for (i = 0; i < size / sizeof(double); i++)
        printf("%lf \n", datbuf[i]);
    free(datbuf);
    free(ntype);
}
Example 6

/*
 * In this example we retrieve information about (1) dimensions, (2) dimension mappings (geolocation relations), (3) swath fields, and (4) the global/group/local attributes
*/

/* he5_sw_info */
#include <HE5_HdfEosDef.h>

main()
{
    herr_t status = FAIL;
    int i, rk, *rank;
    hid_t swfid = FAIL, SWid = FAIL;
    hid_t ntype[10];
    hid_t dtype = FAIL;
    long ndims, strbufsize, nmaps, nfields, nattr;
    long *off, *inc, *indx, offset, incr;
    hsize_t *sizes, dimsize;
    hsize_t dim[8], *dims;
    hsize_t n, nelem = 0;
    char version[80];
    char *dimname, *dimmap, *fieldlist;
    char dimlist[80], attrlist[80];

    /* Open the Swath HDF-EOS File "Swath.he5" for reading only */
    /* ----------------------------------------------- */
    swfid = HE5_SWopen("Swath.he5", H5F_ACC_RDONLY);
    if (swfid != FAIL)
    {
        HE5_EHgetversion(swfid, version);
        printf("HDF-EOS library version: \"%s\" \n", version);

        /* Attach the swath "Swath1" */
        /* -------------------------------- */
        SWid = HE5_SWattach(swfid, "Swath1");
        if (SWid != FAIL)
        {
            /* Inquire Dimensions */
            /* ----------------------- */
            ndims = HE5_SWnentries(SWid, HE5_HDFE_NENTDIM, &strbufsize);
            dims = (hsize_t *) calloc(ndims, sizeof(hsize_t));
            dimname = (char *) calloc(strbufsize + 1, 1);

            ndims = HE5_SWndims(SWid, dimname, dims);
            printf("Dimension list: %s\n", dimname);
            for (i = 0; i < ndims; i++)
printf("dim size: %lu\n", (unsigned long)dims[i]);
free(dims);
free(dimname);

/* Inquire Dimension Mappings */
/* -------------------------- */
nmaps = HE5_SECnentries(SWid, HE5_HDFE_NENTMAP, &strbufsize);
off = (long *)calloc(nmaps, sizeof(long));
icn = (long *)calloc(nmaps, sizeof(long));
dimmap = (char *)calloc(strbufsize + 1, 1);

nmaps = HE5_SECinqmaps(SWid, dimmap, off, inc);
printf("Dimension map: %s\n", dimmap);
for (i = 0; i < nmaps; i++)
    printf("offset increment: %li %li\n", off[i], inc[i]);
free(off);
free(inc);
free(dimmap);

/* Inquire Indexed Dimension Mappings */
/* ------------------------------- */
nmaps = HE5_SECnentries(SWid, HE5_HDFE_NENTIMAP, &strbufsize);
sizes = (hsize_t *) calloc(nmaps, sizeof(hsize_t));
dimmap = (char *) calloc(strbufsize + 1, 1);
nmaps = HE5_SECinqidxmaps(SWid, dimmap, sizes);
printf("Index Dimension map: %s\n", dimmap);
for (i = 0; i < nmaps; i++)
    printf("sizes: %lu\n", (unsigned long)sizes[i]);
free(sizes);
free(dimmap);

/* Inquire Geolocation Fields */
/* -------------------------- */
nflds = HE5_SECnentries(SWid, HE5_HDFE_NENTGFLD, &strbufsize);
rank = (int *)calloc(nflds, sizeof(int));
fieldlist = (char *) calloc(strbufsize + 1, 1);
nflds = HE5_SECinqgeofields(SWid, fieldlist, rank, ntype);
printf("geo fields: %s\n", fieldlist);
for (i = 0; i < nflds; i++)
    printf("Rank: %d Data type: %d\n", rank[i], ntype[i]);
free(rank);
free(fieldlist);

/* Inquire Data Fields */
/* ------------------- */
nflds = HE5_SECnentries(SWid, HE5_HDFE_NENTDFLD, &strbufsize);
rank = (int *)calloc(nflds, sizeof(int));
fieldlist = (char *) calloc(strbufsize + 1, 1);
nflds = HE5_SECinqdatafields(SWid, fieldlist, rank, ntype);
printf("data fields: %s\n", fieldlist);
for (i = 0; i < nflds; i++)
    printf("Rank: %d Data type: %d\n", rank[i], ntype[i]);
free(rank);
free(fieldlist);

/* Get info on "GeoTrack" dim */
*-------------------------------*/
dimsize = HE5_SWdiminfo(SWid, "GeoTrack");
printf("Size of GeoTrack: %lu\n", (unsigned long)dimsize);

/* Get info on "GeoTrack/Res2tr" mapping */
*----------------------------------------*/
status = HE5_SWmapinfo(SWid, "GeoTrack", "Res2tr", &offset, &incr);
printf("Mapping Offset: %li\n", offset);
printf("Mapping Increment: %li\n", incr);

/* Get info on "IndxTrack/Res2tr" indexed mapping */
*-----------------------------------------------*/
dimsiz = HE5_SWdiminfo(SWid, "IndxTrack");
indx = (long *) calloc(dimsiz, sizeof(long));
n = HE5_SWidxmapinfo(SWid, "IndxTrack", "Res2tr", indx);
for (i = 0; i < n; i++)
    printf("Index Mapping Entry %d: %li\n", i+1, indx[i]);
free(indx);

/* Get info on "Longitude" Field */
*------------------------------*/
status = HE5_SWfieldinfo(SWid, "Longitude", &rk, dim, &dtype, dimlist, NULL);
printf("Longitude Rank: %d\n", rk);
printf("Longitude NumberType: %d\n", dtype);
printf("Longitude Dimension List: %s\n", dimlist);
for (i = 0; i < rk; i++)
    printf("Dimension %d: %lu\n", i+1,(unsigned long)dim[i]);
dtype = FAIL;

/* Get info about Global Attributes */
*--------------------------------*/
printf("Global Attribute:\n");
status = HE5_SWattrinfo(SWid, "GlobalAttribute", &dtype, &nelem);
printf("\"\"\"Data type: %d\n", dtype);
printf("\"\"\"Number of elements: %lu \n", (unsigned long)nelem);
nelem = 0;
dtype = FAIL;
/* Get info about Group Attributes */
*--------------------------------*/
printf("Group Attribute:\n");
status = HE5_SWgrpattrinfo(SWid, "GroupAttribute", &dtype, &nelem);
printf("\"\"\"Data type: %d\n", dtype);
printf("\"\"\"Number of elements: %lu \n", (unsigned long)nelem);
nelem = 777;
dtype = FAIL;
/* Get info about Local Attributes */
/* ----------------------------------- */
printf("Local Attribute:\n");
status = HE5_SWlocattrinfo(SWid,"Density",
"LocalAttribute_1",&dtype,&nelem);
printf("\t\t Data type:          %d\n", dtype);
printf("\t\t Number of elements: %lu \n", (unsigned long)nelem);
printf("\n");
status = HE5_SWlocattrinfo(SWid,"Longitude",
"LocalAttribute_2",&dtype,&nelem);
printf("\t\t Data type:          %d\n", dtype);
printf("\t\t Number of elements: %lu \n", (unsigned long)nelem);

/* Inquire Global Attributes */
/* -------------------------- */
printf("Global Attributes:\n");
nattr = HE5_SWinqattrs(SWid, NULL, &strbufsize);
printf("\t\t Number of attributes:             %li \n", nattr);
printf("\t\t String length of attribute list:  %li \n", strbufsize);

n = HE5_SWinqattrs(SWid, attrlist, &strbufsize);
printf("\t\t Attribute list:                   %s \n", attrlist);

/* Inquire Group Attributes */
/* ------------------------ */
strbufsize = 0;
printf("\n");
printf("Group Attributes:\n");
nattr = HE5_SWinqgrattrs(SWid, NULL, &strbufsize);
printf("\t\t Number of attributes:             %li \n", nattr);
printf("\t\t String length of attribute list:  %li \n", strbufsize);
strcpy(attrlist,"");
nattr = HE5_SWinqgrattrs(SWid, attrlist, &strbufsize);
printf("\t\t Attribute list:                   %s \n", attrlist);

/* Inquire Local Attributes */
/* ------------------------ */
strbufsize = 0;
printf("\n");
printf("Local Attributes:\n");
nattr = HE5_SWinqlocattrs(SWid, "Density", NULL, &strbufsize);
printf("\t\t Number of attributes:             %li \n", nattr);
printf("\t\t String length of attribute list:  %li \n", strbufsize);
strcpy(attrlist,"");
nattr = HE5_SWinqlocattrs(SWid, "Density", attrlist, &strbufsize);
printf("\t\t Attribute list:                   %s \n", attrlist);

nattr = HE5_SWinqlocattrs(SWid, "Longitude", NULL, &strbufsize);
printf("\t\t Number of attributes:             %li \n", nattr);
printf("\t\t String length of attribute list: %li \n", strbufsize);
strcpy(attrlist,"";
nattr = HE5_SWinqlocattrs(SWid, "Longitude", attrlist, &strbufsize);
printf("\t\t Attribute list:                   %s \n", attrlist);
}
}
status = HE5_SWdetach(SWid);
status = HE5_SWclose(swfid);
return 0;
}

7.2.1.2 A FORTRAN Example of a Simple Swath Creation

Example 1

In this example we (1) open an HDF-EOS file, (2) create the swath interface, and (3) define the swath field dimensions

```
program he5_sw_setupF_32
integer          status
integer          he5_swopen
integer          he5_swcreate
integer          he5_swdefdim
integer          he5_swdefmap
integer          he5_swdefimap
integer          he5_swdetach
integer          he5_swclose
integer*4        swfid, swid
integer*4        datatrack
integer*4        offset, incr
integer*4        indx(12)
integer          HE5F_ACC_TRUNC
parameter (HE5F_ACC_TRUNC=102)
integer*4        HE5T_UNLIMITED_F
parameter (HE5T_UNLIMITED_F=-1)
data indx   /0,1,3,6,7,8,11,12,14,24,32,39/
```

Open the HDF-EOS file, "swath.he5" using HE5F_ACC_TRUNC access code
```
swfid = he5_swopen("swath.he5",HE5F_ACC_TRUNC)
```
Create the swath, "Swath1", within the file
```
swid = he5_swcreate(swfid, "Swath1")
```
Define Geolocation and Data dimensions
----------------------------------------

Typically, many fields within a swath share the same dimension. The swath interface therefore provides a way of defining dimensions that will then be used to define swath fields. A dimension is defined with a name and a size and is connected to the particular swath through the swath id. In this example, we define the geolocation track and cross track dimensions with size 20 and 10 respectively and two dimensions corresponding to these but with twice the resolution. Also, we define "Bands" and "unlimited" dimensions.

```
datatrack = 20
status = he5_swdefdim(swid, "GeoTrack", datatrack)

datatrack = 10
status = he5_swdefdim(swid, "GeoXtrack", datatrack)

datatrack = 40
status = he5_swdefdim(swid, "Res2tr", datatrack)

datatrack = 20
status = he5_swdefdim(swid, "Res2xtr", datatrack)

datatrack = 15
status = he5_swdefdim(swid, "Bands", datatrack)

datatrack = 12
status = he5_swdefdim(swid, "IndxTrack", datatrack)

datatrack = 4
status = he5_swdefdim(swid, "ProfDim", datatrack)
```

Define Unlimited (appendable) dimension
---------------------------------------
```
status = he5_swdefdim(swid, "Unlim", HE5T_UNLIMITED_F)
```

Once the dimensions are defined, the relationship (mapping) between the geolocation dimensions, such as track and cross track, and the data dimensions, must be established. This is done through the SWdefdimmap routine. It takes as input the swath id, the names of the dimensions designating the geolocation and data dimensions, respectively, and the offset and increment defining the relation.

In the first example we relate the "GeoTrack" and "Res2tr" dimensions with an offset of 0 and an increment of 2. Thus the ith element of "Geotrack" corresponds to the 2 * ith element of "Res2tr".

In the second example, the ith element of "GeoXtrack" corresponds to the 2 * ith + 1 element of "Res2xtr".

```
Define dimension mappings
-------------------------
offset = 0
incr   = 2
status = he5_swdefmap(swid, "GeoTrack", "Res2tr", offset, incr)
```
offset = 1
status = he5_swdefmap(swid, "GeoXtrack", "Res2xtr", offset, incr)
c     Define indexed dimension mapping
--------------------------
status = he5_swdefimap(swid, "IndxTrack", "Res2tr", indx)
c     Detach from the swath
---------------------
status = he5_swdetach(swid)
c     Close the swath file
-------------------
status = he5_swclose(swfid)
stop
end

Example 2
In this example we (1) open the "swath.he5" HDF-EOS file, (2) attach to the "Swath1" swath, and (3) define the swath fields.
program     he5_sw_definefieldsF_32
integer     status
integer     he5_swopen
integer     he5_swattach
integer     he5_swdefgfld
integer     he5_swdefdfld
integer     he5_prdefine
integer     he5_swdetach
integer     he5_swclose
integer     swfid, swid

integer     HE5F_ACC_RDWR
parameter   (HE5F_ACC_RDWR=100)
integer     HE5T_NATIVE_FLOAT
parameter   (HE5T_NATIVE_FLOAT=10)
integer     HE5T_NATIVE_DOUBLE
parameter   (HE5T_NATIVE_DOUBLE=11)
integer     HE5T_NATIVE_INT
parameter   (HE5T_NATIVE_INT=0)

c     Open the HDF-EOS file, "swath.he5" using HE5F_ACC_RDWR access code
----------------------------------------------------------------------------------
swfid = he5_swopen("swath.he5", HE5F_ACC_RDWR)
if (swfid .NE. FAIL) then
swid = he5_swattach(swfid, "Swath1")
if (swid .NE. FAIL) then

c     Define Geolocation and Data fields
We define six fields. The first three, "Time", "Longitude" and "Latitude" are geolocation fields and thus we use the geolocation dimensions "GeoTrack" and "GeoXtrack" in the field definitions. We also must specify the data type using the standard HDF data type codes. In this example the geolocation are 4-byte (32 bit) floating point numbers.

The next three fields are data fields. Note that either geolocation or data dimensions can be used.

```c
status = he5_swdefgfld(swid, "Time", "GeoTrack", "", 1HE5T_NATIVE_DOUBLE, 0)
status = he5_swdefgfld(swid, "Longitude", "GeoXtrack,GeoTrack", 1"", HE5T_NATIVE_FLOAT, 0)
status = he5_swdefgfld(swid, "Latitude", "GeoXtrack,GeoTrack", 1"", HE5T_NATIVE_FLOAT, 0)
status = he5_swdefdfld(swid, "Density", "GeoTrack", 1"", HE5T_NATIVE_FLOAT, 0)
status = he5_swdefdfld(swid, "Temperature", "GeoXtrack,GeoTrack", 1"", HE5T_NATIVE_FLOAT, 0)
status = he5_swdefdfld(swid, "Pressure", "Res2xtr,Res2tr", 1"", HE5T_NATIVE_FLOAT, 0)
status = he5_swdefdfld(swid, "Spectra", "Res2xtr,Res2tr,Bands", 1"", HE5T_NATIVE_DOUBLE, 0)
```

Define Profile Field

```c
status = he5_prdefine(swid, "Profile-2000", "ProfDim", 1"", HE5T_NATIVE_INT, 0)
```

Example 3

In this program we (1) open the "swath.h5" file, (2) attach to the "Swath1" swath, and (3) write data to the "Longitude", "Longitude" and "Spectra" fields.
program he5_sw_writedataF_32
implicit none

integer status
integer he5_swopen
integer he5_swattach
integer he5_swwrfld
integer he5_swwrattr
integer he5_swdetach
integer he5_swclose
integer he5_prwrite
integer swfid, SWid
integer buffer(250)
integer counter
integer i, j, k
integer itrack

integer*4 attr(4)
integer*4 track
integer*4 start(3)
integer*4 stride(3)
integer*4 count(3)
integer*4 len(4)
integer*4 datasize

real lng(10)
real lat(10)

real*8 plane(800)
real*8 tme(20)

integer HE5F_ACC_RDWR
parameter (HE5F_ACC_RDWR=100)

integer HE5T_NATIVE_INT
parameter (HE5T_NATIVE_INT=0)

integer FAIL
parameter (FAIL=-1)

! Set longitude values along the cross track -----------------------------------------
do i=1,10
   lng(i) = i-1.0
endo

! Open HDF-EOS file, "swath.h5" ----------------------------------------------------
swfid = he5_swopen("swath.h5", HE5F_ACC_RDWR)
if (swfid .NE. FAIL) then
   SWid = he5_swattach(swfid, "Swath1")
   if (swid .NE. FAIL) then
      ! Write data starting at the beginning of each cross track ---------------------
      start(1) = 0
   endif
endif
stride(1) = 1
stride(2) = 1
count(1)  = 10
count(2)  = 1

c Loop through all the tracks, incrementing the track starting
position by one each time
------------------------------------------------------------
do track = 1,20
  start(2) = track - 1
  status = he5_swwrfld(swid,"Longitude",start,
  stride,count,lng)
  do itrack = 1,10
    lat(itrack) = track
  enddo
  status = he5_swwrfld(swid,"Latitude",start,
  stride,count,lat)
  enddo

do i = 1,20
tme(i) = 34574087.3 + 84893.2*(i-1)
enddo

start(1)  = 0
stride(1) = 1
count(1)  = 20

status = he5_swwrfld(swid,"Time",start,stride,
count,tme)

  Write Spectra one plane at a time
  Value is 100 * track index + band index (0-based)
  --------------------------------------------------
  start(1)  = 0
  start(2)  = 0
  count(1)  = 20
  count(2)  = 40
  count(3)  = 1
  stride(3) = 1

  do i=1,15
    start(3) = i - 1
    do j=1,40
      do k=1,20
        plane((j-1)*20+k) = (j-1)*100 + i-1
      enddo
    enddo
    status = he5_swwrfld(swid,"Spectra",start,
    stride,count,plane)
  enddo

  Populate data buffer and write data to the Profile Field
  --------------------------------------------------------
datasize = 0
counter  = 0

do i=1,4
  len(i) = i*25
  datasize = datasize + len(i)
do j=1,(25*i)
    counter = counter + 1
    buffer(counter) = (i)*1000 + j - 1
endo
enddo

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

status = he5_prwrite(swid,"Profile-2000",start,stride,count,
ldatasize,len,buffer)
write(*,*) 'Status returned by he5_prwrite(): ',status

attr(1) = 3
attr(2) = 5
attr(3) = 7
attr(4) = 11
count(1) = 4
status = he5_swwrattr(swid,"TestAttr",HE5T_NATIVE_INT,
c
lcount,attr)

endif
endif

status = he5_swdetach(swid)
write(*,*) 'Status returned by he5_swdetach(): ',status

status = he5_swclose(sfid)
write(*,*) 'Status returned by he5_swclose(): ',status
stop
end

Example 4

c In this program we (1) open the "swath.h5" file, (2) attach to
c the "Swath1" swath, and (3) read data from the "Longitude" field.

program he5_sw_readdataF_32
implicit none
integer status
integer he5_swopen
integer he5_swattach
integer he5_swrdfld
integer he5_swwrattr
t...
integer  he5_swclose
integer  swfid, swid
integer  buffer_out(250)
integer  i,j,jl
integer  element1(25)
integer  element2(50)
integer  element3(75)
integer  element4(100)

real*4       lng(10,20)
integer*4    attr(4)
integer*4    start(2)
integer*4    stride(2)
integer*4    count(2)
integer*4    len(4)

integer    HE5F_ACC_RDWR
parameter    (HE5F_RDWR=100)

integer    FAIL
parameter    (FAIL=-1)

Open HDF swath file, "swath.h5"
-----------------------------

swfid = he5_swopen("swath.h5",HE5F_ACC_RDWR)
if (swfid .NE. FAIL) then
    swid = he5_swattach(swfid, "Swath1")
    if (swid .NE. FAIL) then

Read the entire Longitude field
-------------------------------

    start(1) = 0
    start(2) = 0
    stride(1) = 1
    stride(2) = 1
    count(1) = 10
    count(2) = 20

    status = he5_swrdfld(swid,"Longitude",
        1, start,stride,count,lng)

    do i=1,20
        do j=1,10
            write(*,'(i,j,1x,10f8.4)') i,j,lng(j,i)
        enddo
    enddo

Read data from the Profile
--------------------------

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

    status = he5_prread(swid,"Profile-2000",start,stride,count,
        len,buffer_out)
    write(*,*)


write(*,*) 'Status from he5_prread: ',status

Display the Profile data
------------------------
do i=1,4
   write(*,*) 'len(',i,'): ',len(i)
enddo
write(*,*)
write(*,*) 'buffer_out: '
write(*,*) buffer_out
write(*,*)

j = 0
do i=1,25
   element1(i) = buffer_out(i)
   j = j + 1
enddo
write(*,*) '1st element: '
write(*,*) element1
write(*,*)

j1 = j
do i=1,50
   element2(i) = buffer_out(j1 + i)
   j = j + 1
enddo
write(*,*) '2nd element: '
write(*,*) element2
write(*,*)

j1 = j
do i=1,75
   element3(i) = buffer_out(j1 + i)
   j = j + 1
enddo
write(*,*) '3rd element: '
write(*,*) element3
write(*,*)

j1 = j
do i=1,100
   element4(i) = buffer_out(j1 + i)
   j = j + 1
enddo
write(*,*) '4th element: '
write(*,*) element4
write(*,*)

Read Attribute
-------------
status = he5_swrdattr(swid, "TestAttr", attr)
do i=1,4
   write(*,*) 'Attribute Element', i, ':', attr(i)
enddo
endif
endif
Detach from swath

status = he5_swdetach(swid)
write(*,*) 'Status from he5_swdetach: ',status

Close the file

status = he5_swclose(swfid)
write(*,*) 'Status from he5_swclose: ',status

stop
end

Example 5

In this example we (1) open the "swath.he5" HDF-EOS file, (2) attach to the "Swath1" swath, and (3) subset data from the "Spectra" field

program he5_sw_subsetF_32

integer status
integer he5_swopen
integer he5_swattach
integer he5_swextper
integer he5_swperinfo
integer he5_swreginfo
integer he5_swdefboxreg
integer he5_swdefboxperi
integer he5_swdefboxperi
integer he5_swdefboxperi
integer he5_swclose
integer swfid
integer swid
integer rank
integer ntype
integer regionid
integer periodid
integer*4 dims(8)
integer*4 size
real*8 cornerlon(2)
real*8 cornerlat(2)
real*8 datbuf(40,20,15)
real*8 timebuf(20)
real*8 t1
real*8 t2
integer HE5F_ACC_RDONLY
parameter (HE5F_ACC_RDONLY=101)
integer HE5_HDFE_MIDPOINT
parameter (HE5_HDFE_MIDPOINT=0)
integer HE5_HDFE_INTERNAL
parameter (HE5_HDFE_INTERNAL=0)

Open HDF-EOS swath file, "swath.he5"
swfid = he5_swopen("swath.he5", HE5F_ACC_RDONLY)
if (swfid .NE. FAIL) then
  swid = he5_swattach(swfid, "Swath1")
  if (swid .NE. FAIL) then
    cornerlon(1) = 3.
    cornerlat(1) = 5.
    cornerlon(2) = 7.
    cornerlat(2) = 12.
    regionid = he5_swdefboxreg(swid, cornerlon, 1, cornerlat, HE5_HDFE_MIDPOINT)
    write(*,*) regionid, swid
    status = he5_swreginfo(swid, regionid, "Spectra", ntype, 1, rank, dims, size)
    write(*,*) dims(1), dims(2), dims(3), rank, ntype, size
    status = he5_swextreg(swid, regionid, "Spectra", 1, HE5_HDFE_INTERNAL, datbuf)
    do 10 i = 1, size/8
      write(*,*) i, tmebuf(i)
  endif
  endif
detach from swath
  status = he5_swdetach(swid)
close the file
  status = he5_swclose(swfid)
endif
stop
end
Example 6

In this program we retrieve (1) information about the dimensions, (2) the dimension mappings (geolocation relations), and (3) the swath fields.

```fortran
program he5_sw_infoF_32
implicit none

integer   i
integer   status
integer   swfid, swid
integer   he5_swopen
integer   he5_swattach
integer   he5_swfdinfo
integer   he5_swmapinfo
integer   he5_swdetach
integer   he5_swclose
integer   rank(32)
integer   ntype(32)
integer   rk
integer   nt

integer*4   he5_swinqdims
integer*4   he5_swinqmaps
integer*4   he5_swinqgflds
integer*4   he5_swinqdflds
integer*4   he5_swdiminfo
integer*4   he5_swimapinfo
integer*4   he5_swinqimaps
integer*4   n
integer*4   offset
integer*4   incr
integer*4   ndims
integer*4   nmaps
integer*4   nflds
integer*4   dims(32)
integer*4   off(32)
integer*4   inc(32)
integer*4   sizes(8)
integer*4   indx(32)
integer*4   dimsize

character*72   dimname
character*72   dimmap
character*72   dimlist
character*72   maxdimlist
character*72   fieldlist

HE5F_ACC_RDONLY
parameter (HE5F_ACC_RDONLY=101)

FAIL
parameter (FAIL=-1)
```
Open the Swath file for "read only" access
------------------------------------------
    swfid = he5_swopen("swath.h5", HE5F_ACC_RDONLY)
    if (swfid .NE. FAIL) then

Attach the swath
----------------
    swid = he5_swattach(swid, "Swath1")
    if (swid .NE. FAIL) then

Inquire Dimensions
-------------------
    ndims = he5_swinqdims(swid, dimname, dims)
    write(*,*) 'Dimension list: ', dimname
    do i = 1,ndims
        write(*,*) 'dim size: ', dims(i)
    enddo
    write(*,*)

Inquire Dimension Mappings
--------------------------
    nmaps = he5_swinqmaps(swid, dimmap, off, inc)
    write(*,*) 'Dimension map: ', dimmap
    do i = 1,nmaps
        write(*,*) 'offset increment: ', off(i), inc(i)
    enddo
    write(*,*)

Inquire Indexed Dimension Mappings
----------------------------------
    nmaps = he5_swinqimaps(swid, dimmap, sizes)
    write(*,*) 'Index Dimension map: ', dimmap
    do i=1,nmaps
        write(*,*) 'sizes: ', sizes(i)
    enddo
    write(*,*)

Inquire Geolocation Fields
--------------------------
    nflds = he5_swinqgflds(swid, fieldlist, rank, ntype)
    write(*,*) 'Geolocation fieldlist: ', fieldlist
    do i=1,nflds
        write(*,*) 'field rank & datatype: ',rank(i),ntype(i)
    enddo
    write(*,*)

Inquire Data Fields
-------------------
    nflds = he5_swinqdflds(swid, fieldlist, rank, ntype)
    write(*,*) 'Data Fieldlist: ', fieldlist
    do i=1,nflds
        write(*,*) 'field rank & datatype: ',rank(i),ntype(i)
    enddo
    write(*,*)

Get info on "GeoTrack" dim
--------------------------
dimsize = he5_swdiminfo(swid, "GeoTrack")
write(*,*) 'Size of GeoTrack: ', dimsize
write(*,*)
c    Get info on "GeoTrack/Res2tr" mapping
    -----------------------------
    status = he5_swmapinfo(swid, "GeoTrack", "Res2tr",
    1         offset, incr)
    write(*,*) 'Mapping Offset: ', offset
    write(*,*) 'Mapping Increment: ', incr
    write(*,*)
c    Get info on "IndxTrack/Res2tr" indexed mapping
    ----------------------------------------------
n = he5_swimapinfo(swid, "IndxTrack", "Res2tr", indx)
do i=1,n
       write(*,*) 'Index Mapping Entry ', i, indx(i)
endo
write(*,*)
c    Get info on "Longitude" Field
    -----------------------------
    status = he5_swfldinfo(swid, "Longitude", rk, dims, nt,
    1         dimlist, maxdimlst)
    write(*,*) 'Longitude Rank: ', rk
    write(*,*) 'Longitude NumberType: ', nt
    write(*,*) 'Longitude Dimlist: ', dimlist
    write(*,*) 'Longitude Max Dimlist: ', maxdimlst
    do i=1,rk
       write(*,*) 'Dimension ', i, dims(i)
endo
dendif
dendif
c    Detach from swath
    -----------------
    status = he5_swdetach(swid)
c    Close the file
    -------------
    status = he5_swclose(swid)
stop
eend

7.3 Grid Examples

This section contains several examples of the use of the Grid interface from both C and FORTRAN programs. First, there are simple examples in C and FORTRAN which demonstrate the use of most of the functions in the Grid interface.
7.3.1 Creating a Simple Grid

The following C and FORTRAN programs each create, define, and write a simple Grid data set to an HDF-EOS file using the HDF-EOS Grid interface.

7.3.1.1 A C Example of a Simple Grid Creation

Example 1

/*
* In this example we will open an HDF-EOS file and create UTM and Polar
* Stereographic grid structures within the file.
*/

/* he5_gd_setup */

#include <HE5_HdfEosDef.h>

main()
{
    herr_t status = FAIL;
    hid_t gfid = FAIL;
    hid_t GDid = FAIL;
    hid_t GDid2 = FAIL;
    hid_t GDid3 = FAIL;
    int i, j;
    int zonecode, projcode, spherecode, dummy = 0;
    long xdim, ydim;
    double projparm[16], uplft[2], lowrgt[2];

    /*
    * We first open the HDF grid file, "Grid.he5". Because this file
    * does not already exist, we use the H5F_ACC_TRUNC access code in the
    * open statement. The GDopen routine returns the grid file id, gfid,
    * which is used to identify the file in subsequent routines in the
    * library.
    */
    gfid = HE5_GDopen("Grid.he5", H5F_ACC_TRUNC);

    /*
    * Create UTM Grid
    * Region is bounded by 54 E and 60 E longitude and 20 N and 30 N latitude.
    * UTM Zone 40
    * Use default spheriod (Clarke 1866 - spherecode = 0)
    */
* Grid into 120 bins along x-axis and 200 bins along y-axis
*     (approx 3' by 3' bins)
*/

zonecode = 40;
spherecode = 0;

/* Upper Left and Lower Right points in meters */
/* ---------------------------------------------- */
uplft[0] = 210584.50041;
uplft[1] = 3322395.95445;
lowrgt[0] = 813931.10599;
lowrgt[1] = 2214162.53278;

xdim = 120;
ydim = 200;

GDid = HE5_GDcreate(gdfid, "UTMGrid", xdim, ydim, uplft, lowrgt);
    printf("Grid ID returned by HE5_GDcreate : %d \n", GDid);
    status = HE5_GDdefproj(GDid, HE5_GCTP_UTM, zonecode, spherecode, projparm);
    printf("status returned by HE5_GDdefproj(..."HE5_GCTP_UTM"...) : %d \n", status);
    /* Define "Time" Dimension */
    status = HE5_GDdefdim(GDid, "Time", 10);
    printf("status returned by HE5_GDdefdim("Time")... : %d \n", status);
    /* Define "Unlim" Dimension */
    status = HE5_GDdefdim(GDid, "Unlim", H5S_UNLIMITED);
    printf("status returned by HE5_GDdefdim("Unlim")... : %d \n", status);
    /* Create polar stereographic grid */
    * Northern Hemisphere (True Scale at 90 N, 0 Longitude below pole)
    * Use International 1967 spheriod (spherecode = 3)
    * Grid into 100 bins along x-axis and y-axis
*/
spherecode = 3;

/* Define GCTP Projection Parameters */
/* ----------------------------------- */
for (i = 0; i < 16; i++)
    projparm[i] = 0;
/* Set Longitude below pole & true scale in DDDMMSSS.SSS format */
    projparm[4] = 0.0;
    projparm[5] = 90000000.00;

xdim = 100;
ydim = 100;
GDid2 = HE5_GDcreate(gdfid, "PolarGrid", xdim, ydim, NULL, NULL);
printf("Grid ID returned by HE5_GDcreate() : %d \n", GDid2);

status = HE5_GDdefproj(GDid2, HE5_GCTP_PS, dummy, spherecode, projparm);
printf("status returned by HE5_GDdefproj(...) : %d \n", status);

status = HE5_GDdeforigin(GDid2, HE5_HDFE_GD_LR);
printf("status returned by HE5_GDdeforigin() : %d \n", status);

/* Define "Bands" Dimension */
status = HE5_GDdefdim(GDid2, "Bands", 3);
printf("status returned by HE5_GDdefdim(...) : %d \n", status);

/*
 * Create geographic (linear scale) grid
 * 0 - 15 degrees longitude, 20 - 30 degrees latitude
 */

xdim = 60;
ydim = 40;

uplft[0] = HE5_EHconvAng(0., HE5_HDFE_DEG_DMS);
uplft[1] = HE5_EHconvAng(30., HE5_HDFE_DEG_DMS);
lowrgt[0] = HE5_EHconvAng(15., HE5_HDFE_DEG_DMS);
lowrgt[1] = HE5_EHconvAng(20., HE5_HDFE_DEG_DMS);

GDid3 = HE5_GDcreate(gdfid, "GEOGrid", xdim, ydim, uplft, lowrgt);
printf("Grid ID returned by HE5_GDcreate() : %d \n", GDid3);

status = HE5_GDdefproj(GDid3, HE5_GCTP_GEO, dummy, dummy, NULL);
printf("status returned by HE5_GDdefproj(...) : %d \n", status);

/*
 * We now close the grid interface with the GDdetach routine. This step
 * is necessary to properly store the grid information within the file
 * AND SHOULD BE DONE BEFORE WRITING OR READING DATA TO OR FROM THE FIELD.
 */
status = HE5_GDdetach(GDid);
status = HE5_GDdetach(GDid2);
status = HE5_GDdetach(GDid3);

/*
 * Finally, we close the grid file using the HE5_GDclose routine. This will
 * release the grid file handles established by HE5_GDopen.
 */
status = HE5_GDclose(gdfid);
return 0;
Example 2

/*
 * In this example we will (1) open the "Grid.h5" HDF-EOS file, (2) attach to
 * the "Grid1" grid, and (3) define the grid fields.
 */

/* he5_gd_definefields */

#include <HE5_HdfEosDef.h>

main()
{
    herr_t status = FAIL;
    hid_t gdfid = FAIL;
    hid_t GDid1 = FAIL;
    hid_t GDid2 = FAIL;
    float fillval1 = -7.;
    float fillval2 = -9999.;

    /*
     * We first open the HDF-EOS grid file, "Grid.he5". Because this file
     * already exist and we wish to write to it, we use the H5F_ACC_RDWR access
     * code in the open statement. The HE5_GDopen routine returns the grid file
     * id, gdfid, which is used to identify the file in subsequent routines.
     */
    gdfid = HE5_GDopen("Grid.he5", H5F_ACC_RDWR);
    if (gdfid != FAIL)
    {
        GDid1 = HE5_GDattach(gdfid, "UTMGrid");

        status = HE5_GDdeffield(GDid1, "Pollution", "Time,YDim,XDim", NULL,
                                H5T_NATIVE_FLOAT, 0);
        printf("Status returned by HE5_GDdeffield(...) : %d \n", status);

        status = HE5_GDdeffield(GDid1, "Vegetation", "YDim,XDim", NULL,
                                H5T_NATIVE_FLOAT, 0);
        printf("Status returned by HE5_GDdeffield(...) : %d \n", status);

        status = HE5_GDsetfillvalue(GDid1, "Pollution", H5T_NATIVE_FLOAT,
                                    &fillval1);
        printf("Status returned by HE5_GDsetfillvalue(...) : %d \n", status);
    }
}
Example 3

/* In this example we will (1) open the "Grid.he5" HDF-EOS file, (2) attach to the "UTMGrid", and (3) write data to the "Vegetation" field. We will then attach to the "PolarGrid" and write to the "Temperature" field. */

/* he5_gd_writedata */

#include <HE5_HdfEosDef.h>

main()
{
    herr_t status = FAIL;
    int i, j;
    int grpattr[3] = {3,7,11}; /* group attr */
hid_t           gdfid = FAIL;
hid_t           GDid  = FAIL;
float           flt = 3.1415; /* global attr */
float           attr[4] = {1.1,2.2,3.3,4.4}; /* local attr */
float           veg[200][120];
float           temp[100][100];
hssize_t        start[3];
hsize_t         edge[3];

/* Fill veg array */
for (i = 0; i < 200; i++)
    for (j = 0; j < 120; j++)
        veg[i][j] = (float)(10+i);
/* Fill temp array */
for (i = 0; i < 100; i++)
    for (j = 0; j < 100; j++)
        temp[i][j] = (float)(100*i+j);

/* Open the HDF grid file, "Grid.he5". */
gdfid = HE5_GDopen("Grid.he5", H5F_ACC_RDWR);
if (gdfid != FAIL)
{
    /* Attach the "UTMGrid". */
    GDid = HE5_GDattach(gdfid, "UTMGrid");
    if (GDid != FAIL)
    {
        /* Data Field "Vegetation" */
        /* ----------------------- */
        start[0] = 0;     start[1] = 0;
        edge[0]  = 200;   edge[1]  = 120;
        status = HE5_GDwritefield(GDid, "Vegetation", start, NULL, edge,
        veg);
        printf("Status returned by HE5_GDwritefield() : %d \n",
        status);

        /* Global attribute */
        /* ---------------- */
        edge[0] = 1;
        status = HE5_GDwriteattr(GDid, "GlobalAttribute",
        H5T_NATIVE_FLOAT, edge, &flt);
        printf("Status returned by HE5_GDwriteattr() : %d \n",
        status);

        /* Group attribute */
        /* ---------------- */
        edge[0] = 3;
        status = HE5_GDwritegrpattr(GDid, "GroupAttribute",
        H5T_NATIVE_INT, edge, grpattr);
Example 4

/*
* In this example we will (1) open the "Grid.he5" HDF-EOS file, (2) attach to
* the "UTMGrid", (3) read data from the "Vegetation" field and (4) read
* global, group, and local attributes.
*/

/* he5_gd_readdata */

#include <HE5_HdfEosDef.h>

main()
{
    herr_t status = FAIL;
    int i, j;
    int grpattr[3] = {-9,-9,-9}; /* group attribute */
    hid_t gdfid = FAIL;
    hid_t GDid = FAIL;
}

printf("Status returned by HE5_GDwritegrpattr() : %d \n",
    status);

/* Local attribute */
/* ---------------- */
    edge[0] = 4;
    status = HE5_GDwritelocattr(GDid, "Vegetation",
    "LocalAttribute", H5T_NATIVE_FLOAT, edge, attr);
    printf("Status returned by HE5_GDwritelocattr() : %d \n",
    status);
}

status = HE5_GDdetach(GDid);

GDid = HE5_GDattach(gdfid, "PolarGrid");
if (GDid != FAIL)
{
    /* Data field "Temperature" */
    /* ------------------------ */
        start[0] = 0;     start[1] = 0;
        edge[0] = 100;    edge[1] = 100;
        status = HE5_GDwritefield(GDid, "Temperature", start, NULL,
    edge, temp);
        printf("Status returned by HE5_GDwritefield() : %d \n",
    status);
}

status = HE5_GDdetach(GDid);

}

status = HE5_GDclose(gdfid);
return 0;
}
float flt = -999.; /* global attribute */
float veg[200][120];
hssize_t start[2] = {0, 0};
hsize_t edge[2] = {200, 100};

/* Open the HDF grid file, "Grid.he5". */
gdfid = HE5_GDopen("Grid.he5", H5F_ACC_RDWR);
if (gdfid != FAIL) {
    /* Attach the "UTMGrid". */
    GDid = HE5_GDattach(gdfid, "UTMGrid");
    if (GDid != FAIL) {
        status = HE5_GDreadfield(GDid, "Vegetation", start, NULL, edge,
        veg);
        printf("Status returned by HE5_GDreadfield() : %d \n", status);
        for (i = 0; i < 5; i++)
            for (j = 0; j < 10; j++)
                printf(" \t%f \n", veg[i][j]);

        status = HE5_GDreadattr(GDid, "GlobalAttribute", &flt);
        printf("Status returned by HE5_GDreadattr() : %d \n", status);

        printf("\tGlobal attribute reads: \n");
        printf("\t%f \n", flt);

        status = HE5_GDreadgrppattr(GDid, "GroupAttribute", grpattr);
        printf("Status returned by HE5_GDreadgrppattr() : %d \n", status);

        printf("\tGroup attribute reads: \n");
        for (i = 0; i < 3; i++)
            printf("\t%d \n", grpattr[i]);

        status = HE5_GDreadlocattr(GDid, "Vegetation","LocalAttribute",
        attr);
        printf("Status returned by HE5_GDreadlocattr() : %d \n", status);

        printf("\tLocal attribute reads: \n");
        for (i = 0; i < 4; i++)
            printf("\t%f \n", attr[i]);
    }
}

status = HE5_GDdetach(GDid);
status = HE5_GDClose(gdfid);
return 0;

Example 5
/* In this example we will (1) open the "Grid.he5" HDF-EOS file, (2) attach to * the "PolarGrid", and (3) subset data from the "Temperature" field. */

#include <HE5_HdfEosDef.h>

main()
{
    herr_t status = FAIL;
    int rank = FAIL;
    int i;

    hid_t gdfid = FAIL;
    hid_t GDid = FAIL;
    hid_t regionID = FAIL;

    hid_t *ntype;
    long size;
    hsize_t dims[8];
    float *datbuf;
    double cornerlon[2], cornerlat[2];
    double upleft[2], lowright[2];

    /* Open the HDF-EOS grid file, "Grid.he5". */
    gdfid = HE5_GDopen("Grid.he5", H5F_ACC_RDWR);
    if (gdfid != FAIL)
    {
        GDid = HE5_GDattach(gdfid, "PolarGrid");
        if (GDid != FAIL)
        {
            cornerlon[0] = 57.; cornerlat[0] = 23.;
            cornerlon[0] = 0.; cornerlat[0] = 90.;
            cornerlon[1] = 90.; cornerlat[1] = 0.;

            regionID = HE5_GDdefboxregion(GDid, cornerlon, cornerlat);
            printf("Region ID returned by HE5_GDdefboxregion() : %d \n", regionID);

            ntype = (hid_t *)calloc(1, sizeof(hid_t));
            status = HE5_GDregioninfo(GDid, regionID, "Temperature", ntype, &rank, dims, &size, upleft, lowright);
            printf("Status returned by HE5_GDregioninfo() : %d \n", status);
        }
    }
}
printf("Byte size of region data buffer:\n", (int)size);

datbuf = (float *)malloc(size);
status = HE5_GDextractregion(GDid, regionID, "Temperature", datbuf);
printf("Status returned by HE5_GDextractregion() : %d \n", status);
printf("First 20 values of data buffer: \n");
for (i = 0; i < 20; i++)
    printf("\t%f \n", datbuf[i]);
free(datbuf);
free(ntype);
}
}
status = HE5_GDdetach(GDid);
printf("Status returned by HE5_GDdetach() : %d \n", status);
status = HE5_GDclose(gdfid);
printf("Status returned by HE5_GDclose() : %d \n", status);
return 0;
}

Example 6

/*
 * In this example we will retrieve information about (1) dimensions,
 * (2) dimension mappings (geolocation relations), (3) grid fields,
 * and (4) (global/group/local) grid attributes.
 */

/* he5_gd_info */

#include <HE5_HdfEosDef.h>

main()
{

    herr_t    status = FAIL;
    int       i, rank[32];
    int       projcode, zonecode, spherecode;
    int       ndim = FAIL, nflds = FAIL;
    hid_t     gdfid = FAIL;
    hid_t     GDid1 = FAIL;
    hid_t     GDid2 = FAIL;
    hid_t     *ntype;
    hsize_t   Dims[32], dimsize, count = 0;
    long      xdimsize, ydimsize, n, strbufsize;
    double    upleftpt[2], lowrightpt[2], projparm[16];
char version[80];
char dimname[1024], fieldlist[1024];
char attrlist[80];

/*
 * Open the Grid File for read only access
 */
gdfid = HE5_GDopen("Grid.he5", H5F_ACC_RDONLY);
if (gdfid != FAIL) {
    HE5_EHgetversion(gdfid, version);
    printf("Version:
%s
", version);

    /* Attach the grid */

    GDId1 = HE5_GDattach(gdfid, "UTMGrid");
    GDId2 = HE5_GDattach(gdfid, "PolarGrid");

    ndim = HE5_GDinqdims(GDId1, dimname, Dims);
    printf("Dimension list (UTMGrid): %s
", dimname);
    for (i = 0; i < ndim; i++) printf("dim size: %lu
", (unsigned long)Dims[i]);

    ndim = HE5_GDinqdims(GDId2, dimname, Dims);
    printf("Dimension list (PolarGrid): %s
", dimname);
    for (i = 0; i < ndim; i++) printf("dim size: %lu
", (unsigned long)Dims[i]);

    dimsize = HE5_GDdiminfo(GDId1, "Time");
    printf("Size of "Time" Array: %lu
", (unsigned long)dimsize);

    dimsize = HE5_GDdiminfo(GDId2, "Bands");
    printf("Size of "Bands" Array: %lu
", (unsigned long)dimsize);

    status = HE5_GDgridinfo(GDId1, &xdimsize, &ydimsize, upleftpt, lowrightpt);
    printf("X dim size, Y dim size (UTMGrid): %li %li
", xdimsize, ydimsize);
    printf("Up left pt (UTMGrid): %lf %lf
", upleftpt[0], upleftpt[1]);
    printf("Low right pt (UTMGrid): %lf %lf
", lowrightpt[0], lowrightpt[1]);

    status = HE5_GDgridinfo(GDId2, &xdimsize, &ydimsize, upleftpt, lowrightpt);
    printf("X dim size, Y dim size (PolarGrid): %li %li
", xdimsize, ydimsize);
    printf("Up left pt (PolarGrid): %lf %lf
", upleftpt[0], upleftpt[1]);
    printf("Low right pt (PolarGrid): %lf %lf
", lowrightpt[0], lowrightpt[1]);

    status = HE5_GDprojinfo(GDId1, &projcode, &zonecode, &spherecode, NULL);
    printf("projcode, zonecode (UTMGrid): %d %d
", projcode, zonecode);
    printf("spherecode (UTMGrid): %d
", spherecode);
status = HE5_GDprojinfo(GDid2, &projcode, NULL, &spherecode, projparm);
printf("projcode (PolarGrid): %d\n", projcode);
printf("spherecode (PolarGrid): %d\n", spherecode);
for (i = 0; i < 13; i++)
    printf("Projection Parameter: %d %lf\n",i,projparm[i]);

ntype = (hid_t *)calloc(10, sizeof(hid_t));
flds = HE5_GDinqfields(GDid1, fieldlist, rank, ntype);
if (nflds != FAIL)
{
    printf("Data fields (UTMGrid): %s\n", fieldlist);
    for (i = 0; i < nflds;i++)
        printf("Rank:   %i   Data type:   %i\n",rank[i],
            (int)ntype[i]);
}

nflds = HE5_GDinqfields(GDid2, fieldlist, rank, ntype);
if (nflds != FAIL)
{
    printf("Data fields (PolarGrid): %s\n", fieldlist);
    for (i = 0; i < nflds;i++)
        printf("Rank:   %i   Data type:   %i\n",rank[i],
            (int)ntype[i]);
}

status = HE5_GDfieldinfo(GDid2, "Spectra", rank, Dims, ntype, dimname, NULL);
printf("Spectra rank: %d\n",rank[0]);
printf("Spectra dimensions: \n");
for (i = 0; i < rank[0]; i++)
    printf("   %lu\n",(unsigned long)Dims[i]);
printf("Spectra dimension list: \n");
printf("\n"), dimname);

printf("\n");
printf("Global Attributes \n");
status = HE5_GDattrinfo(GDid1, "GlobalAttribute", ntype, &count);
printf("\tNumber of attribute elements: %lu \n", (unsigned long)count);
printf("\tData type of attribute: %d \n", (int)*ntype);

printf("\n");
printf("Group Attributes \n");
status = HE5_GDgrpattrinfo(GDid1, "GroupAttribute", ntype, &count);
printf("\tNumber of attribute elements: %lu \n", (unsigned long)count);
printf("\tData type of attribute: %d \n", (int)*ntype);

printf("\n");
printf("Local Attributes \n");
status = HE5_GDlocattrinfo(GDid1, "Vegetation", "LocalAttribute", ntype, &count);
printf("\tNumber of attribute elements: %lu \n", (unsigned long)count);
printf("\tData type of attribute: %d \n", (int)*ntype);
printf(" \n");
printf("Global Attributes \n");
n = HE5_GDingattrs(GDid1, NULL, &strbufsize);
printf("\tNumber of attributes: %li\n", n);
printf("\tSize (in bytes) of attribute list: %li\n", strbufsize);
n = HE5_GDingattrs(GDid1, attrlist, &strbufsize);
printf("\tAttribute list: %s\n", attrlist);

printf(" \n");
printf("Group Attributes \n");
n = HE5_GDinggrpattrs(GDid1, NULL, &strbufsize);
printf("\tNumber of attributes: %li\n", n);
printf("\tSize (in bytes) of attribute list: %li\n", strbufsize);
n = HE5_GDinggrpattrs(GDid1, attrlist, &strbufsize);
printf("\tAttribute list: %s\n", attrlist);

printf(" \n");
printf("Local Attributes \n");
n = HE5_GDinglocattrs(GDid1, "Vegetation", NULL, &strbufsize);
printf("\tNumber of attributes: %li\n", n);
printf("\tSize (in bytes) of attribute list: %li\n", strbufsize);
n = HE5_GDinglocattrs(GDid1, "Vegetation", attrlist, &strbufsize);
printf("\tAttribute list: %s\n", attrlist);

free(ntype);

n = HE5_GDnentries(GDid1, HE5_HDFE_NENTDIM, &strbufsize);
printf("Number of dimension entries (UTMGrid): %li\n", n);
printf("Length of Dimension List (UTMGrid): %li\n", strbufsize);

n = HE5_GDnentries(GDid1, HE5_HDFE_NENTDFLD, &strbufsize);
printf("Number of data fields (UTMGrid): %li\n", n);
printf("Length of Field List (UTMGrid): %li\n", strbufsize);

}

status = HE5_GDdetach(GDid1);
status = HE5_GDdetach(GDid2);
status = HE5_GDclose(gdfid);

return 0;
}

7.3.1.2 A FORTRAN Example of a Simple Grid Creation

Example 1

c In this example we open an HDF-EOS file and create UTM, Polar

c Stereographic, and Geographic grids within the file

program he5_gd_setupF_32
integer status, gdfid
integer he5_gdcreate, he5_gdopen
integer he5_gddefdim, he5_gddefproj
integer he5_gddeforigin
integer he5_gddetach, he5_gdclose
integer gid, gid2, gid3
integer zonecode, spherecode

integer*4 xdim, ydim, dim
real*8 uplft(2), lowrgt(2)
real*8 projparm(16), he5_ehconvang

integer HE5F_ACC_TRUNC
parameter (HE5F_ACC_TRUNC=102)
integer HE5_GCTP_UTM
parameter (HE5_GCTP_UTM=1)
integer HE5_GCTP_PS
parameter (HE5_GCTP_PS=6)
integer HE5_GCTP_GEO
parameter (HE5_GCTP_GEO=0)
integer HE5_HDFE_GD_LR
parameter (HE5_HDFE_GD_LR=3)
integer HE5_HDFE_DEG_DMS
parameter (HE5_HDFE_DEG_DMS=3)

c Open the HDF grid file, "grid.he5"
---------------------------------  
gdfid = he5_gdopen('grid.he5',HE5F_ACC_TRUNC)

Create UTM Grid
--------------

Region is bounded by 54 E and 60 E longitudes and 20 N and 30 N latitudes. UTM Zone: 40.
Use default spheriod (Clarke 1866: spherecode = 0)

Grid into 120 bins along x-axis and 200 bins along y-axis (approx 3' by 3' bins).

zonecode = 40
spherecode = 0

Upper Left and Lower Right points in meters
------------------------------------------
uplft(1) = 210584.50041
uplft(2) = 3322395.95445
lowrgt(1) = 813931.10959
lowrgt(2) = 2214162.53278

xdim = 120
ydim = 200

Define GCTP Projection Parameters
--------------------------------
doi=1,16
  projparm(i) = 0.d0
enddo

gdid = he5_gdcreate(gdfid, "UTMGrid", xdim, ydim, uplft, lowrgt)
status = he5_gddefproj(gdid,GCTP_UTM,zonecode,spherecode,projparm)

c Define "Time" Dimension

----------------------------------- 
dim = 10
status = he5_gddefdim(gdid, "Time", dim)
c Create polar stereographic grid
----------------------------------- 

c Northern Hemisphere (True Scale at 90 N, 0 Longitude below pole). 
Use International 1967 spheriod (spherecode = 3) 
Grid into 100 bins along x-axis and y-axis. 
c ---------------------------------------

xdim = 100
ydim = 100
spherecode = 3

c Set Longitude below pole & true scale in DDDMMSSS.SSS format)
----------------------------------------------
projparm(5) = 0.0
projparm(6) = 90000000.00

c Use default boundaries for Polar Stereographic (hemisphere)
---------------------------------------------
uplft(1) = 0
uplft(2) = 0
lowrgt(1) = 0
lowrgt(2) = 0
zonecode = 0

gdid2 = he5_gdcreate(gdfid,"PolarGrid",xdim,ydim,uplft,lowrgt)
status = he5_gddefproj(gdid2,HE5_GCTP_PS,zonecode, 
1spherecode,projparm)
status = he5_gddeforigin(gdid2, HE5_HDFE_GD_LR)
c Define "Bands" Dimension
----------------------------------- 
dim = 3
status = he5_gddefdim(gdid2, "Bands", dim)
c Create geographic (linear scale) grid
----------------------------------- 

0-15 degrees longitude, 20-30 degrees latitude 

xdim = 60
ydim = 40
uplft(1) = he5_ehconvAng(0.d0, HE5_HDFE_DEG_DMS)
uplft(2) = he5_ehconvAng(30.d0, HE5_HDFE_DEG_DMS)
lowrgt(1) = he5_ehconvAng(15.d0, HE5_HDFE_DEG_DMS)
lowrgt(2) = he5_ehconvAng(20.d0, HE5_HDFE_DEG_DMS)

do i=1,16
    projparm(i) = 0.d0
enddo

zonecode   = 0
spherecode = 0

gdid3 = he5_gdcreate(gdfid,"GEOGrid",xdim,ydim,uplft,lowrgt)
status=he5_gddefproj(gdid3,HE5_GCTP_GEO,zonecode,spherecode,projparm)

c  Detach from the created grids

----------------------------
status = he5_gdDetach(gdid)
status = he5_gdDetach(gdid2)
status = he5_gdDetach(gdid3)

---------------------------
status = he5_gdclose(gdfid)

stop
end

Example 2

In this example we open an HDF-EOS file, attached to the specified grids, define fields, and set fill values, detached from the grids, and close the file

program he5_gd_definefldsF_32

integer   status, he5_gddeffld, he5_gdsetfill
integer   he5_gdattach, he5_gddetach, he5_gdclose
integer   gdfid, gdid1, gdid2, he5_gdopen
real*4    fillval1, fillval2

integer   HE5F_ACC_RDWR
parameter (HE5F_ACC_RDWR=100)

integer   HE5T_NATIVE_FLOAT
parameter (HE5T_NATIVE_FLOAT=10)

integer   HE5T_NA_TIVE_DOUBLE
parameter (HE5T_NATIVE_DOUBLE=11)

fillval1 = -7.0
fillval2 = -9999.0

Open HDF-EOS file "grid.he5"
---------------------------
gdfid = he5_gdopen("grid.he5", HE5F_ACC_RDWR)

c     Attach to the UTM grid

c     ----------------------
gdid1 = he5_gdattach(gdfid, "UTMGrid")

c     Define Fields

c     -------------
status = he5_gddeffld(gdid1, "Pollution", "XDim,YDim,Time",
1" ", HE5_HDFE_NATIVE_FLOAT, 0)

status = he5_gddeffld(gdid1, "Vegetation", "XDim,YDim",
1" ", HE5T_NATIVE_FLOAT, 0)

c     Attach to the POLAR grid

c     ------------------------
gdid2 = he5_gdattach(gdfid, "PolarGrid")

c     Define Fields

c     -------------
status = he5_gddeffld(gdid2, "Temperature", "XDim,YDim",
1" ", HE5T_NATIVE_FLOAT, 0)

status = he5_gddeffld(gdid2, "Pressure", "XDim,YDim",
1" ", HE5T_NATIVE_FLOAT, 0)

c     Set fill value for "Pressure" field

c     -------------------------------
status = he5_gdsetfill(gdid2, "Pressure", HE5T_NATIVE_FLOAT,
1fillval2)

c     Define Fields

c     -------------
status = he5_gddeffld(gdid2, "Soil Dryness", "XDim,YDim",
1" ", HE5T_NATIVE_FLOAT, 0)

status = he5_gddeffld(gdid2, "Spectra", "XDim,YDim,Bands",
1" ", HE5T_NATIVE_DOUBLE, 0)

c     Detach from the grids

c     ---------------------
status = he5_gddetach(gdid1)
status = he5_gddetach(gdid2)

c     Close the file

c     --------------
status = he5_gdclose(gdfid)

stop
end

Example 3

c In this example we open HDF-EOS grid file, attach to the UTM grid,
c and write data to the “Vegetation” field. Also, we attach to the
Polar grid and write data to the "Temperature" field.

```fortran
program he5_gd_writedataF_32
integer i, j, status, he5_gdwrfld
integer he5_gdwrattr, he5_gddetach, he5_gdclose
integer gdfid, gdid, he5_gdopen, he5_gdattach
integer*4 start(2), stride(2), count(2)
real*4 f, veg(120,200), temp(100,100)
integer HE5F_ACC_RDWR
parameter (HE5F_ACC_RDWR=100)
integer HE5T_NATIVE_FLOAT
parameter (HE5T_NATIVE_FLOAT=10)

! Create data buffers
!
! ---------------
! do i=1,200
! do j=1,120
! veg(j,i) = 10 + i
! enddo
! enddo
! do i=1,100
! do j=1,100
! temp(j,i) = 100*(i-1) + j
! enddo
! enddo
!
!
! Open HDF-EOS file "grid.he5"
!
! -----------------------------
gdfid = he5_gdopen("grid.he5", HE5F_ACC_RDWR)
if (gdfid .ne. FAIL) then
!
! Attach to the UTM grid
!
! -----------------------------
gdid = he5_gdattach(gdfid, "UTMGrid")
if (gdid .ne. FAIL) then
    start(1) = 0
    start(2) = 0
    stride(1) = 1
    stride(2) = 1
    count(1) = 120
    count(2) = 200
!
! Write data to the field "Vegetation"
!
! -----------------------------
status = he5_gdwrfld(gdid,"Vegetation",start,
1      stride,count,veg)
f = 1
count(1) = 1
!
!
! Write global attribute "float"
!
! -----------------------------
```

```
status = he5_gdwrattr(gdid,"float", 1
   HE5T_NATIVE_FLOAT,count,f)
endif
endif

c  Detach from the grid

c  ---------------------
   status = he5_gddetach(gdid)

c  Attach to the POLAR grid

c  ------------------------
   gdid = he5_gdattach(gdfid, "PolarGrid")
   if (gdid .ne. FAIL) then
      start(1) = 0
      start(2) = 0
      stride(1) = 1
      stride(2) = 1
      count(1) = 100
      count(2) = 100
   endif

c  Write data to the "Temperature" field

c  ---------------------
   status = he5_gdwrfld(gdid,"Temperature",start,stride,count,temp)
endif

c  Detach from the grid

c  ---------------------
   status = he5_gddetach(gdid)

c  Close the file

c  -------------
   status = he5_gdclose(gdfid)

stopend

Example 4

In this example we open HDF-EOS grid file, attach to the UTM grid and read “Vegetation” field

program he5_gd_readdataF_32
integer status
integer gdfid
integer gdid

integer he5_gdopen
integer he5_gdattach
integer he5_gdrdfld
integer he5_gdrdattr
integer he5_gddetach
integer he5_gdclose

integer*4 start(2)
integer*4 stride(2)
integer*4 count(2)
c real*4 f
c real*4 veg(120,200)
integer HE5F_ACC_RDWR
parameter (HE5F_ACC_RDWR=100)

c Open HDF-EOS "grid.h5" file
-----------------------------
gdfid = he5_gdopen("grid.he5", HE5F_ACC_RDWR)
if (gdfid .ne. FAIL) then
  c Attach to the UTM grid
  ----------------------
gdid = he5_gdattach(gdfid, "UTMGrid")
  if (gdid .ne. FAIL) then
    start(1) = 0
    start(2) = 0
    stride(1) = 1
    stride(2) = 1
    count(1) = 120
    count(2) = 200
    c Read the data from "Vegetation" field
    -------------------------------------
    status = he5_gdrdfld(gdid,"Vegetation",start,stride,count,veg)
    c Read global attribute "float"
    -----------------------------
    status = he5_gdrdattr(gdid, "float", f)
    write(*,*) 'global attribute value: ', f
  endif
endif

c Detach from the grid
---------------------
status = he5_gddetach(gdid)

c Close the file
--------------
status = he5_gdclose(gdfid)
stop
end

Example 5

c In this example we will (1) open the HDF-EOS grid file, (2) attach to
the "PolarGrid" grid, and (3) subset data from the "Temperature" field.

program he5_gd_subsetF_32
integer status
integer gdfid
integer gdid
c
Open the HDF-EOS grid file, "grid.he5"
-------------------------------------
gdfid = he5_gdopen("grid.he5", HE5F_ACC_RDWR)
if (gdfid .NE. FAIL) then
   c        Attach to the POLAR grid
   ------------------
gdid = he5_gdattach(gdfid, "PolarGrid")
if (gdid .NE. FAIL) then
   cornerlon(1) = 0.
   cornerlat(1) = 90.
   cornerlon(2) = 90.
   cornerlat(2) = 0.
   c        Define box region
   ------------
   rgid = he5_gddefboxreg(gdid,cornerlon, cornerlat)
   c        Get region information
   ------------
   status = he5_gdreginfo(gdid,rgid,"Temperature",nt,
      lrk, dims, size, upleft, lowright)
   write(*,*) dims(1), dims(2), rk, nt
   c        Extract region
   ----------
   status = he5_gdextreg(gdid,rgid,"Temperature",datbuf)
endif
   c        Detach from the grid
   -----------
   status = he5_gddetach(gdid)
   c        Close the file
---

status = he5_gdclose(gdfid)

end if

stop

eend

**Example 6**

c In this example we will retrieve information about (1) dimensions, (2) dimension mappings (geolocation relations), (3) grid fields, and (4) (global/group/local) grid attributes.
c

```fortran
program he5_gd_infoF_32
    integer status
    integer i
    integer he5_gdopen
    integer he5_gdattach
    integer he5_gdflldinfo
    integer he5_gdprojinfo
    integer he5_gdgridinfo
    integer he5_gdingflds
    integer he5_gddetach
    integer he5_gdclose
    integer gdfid
    integer gdid1
    integer gdid2
    integer nflds
    integer rk(32)
    integer nt(32)
    integer spherecode
    integer projcode
    integer zonecode
    
    integer*4 he5_gddiminfo
    integer*4 he5_gdingdims
    integer*4 he5_gdnentries
    integer*4 ndim
    integer*4 dims(32)
    integer*4 xdimsize
    integer*4 ydimsize
    integer*4 dimsize
    integer*4 n, strbufsize
    
    real*8 upleftpt(2)
    real*8 lowrightpt(2)
    real*8 projparm(13)
    
    character*256 dimname
    character*256 mxdimname
    character*256 fieldlist
    
    integer HE5F_ACC_RDWR
    parameter (HE5F_ACC_RDWR=100)
```
integer       HE5_HDFE_NENTDIM
parameter     (HE5_HDFE_NENTDIM=0)

integer       HE5_HDFE_NENTDFLD
parameter     (HE5_HDFE_NENTDFLD=4)

c Open HDF-EOS "grid.he5" file
-------------------------------
gdfid = he5_gdopen('grid.he5', HE5F_ACC_RDWR)
if (gdfid .ne. FAIL) then

  Attach to the UTM and POLAR grids
-------------------------------
gdid1 = he5_gdattach(gdfid, 'UTMGrid')
gdid2 = he5_gdattach(gdfid, 'PolarGrid')

  Inquire dimensions
------------------
  ndim = he5_gdinqdims(gdid1, dimname, dims)
  write(*,*)'Dimension list (UTMGrid): ', dimname
  do i=1,ndim
      write(*,*) 'dim size: ', dims(i)
  enddo

  ndim = he5_gdinqdims(gdid2, dimname, dims)
  write(*,*)'Dimension list (PolarGrid): ', dimname
  do i=1,ndim
      write(*,*) 'dim size: ', dims(i)
  enddo

  Get the sizes of certain dimensions
-----------------------------------
  dimsize = he5_gddiminfo(gdid1, 'Time')
  write(*,*) 'Size of "Time" Array: ', dimsize

  dimsize = he5_gddiminfo(gdid2, 'Bands')
  write(*,*) 'Size of "Bands" Array: ', dimsize

  Get grid parameters
----------------------
  status = he5_gdgridinfo(gdid1,xdimsize,ydimsize,
                          1upleftpt,lowrightpt)
  write(*,*)'X dim size, Y dim size (UTMGrid): ',
              2xdimsize,ydimsize

          write(*,*) 'Up left pt (UTMGrid): ',upleftpt(1),
                      1upleftpt(2)
          write(*,*) 'Low right pt (UTMGrid): ',lowrightpt(1),
                      2lowrightpt(2)

  status = he5_gdgridinfo(gdid2,xdimsize,ydimsize,
                          1upleftpt,lowrightpt)
  write(*,*) 'X dim size, Y dim size (PolarGrid): ',
              2xdimsize,ydimsize

          write(*,*) 'Up left pt (PolarGrid): ',upleftpt(1),
                      1upleftpt(2)
          write(*,*) 'Low right pt (PolarGrid): ',lowrightpt(1),
Get projection parameters

status = he5_gdprojinfo(gdid1, projcode, zonecode, 
spherecode, projparm)
write(*,*) 'projcode,zonecode (UTMGrid): ', projcode,
zonecode
write(*,*) 'spherecode (UTMGrid): ', spherecode

status = he5_gdprojinfo(gdid2, projcode, zonecode, 
spherecode, projparm)
write(*,*) 'projcode (PolarGrid): ', projcode
write(*,*) 'spherecode (PolarGrid): ', spherecode

do i = 1, 13
write(*,*) 'Projection Parameter: ', i, projparm(i)
enddo

Get information about fields

nflds = he5_gdingflds(gdid1, fieldlist, rk, nt)
if (nflds .ne. 0) then
write(*,*) 'Data fields (UTMGrid): ', fieldlist
do i = 1, nflds
write(*,*) 'rank type: ', rk(i), nt(i)
enddo
endif

nflds = he5_gdingflds(gdid2, fieldlist, rk, nt)
if (nflds .ne. 0) then
write(*,*) 'Data fields (PolarGrid): ', fieldlist
do i = 1, nflds
write(*,*) 'rank type: ', rk(i), nt(i)
enddo
endif

status = he5_gdfldinfo(gdid2, 'Spectra', rk, dims, nt,
dimname, mxdimname)
write(*,*) 'Spectra rank dims: ', rk(1)
write(*,*) 'Spectra dim names: ', dimname
write(*,*) 'Spectra max. dim names: ', mxdimname
do i = 1, rk(1)
write(*,*) 'Spectra dims: ', i, dims(i)
enddo

Get number of grid dimensions and dim. list length

n = he5_gdnentries(gdid1, HE5_HDFE_NENTDIM, strbufsize)
write(*,*) 'Number of dimension entries (UTMGrid): ', n
write(*,*) 'Length of Dimension List (UTMGrid): ', strbufsize

Get number of data fields and field list length

n = he5_gdnentries(gdid1, HE5_HDFE_NENTDFLD, strbufsize)
write(*,*) 'Number of data fields (UTMGrid): ', n
write(*,*) 'Length of Field List (UTMGrid): ', strbufsize
c    Detach from the grids
    -------------------------------------
    status = he5_gddetach(gdid1)
    status = he5_gddetach(gdid2)

c    Close the file
    -------------------------------------
    status = he5_gdclose(gdfid)

stop
end

7.4 Zonal Average Examples

This section contains several examples of the use of the Zonal Average interface from both C and FORTRAN programs. First, there are simple examples in C and FORTRAN which demonstrate the use of most of the functions in the Zonal Average interface.

7.4.1 Creating a Simple Zonal Average

The following C and FORTRAN programs each create, define, and write a simple Zonal Average data set to an HDF-EOS file using the HDF-EOS Zonal Average interface.

7.4.1.1 A C Example of a Simple Zonal Average Creation

Example 1

/* he5_za_setup */
#include <HE5_HdfEosDef.h>

/* In this program we (1) open an HDF-EOS file, (2) create the ZA */
/* interface within the file, and (3) define the za field dimensions */
/* ----------------------------------------------------------------- */

int main()
{
    herr_t status = FAIL;
    hid_t zafid = FAIL;
    hid_t ZAid = FAIL;

    /* Open a new HDF-EOS za file, "ZA.he5" */
    /* ---------------------------------------- */
    zafid = HE5_ZAopen("ZA.he5", H5F_ACC_TRUNC);
    printf("File ID returned by HE5_ZAopen(): %d \n", zafid);

    /* Create the ZA, "ZA1", within the file */
    /* --------------------------------------- */
    ZAid = HE5_ZAcreate(zafid, "ZA1");
    printf("ZA ID returned by HE5_ZAcreate(): %d \n", ZAid);
Example 2

/* Define dimensions and specify their sizes */
/* ----------------------------------------- */
status = HE5_ZAdefdim(ZAid, "MyTrack1", 20);
printf("Status returned by HE5_ZAdefdim(): \%d \n", status);
status = HE5_ZAdefdim(ZAid, "MyTrack2", 10);
printf("Status returned by HE5_ZAdefdim(): \%d \n", status);
status = HE5_ZAdefdim(ZAid, "Res2tr", 40);
printf("Status returned by HE5_ZAdefdim(): \%d \n", status);
status = HE5_ZAdefdim(ZAid, "Res2xtr", 20);
printf("Status returned by HE5_ZAdefdim(): \%d \n", status);
status = HE5_ZAdefdim(ZAid, "Res2tr", 10);
printf("Status returned by HE5_ZAdefdim(): \%d \n", status);
status = HE5_ZAdefdim(ZAid, "Res2xtr", 20);
printf("Status returned by HE5_ZAdefdim(): \%d \n", status);
status = HE5_ZAdefdim(ZAid, "IndxTrack", 12);
printf("Status returned by HE5_ZAdefdim(): \%d \n", status);
status = HE5_ZAdefdim(ZAid, "ExtDim", 60);
printf("Status returned by HE5_ZAdefdim(): \%d \n", status);
status = HE5_ZAdefdim(ZAid, "Unlim", H5S_UNLIMITED);
printf("Status returned by HE5_ZAdefdim(): \%d \n", status);

/* Close the za interface */
/* ---------------------- */
status = HE5_ZAdetach(ZAid);
printf("Status returned by HE5_ZAdetach(): \%d \n", status);
status = HE5_ZAclose(ZAid);
printf("Status returned by HE5_ZAclose(): \%d \n", status);

return 0;

Example 2

/* he5_za_definefields */
#include <HE5_HdfEosDef.h>
#define RANK 3
#define rank 1

/* In this program we (1) open the "ZA.he5" HDF-EOS file, */
/* (2) attach to the "ZA1" za, and (3) define the fields */
/* ----------------------------------------------- */
int main()


```
{
    herr_t     status = FAIL;
    int        comp_level[ 5 ] = {0,0,0,0,0};
    int        comp_code;
    hid_t      zafid = FAIL;
    hid_t      ZAid  = FAIL;
    hsize_t    chunk_dims[ 3 ];
/* Open the file, "ZA.he5", using the H5F_ACC_RDWR access code */
/* -------------------------- */
zafid = HE5_ZAopen("ZA.he5", H5F_ACC_RDWR);
if (zafid != FAIL)
{
    ZAid = HE5_ZAattach(zafid, "ZA1");
    if (ZAid != FAIL)
    {
        status = HE5_ZAdefine(ZAid, "Density", "MyTrack1", NULL,
H5T_NATIVE_FLOAT);
        printf("Status returned by HE5_ZAdefine(...\"Density\",... ) :
%d\n",status);
        status = HE5_ZAdefine(ZAid, "Temperature", "MyTrack1,MyTrack2",NULL,
H5T_NATIVE_FLOAT);
        printf("Status returned by HE5_ZAdefine(...\"Temperature\",... ) :
%d\n",status);
        status = HE5_ZAdefine(ZAid, "Pressure", "Res2tr,Res2xtr", NULL,
H5T_NATIVE_DOUBLE);
        printf("Status returned by HE5_ZAdefine(...\"Pressure\",... ) :
%d\n",status);
    } /* Second, define compression scheme */
    /* -------------------------- */
    /* set the value of compression code: */
    /* HDFE_COMP_NONE              0 */
    /* HDFE_COMP_RLE               1 */
    /* HDFE_COMP_NBIT              2 */
    /* HDFE_COMP_SKPHUFF           3 */
    /* HDFE_COMP_DEFLATE           4 */
    comp_code = 4;
    /*comp_code = 0;*/
```
/* Set compression level: value 0,1,2,3,4,5,6,7,8, or 9 */
/* ----------------------------------------------- */
comp_level[0] = 6;
/*comp_level[0] = 0;/*

status = HE5_ZAdefcomp(ZAid,comp_code, comp_level);
printf("\tStatus returned by HE5_ZAdefcomp() : %d\n",status);
status = HE5_ZAdefine(ZAid, "Spectra", "Bands,Res2tr,Res2xtr", NULL, H5T_NATIVE_FLOAT);
printf("Status returned by HE5_ZAdefine(...) : %d\n",status);

/* Define Appendable Field */
/* ------------------------ */

/* First, define chunking */
/* (the appendable dataset must be chunked) */
/* -------------------------------------------- */
chunk_dims[0] = 20;
status = HE5_ZAdefchunk(ZAid, rank, chunk_dims);
printf("\tStatus returned by HE5_ZAdefchunk() : %d\n",status);
/* Second, define compression scheme */
/* --------------------------------- */

/* set the value of compression code: */
/* HDFE_COMP_NONE                  0 */
/* HDFE_COMP_RLE                   1 */
/* HDFE_COMP_NBIT                  2 */
/* HDFE_COMP.SKPHUFF               3 */
/* HDFE_COMP.DEFLATE               4 */
comp_code = 4;
/*comp_code = 0;/*

/* Set compression level: value 0,1,2,3,4,5,6,7,8, or 9 */
/* ----------------------------------------------- */
comp_level[0] = 6;
/*comp_level[0] = 0;/*

status = HE5_ZAdefcomp(ZAid,comp_code, comp_level);
printf("\tStatus returned by HE5_ZAdefcomp() : %d\n",status);
status = HE5_ZAdefine(ZAid, "Count", "MyTrack1", "Unlim", H5T_NATIVE_INT);
printf("Status returned by HE5_ZAdefine(...) : %d\n",status);

} }

status = HE5_ZAdetach(ZAid);
status = HE5_ZAclose(zafid);
return 0;
Example 3

/* he5_za_writedata */

#include <HE5_HdfEosDef.h>

/* In this program we (1) open the "ZA.he5" file, (2) attach to the      */
/* "ZA1" za, and (3) write data to the "Spectra" fields. Also, set up */
/* the global, group, and local attributes                           */

int main()
{
    herr_t status = FAIL;
    int i, j, k;
    int attr1[4] = {1, 2, 3, 4};    /* global attribute */
    int attr2[4] = {10, 20, 30, 40}; /* group attribute */
    int attr3[4] = {100, 200, 300, 400}; /* local attribute */
    hid_t zafid = FAIL;
    hid_t ZAid = FAIL;
    char attr4[7]; /* Global 'char' attribute */
    long attr5[4] = {1111111L,2222222L,3333333L,4444444L}; /* Global 'long' attribute */
    hssize_t start[3];
    hsize_t count[3];
    double plane[15][40][20];

    /* Populate spectra data array. Value = 100*(track index)+(band index) */
    /* -------------------------------------------- */
    for (i = 0; i < 15; i++)
    {
        for (j = 0; j < 40; j++)
            for (k = 0; k < 20; k++)
                plane[i][j][k] = (double)(j*100 + i);
    }

    /* Open the HDF za file, "ZA.he5" */
    /* -------------------------------- */
    zafid = HE5_ZAopen("ZA.he5", H5F_ACC_RDWR);
    if (zafid != FAIL)
    {
        /* Attach the "ZA1" za */
        /* ------------------ */
        ZAid = HE5_ZAattach(zafid, "ZA1");
        if (ZAid != FAIL)
        {
/* Write Spectra Field */
/* ------------------- */
start[0] = 0;  count[0] = 15;
start[1] = 0;  count[1] = 40;

status = HE5_ZAwrite(ZAid, "Spectra", start, NULL, count, plane);
printf("status returned by HE5_ZAwrite("Spectra"):
%d\n", status);

/* Write Global 'int' Attribute */
/* ----------------------------- */
count[0] = 4;
status = HE5_ZAwriteattr(ZAid, "GlobalAttribute", H5T_NATIVE_INT,
count, attr1);
printf("status returned by HE5_ZAwriteattr("GlobalAttribute"):
%d\n", status);

/* Write Global 'char' Attribute */
/* ------------------------------ */
strcpy(attr4,"ABCDEF");
count[0] = 6;
status = HE5_ZAwriteattr(ZAid, "GLOBAL_CHAR_ATTR", H5T_NATIVE_CHAR,
count, attr4);
printf("status returned by HE5_ZAwriteattr("GLOBAL_CHAR_ATTR"):
%d\n", status);

/* Write Global 'long' Attribute */
/* ------------------------------- */
count[0] = 4;
status = HE5_ZAwriteattr(ZAid, "GLOBAL_LONG_ATTR", H5T_NATIVE_LONG,
count, attr5);
printf("status returned by HE5_ZAwriteattr("GLOBAL_LONG_ATTR"):
%d\n", status);

/* Write Global 'double' Attribute */
/* ---------------------------------- */
count[0] = 4;
status = HE5_ZAwriteattr(ZAid, "GLOBAL_DOUBLE_ATTR", H5T_NATIVE_DOUBLE,
count, attr6);
printf("status returned by HE5_ZAwriteattr("GLOBAL_DOUBLE_ATTR"):
%d\n", status);

/* Write Group Attribute */
/* ----------------------- */
status = HE5_ZAwritegrpattr(ZAid, "GroupAttribute", H5T_NATIVE_INT,
count, attr2);
printf("status returned by HE5_ZAwritegrpattr("GroupAttribute"):
%d\n", status);

/* Write Local Attribute */
/* ---------------------- */
status = HE5_ZAwriteilocattr(ZAid, "Density", "LocalAttribute",
H5T_NATIVE_INT, count, attr3);
printf("status returned by HE5_ZAwriteilocattr("LocalAttribute"):
%d\n", status);}
Example 4

/* he5_za_readdata */

#include <HE5_HdfEosDef.h>

/* In this program we (1) open the "ZA.he5" HDF-EOS file, (2) attach to */
/* the "ZA1" za, and (3) read data from the "Spectra" field. Also, we */
/* read the global/group/local attributes */

int main()
{
    herr_t          status = FAIL;
    int             i, j, k;
    int             attr1[4];  /* data buffer for global attribute */
    int             attr2[4];  /* .... for group attribute */
    int             attr3[4];  /* .... for local attribute */

    hid_t           zafid = FAIL;
    hid_t           ZAid  = FAIL;

    char            attr4[7];  /* ... for global 'char' attribute */
    long            attr5[4];  /* ... for global 'long' attribute */
    double          attr6[4];  /* ... for global 'double' attribute */

    hssize_t        start[3];
    hsize_t         count[3];

    double          plane[15][40][20];

    /* Populate spectra data array. Value = 100*(track index)+(band index) */
    /* ----------------------------------------------- */
    for (i = 0; i < 15; i++)
    {
        for (j = 0; j < 40; j++)
            for (k = 0; k < 20; k++)
                plane[i][j][k] = (double)(j*100 + i);
    }

    /* Open the HDF-EOS za file, "ZA.he5" */
    /* ----------------------------------- */
    zafid = HE5_ZAopen("ZA.he5", H5F_ACC_RDONLY);
    if (zafid != FAIL)
    {
        /* Attach the "ZA1" za */
        /* ------------------- */
        ZAid = HE5_ZAattach(zafid, "ZA1");
    }
}
if (ZAid != FAIL)
{
    /* Read the entire Spectra field */
    /* ----------------------------------- */
    start[0] = 0;   start[1] = 0;   start[2] = 0;
    status = HE5_ZAread(ZAid, "Spectra", start, NULL, count, plane);
    printf("Status returned by HE5_ZAread() : %d \n", status);

    /* Read Global 'int' Attribute */
    /* -------------------------------- */
    status = HE5_ZAreadattr(ZAid, "GlobalAttribute", attr1);
    printf("Status returned by HE5_ZAreadattr() : %d \n", status);
    printf("Global attribute values:\n");
    for (i = 0; i < 4; i++)
        printf("%d \n", attr1[i]);

    /* Read Group Attribute */
    /* ----------------------- */
    status = HE5_ZAreadgrpattr(ZAid, "GroupAttribute", attr2);
    printf("Status returned by HE5_ZAreadgrpattr() : %d \n", status);
    printf("Group attribute values:\n");
    for (i = 0; i < 4; i++)
        printf("%d \n", attr2[i]);

    /* Read Local Attribute */
    /* ---------------------- */
    status = HE5_ZAreadlocattr(ZAid, "Density", "LocalAttribute", attr3);
    printf("Status returned by HE5_ZAreadlocattr() : %d \n", status);
    printf("Local attribute values:\n");
    for (i = 0; i < 4; i++)
        printf("%d \n", attr3[i]);

    /* Read Global 'char' Attribute */
    /* -------------------------------- */
    status = HE5_ZAreadattr(ZAid, "GLOBAL_CHAR_ATTR", attr4);
    printf("Status returned by HE5_ZAreadattr() : %d \n", status);
    printf("\t%\n", attr4);

    /* Read Global 'long' Attribute */
    /* -------------------------------- */
    status = HE5_ZAreadattr(ZAid, "GLOBAL_LONG_ATTR", attr5);
    printf("Status returned by HE5_ZAreadattr() : %d \n", status);
    printf("\t%li \n", attr5);

    /* Read Global 'double' Attribute */
    /* -------------------------------- */
    status = HE5_ZAreadattr(ZAid, "GLOBAL_DOUBLE_ATTR", attr6);
    printf("Status returned by HE5_ZAreadattr() : %d \n", status);
    printf("\t%f \n", attr6);
}
Example 5

/* he5_za_info */

#include <HE5_HdfEosDef.h>

/* In this program we retrieve information about (1) dimensions, (2) */
/* za fields, and (3) the global/group/local attributes */

int main()
{
    herr_t status = FAIL;
    int i, *rank;
    hid_t zafid = FAIL, ZAid = FAIL;
    hid_t ntype[10];
    hid_t dtype = FAIL;
    long ndims, strbufsize, nflds, nattr;
    hsize_t dimsize;
    hsize_t *dims;
    hsize_t n, nelem = 0;
    char version[80] = {0};
    char *dimname, *fieldlist;
    char attrlist[80];

    /* Open the ZA HDF-EOS File "ZA.he5" for reading only */
    zafid = HE5_ZAopen("ZA.he5", H5F_ACC_RDONLY);
    if (zafid != FAIL)
    {
        HE5_EHgetversion(zafid, version);
        printf("HDF-EOS library version: \"%s\n", version);

        /* Attach the ZA "ZA1" */
        ZAid = HE5_ZAattach(zafid, "ZA1");
        if (ZAid != FAIL)
        {
            /* Inquire Dimensions */
            ndims = HE5_ZAnentries(ZAid, HE5_HDFE_NENTDIM, &strbufsize);
            dims = (hsize_t *) calloc(ndims, sizeof(hsize_t));
            dimname = (char *) calloc(strbufsize + 1, 1);
ndims = HE5_ZAinqdims(ZAid, dimname, dims);

printf("Dimension list: \%s\n", dimname);
for (i = 0; i < ndims; i++)
    printf("dim size: \%li \n", (long)dims[i]);
free(dims);
free(dimname);

/* Inquire Data Fields */
/* ------------------- */
fld = HE5_ZAentries(ZAid, HE5_HDFE_NENTDFLD, &strbufsize);
rank = (int *) calloc(nflds, sizeof(int));
fieldlist = (char *) calloc(strbufsize + 1, 1);
fld = HE5_ZAinquire(ZAid, fieldlist, rank, ntype);
printf("data fields: \%s\n", fieldlist);
for (i = 0; i < fld; i++)
    printf("Rank: \%d Data type: \%d\n", rank[i], ntype[i]);
free(rank);
free(fieldlist);

/* Get info on "MyTrack1" dim */
/* -------------------------- */
dimsize = HE5_ZAinfo(ZAid, "MyTrack1");
printf("Size of MyTrack1: \%lu\n", (unsigned long)dimsize);

dtype = FAIL;
/* Get info about Global Attributes */
/* -------------------------------- */
printf("Global Attribute:\n");
status = HE5_ZAattrinfo(ZAid, "GlobalAttribute", &dtype, &nelem);
printf("\t\tData type: \%d\n", dtype);
printf("\t\tNumber of elements: \%lu\n", (unsigned long)nelem);
nelem = 0;
dtype = FAIL;
/* Get info about Group Attributes */
/* ------------------------------- */
printf("Group Attribute:\n");
status = HE5_ZAgroupattrinfo(ZAid, "GroupAttribute", &dtype, &nelem);
printf("\t\tData type: \%d\n", dtype);
printf("\t\tNumber of elements: \%lu\n", (unsigned long)nelem);
nelem = 777;
dtype = FAIL;
/* Get info about Local Attributes */
/* ------------------------------- */
printf("Local Attribute:\n");
status = HE5_ZAlocalattrinfo(ZAid, "LocalAttribute", &dtype, &nelem);
printf("\t\tData type: \%d\n", dtype);
printf("\t\tNumber of elements: \%lu\n", (unsigned long)nelem);

/* Inquire Global Attributes */
/* -------------------------- */
printf("Global Attributes:\n");
nattr = HE5_ZAinqattrs(ZAid, NULL, &strbufsize);
printf("\t\t\tNumber of attributes: %li \n", nattr);
printf("\t\t\tString length of attribute list: %li \n", strbufsize);
n = HE5_ZAinqattrs(ZAid, attrlist, &strbufsize);
printf("\t\t\tAttribute list: %s \n", attrlist);

/* Inquire Group Attributes */
/* ------------------------ */
strbufsize = 0;
printf("\n");
printf("Group Attributes:\n");
nattr = HE5_ZAinqgrppattrs(ZAid, NULL, &strbufsize);
printf("\t\t\tNumber of attributes: %li \n", nattr);
printf("\t\t\tString length of attribute list: %li \n", strbufsize);
strcpy(attrlist,"");
nattr = HE5_ZAinqgrppattrs(ZAid, attrlist, &strbufsize);
printf("\t\t\tAttribute list: %s \n", attrlist);

/* Inquire Local Attributes */
/* ------------------------ */
strbufsize = 0;
printf("\n");
printf("Local Attributes:\n");
nattr = HE5_ZAinqlocattrs(ZAid, "Density", NULL, &strbufsize);
printf("\t\t\tNumber of attributes: %li \n", nattr);
printf("\t\t\tString length of attribute list: %li \n", strbufsize);
strcpy(attrlist,"");
nattr = HE5_ZAinqlocattrs(ZAid, "Density", attrlist, &strbufsize);
printf("\t\t\tAttribute list: %s \n", attrlist);

Example 6

/* he5_za_datainfo */

#include <HE5_HdfEosDef.h>
#define FILENAME   "ZA.he5"
#define OBJECT     "ZA1"

int main(void)
{
    herr_t status = FAIL;
    int fieldgroup = FAIL;
    hid_t fid = FAIL;
    hid_t ZAid = FAIL;
    hid_t datatype = FAIL;
    H5T_class_t classid = H5T_NO_CLASS;

    status = HE5_ZAdetach(ZAid);
    status = HE5_ZAclose(zafid);

    return 0;
}
H5T_order_t order = H5T_ORDER_ERROR;
size_t size = 0;

/* Open the HDF-EOS ZA file */
fid = HE5_ZAopen(FILENAME, H5F_ACC_RDONLY);
printf("File ID returned by HE5_ZAopen() : %d \n", fid);

/* Attach to the "ZA1" za */
ZAid = HE5_ZAattach(fid, OBJECT);
printf("ZA ID returned by HE5_ZAattach() : %d \n", ZAid);

/* Inquire data type information for the "Spectra" field */
fieldgroup = HE5_HDFE_DATAGROUP;
status = HE5_ZAinqdatatype(ZAid, "Spectra", NULL, fieldgroup, &datatype, &classid, &order, &size);
printf("Status returned by HE5_ZAinqdatatype() : %d \n", status);
if (status != FAIL)
{
    printf("tdatatype: %d \n", datatype);
    printf("tclass ID: %d \n", classid);
    printf("torder: %d \n", order);
    printf("tsize: %d \n", (int)size);
}

/* Inquire data type information for the attributes */
fieldgroup = HE5_HDFE_ATTRGROUP;
status = HE5_ZAinqdatatype(ZAid, NULL, "GlobalAttribute", fieldgroup, &datatype, &classid, &order, &size);
printf("Status returned by HE5_ZAinqdatatype() : %d \n", status);
if (status != FAIL)
{
    printf("tdatatype: %d \n", datatype);
    printf("tclass ID: %d \n", classid);
    printf("torder: %d \n", order);
    printf("tsize: %d \n", (int)size);
}

fieldgroup = HE5_HDFE_GRPATTRGROUP;
status = HE5_ZAinqdatatype(ZAid, NULL, "GroupAttribute", fieldgroup, &datatype, &classid, &order, &size);
printf("Status returned by HE5_ZAinqdatatype() : %d \n", status);
if (status != FAIL)
{
    printf("tdatatype: %d \n", datatype);
    printf("tclass ID: %d \n", classid);
    printf("torder: %d \n", order);
    printf("tsize: %d \n", (int)size);
}

fieldgroup = HE5_HDFE_LOCATTRGROUP;
status = HE5_ZAinqdatatype(ZAid, "Density", "LocalAttribute", fieldgroup, &datatype, &classid, &order, &size);
printf("Status returned by HE5_ZAinqdatatype() : %d \n", status);
if (status != FAIL)
{
    printf("tdatatype: %d \n", datatype);
    printf("tclass ID: %d \n", classid);
    printf("torder: %d \n", order);
    printf("tsize: %d \n", (int)size);
}
7.4.1.2 A FORTRAN Example of a Simple Zonal Average Creation

Example 1

In this program we (1) open an HDF-EOS file, (2) create the za interface, and (3) define the za field dimensions

---

```fortran
program he5_zasampleF_32
implicit none
include 'hdfeos5.inc'
integer status
integer he5_zaopen
integer he5_zacreate
integer he5_zadefdim
integer he5_zadetach
integer he5_zaclose
integer zafid, zaid
integer*4 dtrack, extdata

zafid = he5_zaopen('za.he5',HE5F_ACC_TRUNC)
write(*,*) 'File ID returned by he5_zaopen(): ',zafid

zaid = he5_zacreate(zafid, "ZA1")
write(*,*) 'ZA ID returned by he5_zacreate(): ',zaid

define Data dimensions
```
Example 2

In this program we (1) open the "za.he5" HDF-EOS file, (2) attach to the "ZA1" za, and (3) define the za fields.

```
program he5_zadefieldsF_32
implicit none
!
!! Define "External" dimension
!!
dtrack = 20
status = he5_zadefdim(zaid, "MyTrack1", dtrack)
write(*,*) 'Status returned by he5_zadefdim(): ',status
!
dtrack = 10
status = he5_zadefdim(zaid, "MyTrack2", dtrack)
write(*,*) 'Status returned by he5_zadefdim(): ',status
!
dtrack = 40
status = he5_zadefdim(zaid, "Res2tr", dtrack)
write(*,*) 'Status returned by he5_zadefdim(): ',status
!
dtrack = 20
status = he5_zadefdim(zaid, "Res2xtr", dtrack)
write(*,*) 'Status returned by he5_zadefdim(): ',status
!
dtrack = 15
status = he5_zadefdim(zaid, "Bands", dtrack)
write(*,*) 'Status returned by he5_zadefdim(): ',status
!
dtrack = 12
status = he5_zadefdim(zaid, "IndxTrack", dtrack)
write(*,*) 'Status returned by he5_zadefdim(): ',status
!
Define "External" dimension
!
eextdata = 60
status = he5_zadefdim(zaid, "ExtDim", exdata)
write(*,*) 'Status returned by he5_zadefdim(): ',status
!
Define Unlimited (appendable) dimension
!
status = he5_zadefdim(zaid, "Unlim", HE5S_UNLIMITED_F)
write(*,*) 'Status returned by he5_zadefdim(): ',status
!
Detach from the za
!
status = he5_zadetach(zaid)
write(*,*) 'Status returned by he5_zadetach(): ',status
!
Close the za file
!
status = he5_zaclose(zafid)
write(*,*) 'Status returned by he5_zaclose(): ',status
stop
end
```
include     'hdfeos5.inc'

integer     status
integer     he5_zaoopen
integer     he5_zaattract
integer     he5_zadefine
integer     he5_zadetach
integer     he5_zaclose
integer     FAIL, zaid

define     FAIL
parameter   (FAIL=-1)

! Open the HDF-EOS file, "za.he5" using "READ/WRITE" access code
!--------------------------------------------------------------

zafid = he5_zaoopen("za.he5",HE5F_ACC_RDWR)
write(*,*) 'File ID returned by he5_zaoopen():  ',zafid
if (zafid .NE. FAIL) then
  zaid = he5_zaattract(zafid, "ZA1")
  write(*,*) 'ZA ID  returned by he5_zaattract():  ',zaid
  if (zaid .NE. FAIL) then
    ! Define Data fields
    !------------------
    status = he5_zadefine(zaid, "Density",
      "MyTrack1",",", HE5T_NATIVE_FLOAT)
    write(*,*) 'Status returned by he5_zadefine():  ',status
    status = he5_zadefine(zaid, "Temperature",
      "MyTrack2,MyTrack1",
      2           "", HE5T_NATIVE_FLOAT)
    write(*,*) 'Status returned by he5_zadefine():  ',status
    status = he5_zadefine(zaid, "Pressure",
      "Res2xtr,Res2tr",
      2           "", HE5T_NATIVE_FLOAT)
    write(*,*) 'Status returned by he5_zadefine():  ',status
    status = he5_zadefine(zaid, "Spectra",
      "Res2xtr,Res2tr,Bands",
      2           "", HE5T_NATIVE_DOUBLE)
    write(*,*) 'Status returned by he5_zadefine():  ',status
  endif
endif

! Detach from the za
!------------------
status = he5_zadetach(zaid)
write(*,*) 'Status returned by he5_zadetach():  ',status

! Close the file
!--------------
status = he5_zaclose(zafid)
write(*,*) 'Status returned by he5_zaclose():  ',status
stop
Example 3

In this program we (1) open the "za.he5" file, (2) attach to the "ZA1" za, and (3) write data to the "Spectra" fields.

```fortran
program he5_za_writedataF_32
implicit none
include 'hdfeos5.inc'
integer status
integer he5_zaopen
integer he5_zaattach
integer he5_zawrite
integer he5_zawrattr
integer he5_zadetach
integer he5_zaclose
integer zafid, zaid
integer i, j, k
integer*4 attr(4)
integer*4 start(3)
integer*4 stride(3)
integer*4 count(3)
real*8 plane(800)
integer FAIL
parameter (FAIL=-1)

! Open HDF-EOS file, "za.he5"
! ---------------------------
zafid = he5_zaopen("za.he5", HE5F_ACC_RDWR)
write(*,*) 'File ID returned by he5_zaopen(): ', zafid
if (zafid .NE. FAIL) then
    zaid = he5_zaattach(zafid, "ZA1")
    write(*,*) 'ZA ID returned by he5_zaattach(): ', zaid
endif
if (zaid .NE. FAIL) then

! Write Spectra one plane at a time
! Value is 100 * track index + band index (0-based)
! -----------------------------------------------
    start(1) = 0
    start(2) = 0
    count(1) = 20
    count(2) = 40
    count(3) = 1
    stride(1) = 1
    stride(2) = 1
    stride(3) = 1
    do i=1,15
```

start(3) = i - 1
do j=1,40
  do k=1,20
    plane((j-1)*20+k) = (j-1)*100 + i-1
  enddo
enddo
status = he5_zawrite(zaid,"Spectra",start,
  stride,count,plane)
write(*,*) 'Status returned by he5_zawrite(): ',status

Write User defined Attribute

attr(1) = 3
attr(2) = 5
attr(3) = 7
attr(4) = 11
count(1) = 4
status = he5_zawrattr(zaid,"TestAttr",HE5T_NATIVE_INT,
  count,attr)
endif
endif

Detach from the za
status = he5_zadetach(zaid)
write(*,*) 'Status returned by he5_zadetach(): ',status

Close the file
status = he5_zaclose(zafid)
write(*,*) 'Status returned by he5_zaclose(): ',status
stop
end
Example 4

In this program we (1) open the "za.he5" file, (2) attach to the "ZA1" za, and (3) read data from the "Spectra" field

```fortran
program he5_za_readdataF_32
implicit none
include 'hdfeos5.inc'
integer status
integer he5_zaopen
integer he5_zaattract
integer he5_zaread
integer he5_zardattr
integer he5_zadetach
integer he5_zaclose
integer zafid, zaid
integer i,j,k
integer*4 attr(4)
integer*4 start(3)
integer*4 stride(3)
integer*4 count(3)
real*8 plane(800)
integer FAIL
parameter (FAIL=-1)

! Open HDF-EOS za file, "za.he5"
! -----------------------------
  zafid = he5_zaopen("za.he5",HE5F_ACC_RDWR)
  write(*,*) 'File ID returned by he5_zaopen(): ',zafid
  if (zafid .NE. FAIL) then
    za
  zaid = he5_zaattract(zafid, "ZA1")
  write(*,*) 'ZA ID returned by he5_zaattract(): ',zaid
  if (zaid .NE. FAIL) then

! Read the entire Spectra field
! -----------------------------
  start(1) = 0
  start(2) = 0
  count(1) = 20
  count(2) = 40
  count(3) = 1
  stride(1) = 1
  stride(2) = 1
  stride(3) = 1
  do i=1,15
    start(3) = i - 1
    do j=1,40
      do k=1,20
```
Example 5

! In this program we retrieve (1) information about the dimensions, (2) the za fields
!
program he5_za_infoF_32
implicit none
include 'hdfeos5.inc'
integer i
integer status
integer zafid, zaid
integer he5_zaopen
integer he5_zaattach
integer he5_zainfo
integer he5_zadetach
integer he5_zaclose
integer rank(32)
integer ntype(32)
integer rk
integer nt
integer*4 he5_zainqdims
integer*4 he5_zainquire
integer*4 he5_zadiminfo
integer*4 ndims
integer*4 nflds
integer*4 dims(32)
integer*4       dimsize
character*72    dimname
character*72    dimlist
character*72    maxdimlst
character*72    fieldlist
integer         FAIL
parameter       (FAIL=-1)

Open the "za.he5" file for "read only" access
---------------------------------------------
zafid = he5_zaoopen("za.he5", HE5F_ACC_RDONLY)
write(*,*) 'File ID returned by he5_zaoopen(): ',zafid
if (zafid .NE. FAIL) then

Attach the za
---------------------
zaid = he5_zaattach(zafid, "ZA1")
write(*,*) 'ZA ID returned by he5_zaattach(): ',zaid
if (zaid .NE. FAIL) then

Inquire Dimensions
--------------------
ndims = he5_zainqdims(zaid, dimname, dims)
write(*,*) 'Dimension list: ', dimname
do i = 1, ndims
    write(*,*) 'dim size: ', dims(i)
enddo
write(*,*)

Inquire Data Fields
-------------------
 nflds = he5_zainquire(zaid, fieldlist, rank, ntype)
write(*,*) 'Data Fieldlist: ', fieldlist
do i=1, nflds
    write(*,*) 'field rank & datatype: ', rank(i), ntype(i)
enddo
write(*,*)

Get info on "MyTrack1" dim
--------------------------
dimsize = he5_zadiminfo(zaid,"MyTrack1")
write(*,*) 'Size of MyTrack1: ', dimsize
write(*,*)

Get info on "Spectra" Field
---------------------------
 status = he5_zainfo(zaid,"Spectra",rk,dims,nt,
dimlist,maxdimlst)
  write(*,*) 'Spectra Rank: ', rk
  write(*,*) 'Spectra NumberType: ', nt
  write(*,*) 'Spectra Dimlist: ', dimlist
  write(*,*) 'Spectra Max Dimlist: ', maxdimlst
  do i=1,rk
    write(*,*) 'Dimension ',i,dims(i)
  enddo
endif
endif

! Detach from the za
-------------------
  status = he5_zadetach(zaid)
write(*,*) 'Status returned by he5_zadetach(): ',status

! Close the file
! ---------------
  status = he5_zaclose(zafid)
write(*,*) 'Status returned by he5_zaclose(): ',status

stop
end
8. Writing ODL Metadata into HDF-EOS

8.1 Introduction

The following C code fragments are examples of how a user can write ECS granule metadata (or inventory metadata) into their HDF-EOS file. The codes use a template, called Metadata Configuration File (MCF), which is used to determine what metadata attributes to write into the HDF-EOS file. The output file is written in Object Description Language (ODL). This file is written into the HDF-EOS file as a global attribute object. Part of this file will contain ECS core attributes, which will also be written into ECS databases. The Metadata Configuration File (MCF), which the code accesses is given in Section 8.3. The output ODL file which results from running the codes in 8.2.1 or 8.2.2 is given in Section 8.4.

It should be mentioned that currently both MTD TOOLKIT (a subset of SDP TOOLKIT that handles metadata and Time/Date conversions) and SDP TOOLKIT have been modified to write metadata into HDF-EOS files utilizing HDF5 as well as HDF4. Details on Metadata Configuration Files can be found in the SDP Toolkit Users Guide for EOSDIS Evolution and Development Contract, Section 6.2.1 and Appendix J. Details on how to install and use MTD TOOLKIT tools or SDP TOOLKIT can be found in Toolkit_MTD Users Guide [A Data Formatting Toolkit for Extended Data Providers to NASA’s Earth Observing System Data and Information System (V5.0)] or SDP TOOLKIT Users Guide version 5.2.18 respectively.

Note that the MTD toolkit and SDP toolkit use different process control files. These are configuration files that specify relationships between physical and logical file handles. These, two file templates called filetable.temp and PCF file, used by the MTD and SDP toolkits respectively, are provided in Sections 8.5.1 and 8.5.2. The file filetable.temp, which is used only by MTD TOOLKIT, is similar to the Process Control File, i.e. the PCF file, used only by the larger SDP Toolkit, but simpler. Both files are used to specify the relationship between logical file identifiers used in source code and physical files containing input data or output data. They are also used to specify the IDs for log status reports and for the MCFs.

Please also note that both MTD and SDP toolkits must be installed with both HDF4 and HDF5. If product hdf file that metadata is going to be written to is HDF4 based, user must call PGS_MET_SDstart in the code examples of Section 8.2.1 or 8.2.2 with HDF4_ACC_RDWR access flag. Otherwise, if the product hdf file is HDF5 based user must call PGS_MET_SDstart with HDF5_ACC_RDWR access flag. If the hdf file that metadata is going to be written to does not exist, user must use HDF4_ACC_CREATE or HDF5_ACC_CREATE as access flags for HDF4 based or HDF5 based hdf files respectively.
8.2 Coding Examples of Metadata Writes to HDF-EOS Files

8.2.1 C Code for MTD TOOLKIT Users

/* include files */

#include <PGS_MET.h>
#include <PGS_tk.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hdf5.h>
#include <PGS_SMF.h>

#define INVENTORYMETADATA 1
#define ARCHIVEDMETADATA 2
#define ODL_IN_MEMORY 0

extern PGSt_SMF_status
PGS_PC_GetReference(PGSt_MET_Logical prodID, PGSt_integer *version, char *referenceID);

int main()
{

/*******************************
Declarations.
*******************************/
PGSt_MET_all_handles mdHandles;
PGSt_MET_all_handles handles;
char fileName1[PGSt_MET_FILE_PATH_MAX]="";
char fileName2[PGSt_MET_FILE_PATH_MAX]="";
char my_HDF_file[PGSt_MET_FILE_PATH_MAX]="";
char msg[PGSt_SMF_MAX_MSG_SIZE];
char mnemonic[PGSt_SMF_MAX_MNEMONIC_SIZE];
char fileMessage[PGSt_SMF_MAX_MSG_SIZE];
int32 sdid1;
PGSt_SMF_status ret = PGS_S_SUCCESS;
char *informationName;
PGSt_integer ival =3;
PGSt_double dval=203.2;
PGSt_integer fileId, fileId2;
PGSt_integer i;
PGSt_integer version;
PGSt_SMF_status returnStatus;
char *mysaval[5];

/*******************************
/* Associate logical IDs with physical filenames. */
*******************************
ret=PGS_MET_SetFileId();
printf("ret after PGS_MET_SetFileId() is %d in Main\n",ret);

if(ret != PGS_S_SUCCESS)
{
    printf("Failed in assigning logical IDs\n");
}

/*recovery file name for fileId=PGSd_MET_MCF_FILE */

version = 1;
fileId = PGSd_MET_MCF_FILE;
returnStatus = PGS_PC_GetReference( fileId, &version, fileName1);
if ( returnStatus != PGS_S_SUCCESS )
{
    PGS_SMF_GetMsg( &returnStatus, mnemonic, msg );
    strcpy(fileMessage, msg);
    PGS_SMF_SetDynamicMsg( returnStatus,fileMessage, "metatest" );
}
else
{
    printf("The input file for ID %d is %s\n",fileId,fileName1);
}

informationname=(char *) malloc(330);

/* Initialize MCF file */

fileId = 10250;
ret=PGS_MET_Init(fileId,handles);

if (ret !=PGS_S_SUCCESS)
{
    printf("initialization failed\n");
    return (-1);
}
else
{
    printf("ret after PGS_MET_Init is %d\n",ret);
}

/* test PGS_MET_SetAttr */

ival=667788;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
                   "QAPERCENTINTERPOLATEDDATA.1",&ival);
printf("ret after SetAttr for QAPERCENTINTERPOLATEDDATA.1 is %d\n", ret);

ival=12345;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "QAPercentMissingData.1", &ival);
printf("ret after SetAttr for QAPercentMissingData.1 is %d\n", ret);

ival=123;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "QAPercentOutOfBoundsData.1", &ival);
printf("ret after SetAttr for QAPercentOutOfBoundsData.1 is %d\n", ret);

ival=23456;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "QAPercentOutOfBoundsData.2", &ival);
printf("ret after SetAttr for QAPercentOutOfBoundsData.1 is %d\n", ret);

ival=56789;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "QAPercentMissingData.2", &ival);
printf("ret after SetAttr for QAPercentMissingData.1 is %d\n", ret);

strcpy(informationname,"Exercise1");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "AutomaticQualityFlagExplanation.1", &informationname);
printf("ret after SetAttr for AutomaticQualityFlagExplanation.1 is %d\n", ret);

strcpy(informationname,"1997/12/23");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "RangeBeginningDateTime", &informationname);
printf("ret after SetAttr for RangeBeginningDateTime is %d\n", ret);

strcpy(informationname,"1997.07/30");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "RangeBeginningDate", &informationname);
printf("ret after SetAttr for RangeBeginningDate is %d\n", ret);

strcpy(informationname,"ReprocessingplannINVENT");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ReprocessingPlanned", &informationname);
printf("ret after SetAttr for ReprocessingPlanned is %d\n", ret);

strcpy(informationname,"\"ReprocessingplannARCHIVE\"");
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "ReprocessingPlanned", &informationname);
printf("ret after SetAttr for ReprocessingPlanned is %d\n", ret);

strcpy(informationname,"Reprocessin");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ReprocessingActual", &informationname);
printf("ret after SetAttr for ReprocessingActual is %d\n", ret);

strcpy(informationname,"ID1111");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
   "ReprocessingActual", &informationname);
printf("ret after SetAttr for ReprocessingActual is %d\n", ret);
"LocalGranuleID", &informationname);
printf("ret after SetAttr for LocalGranuleID is %d\n", ret);

strcpy(informationname, "version1234");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "LocalVersionID", &informationname);
printf("ret after SetAttr for LocalVersionID is %d\n", ret);

strcpy(informationname, "Flag1");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "DayNightFlag", &informationname);
printf("ret after SetAttr for DayNightFlag is %d\n", ret);

strcpy(informationname, "Flag1");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "DayNightFlag", &informationname);
printf("ret after SetAttr for DayNightFlag is %d\n", ret);

strcpy(informationname, "information1");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.1", &informationname);
printf("ret after SetAttr for ParameterName is %d\n", ret);

strcpy(informationname, "information2");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.2", &informationname);
printf("ret after SetAttr for ParameterName.2 is %d\n", ret);

strcpy(informationname, "information3");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.3", &informationname);
printf("ret after SetAttr for ParameterName is %d\n", ret);

strcpy(informationname, "information4");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.4", &informationname);
printf("ret after SetAttr for ParameterName is %d\n", ret);

dval = 111.11;
ret = PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "WestBoundingCoordinate", &dval);
printf("ret WestBoundingCoordinate is %d %f\n", ret, dval);

dval = 222.22;
ret = PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "northBoundingCoordinate", &dval);
printf("ret northBoundingCoordinate is %d %f\n", ret, dval);

dval = 333.33;
ret = PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "EastBoundingCoordinate", &dval);
printf("ret EastBoundingCoordinate is %d %f\n", ret, dval);
dval=444.44;
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "SouthBoundingCoordinate", &dval);
printf("ret SouthBoundingCoordinate is \%d \%f\n", ret, dval);

dval=11.11;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "WestBoundingCoordinate", &dval);
printf("ret WestBoundingCoordinate is \%d \%f\n", ret, dval);

dval=22.22;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "northBoundingCoordinate", &dval);
printf("ret northBoundingCoordinate is \%d \%f\n", ret, dval);

dval=33.33;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "EastBoundingCoordinate", &dval);
printf("ret EastBoundingCoordinate is \%d \%f\n", ret, dval);

dval=44.44;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "SouthBoundingCoordinate", &dval);
printf("ret SouthBoundingCoordinate is \%d \%f\n", ret, dval);

/* Get the value of set attribute */

dval=11.11;
ret=PGS_MET_GetSetAttr(handles[INVENTORYMETADATA],
    "SouthBoundingCoordinate", &dval);
printf("after GetSetAttr: ret SouthBoundingCoordinate is \%d \%f\n", ret, dval);

/* Get data from config file */

ret = PGS_MET_GetConfigData("TEST_PARM_FLOAT", &dval);
printf("after PGS_MET_GetConfigData: Ret TEST_PARM_INT is \%d \%f\n", ret, dval);

/* write metadata to HDF and ASCII files */
version = 1;
fileId = 5039;
ret = PGS_PC_GetReference(fileId, &version, my_HDF_file);
printf("after PGS_PC_GetReference ret =\%d\n", ret);
printf("after PGS_PC_GetReference my_HDF_file = %s\n", my_HDF_file);

if (ret == PGS_S_SUCCESS)
{
    ret = PGS_MET_SDstart(my_HDF_file, HDF5_ACC_RDWR, &sdid1);
printf("after PGS MET SDstart sdid1 =\d\n",sdid1);

} else{
    return (-1);
}
printf("After SDstart sdid1 is \d\n",sdid1);

/***** write INVENTORYMETADATA to HDF file *****/  
ret=PGS_MET_Write(handles[INVENTORYMETADATA],"coremetadata",sdid1);  
printf("ret after PGS MET_Write is \d\n",ret);

if(ret !=PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET) {
    if (ret == PGSMET_E_MAND_NOT_SET) {
        printf("some mandatory parameters were not set\n");
    } else{
        printf("HDF write failed\n");
    }
}

/***** write ARCHIVEDMETADATA to HDF file *****/  
ret=PGS_MET_Write(handles[ARCHIVEDMETADATA],"archivemetadata",sdid1);  
printf("ret after PGS MET_Write is \d\n",ret);

if(ret !=PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET) {
    if (ret == PGSMET_E_MAND_NOT_SET) {
        printf("some mandatory parameters were not set\n");
    } else{
        printf("HDF write failed\n");
    }
}

/***** write to non-HDF file *****/  
fileId = 5804;  
printf("non-hdf file to be written has fileId \d\n", fileId);  
ret=PGS_MET_Write(handles[ODL_IN_MEMORY],NULL,fileId);  
printf("ret after PGS MET_Write is \d\n",ret);
if(ret != PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET) {
    if (ret == PGSMET_E_MAND_NOT_SET) {
        printf("some mandatory parameters were not set\n");
    } else {
        printf("ASCII write failed\n");
    }
}

***** write to default non-HDF file *****

ret=PGS_MET_Write(handles[ODL_IN_MEMORY], NULL, NULL);
printf("ret after PGS_MET_Write is %d\n",ret);

if(ret != PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET) {
    if (ret == PGSMET_E_MAND_NOT_SET) {
        printf("some mandatory parameters were not set\n");
    } else {
        printf("ASCII write failed\n");
    }
}

(void)PGS_MET_SDend(sdid1);

PGS_MET_Remove();
free(informationname);

printf("Complete...\n");
return 0;
}

8.2.2 C Code for SDP TOOLKIT Users

/* include files */

#include <PGS_MET.h>
#include <PGS_PC.h>
#include <PGS_tk.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hdf5.h>
#include <hdf.h>
#include <mfhdf.h>
#include <PGS_SMF.h>

#define INVENTORYMETADATA 1
#define ARCHIVEDMETADATA 2
#define ODL_IN_MEMORY 0

int main()
{

/***************************************************************
Declarations.
******************************************************************/
PGSt_MET_all_handles      mdHandles;
PGSt_MET_all_handles      handles;
char                      fileName1[PGSd_PC_FILE_PATH_MAX]="";
char                      fileName2[PGSd_PC_FILE_PATH_MAX]="";
char                      my_HDF_file[PGSd_PC_FILE_PATH_MAX]="";
char                      msg[PGS_SMF_MAX_MSG_SIZE];
char                      mnemonic[PGS_SMF_MAX_MNEMONIC_SIZE];
char                      fileMessage[PGS_SMF_MAX_MSG_SIZE];
int32                     sdid1;
PGSt_SMF_status           ret = PGS_S_SUCCESS;
char                      *informationname;
PGSt_integer             ival =3;
PGSt_double              dval=203.2;
PGSt_integer              fileId, fileId2;
PGSt_integer              i;
PGSt_integer              version;
PGSt_SMF_status           returnStatus;
char                      *mysaval[5];

/*recover file name for fileId=PGSd_MET_MCF_FILE */

version = 1;
fileId = PGSd_MET_MCF_FILE;
returnStatus = PGS_PC_GetReference( fileId, &version, 
                                    fileName1);
if ( returnStatus != PGS_S_SUCCESS )
{
    PGS_SMF_GetMsg( &returnStatus, mnemonic, msg );
    strcpy(fileMessage, msg);
    PGS_SMF_SetDynamicMsg( returnStatus,fileMessage, 
                           "metatest" );
}
else
{
    printf(\"The input file for ID %d is %s\n\",fileId,fileName1);
}

informationname=(char *) malloc(330);
/* Initialize MCF file */

fileId = 10250;
ret=PGS_MET_Init(fileId,handles);

if (ret != PGS_S_SUCCESS)
{
    printf("initialization failed\n");
    return (-1);
}
else
{
    printf("ret after PGS_MET_Init is %d\n", ret);
}

/* test PGS_MET_SetAttr */

ival=667788;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "QAPERCENTINTERPOLATEDDATA.1", &ival);
printf("ret after SetAttr for QAPERCENTINTERPOLATEDDATA.1 is %d\n", ret);

ival=12345;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "QAPercentMissingData.1", &ival);
printf("ret after SetAttr for QAPercentMissingData.1 is %d\n", ret);

ival=123;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "QAPercentOutofBoundsData.1", &ival);
printf("ret after SetAttr for QAPercentOutofBoundsData.1 is %d\n", ret);

ival=23456;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "QAPercentOutofBoundsData.2", &ival);
printf("ret after SetAttr for QAPercentOutofBoundsData.1 is %d\n", ret);

ival=56789;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "QAPercentMissingData.2", &ival);
printf("ret after SetAttr for QAPercentMissingData.1 is %d\n", ret);

strcpy(informationname,"Exercise1");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "AutomaticQualityFlagExplanation.1", &informationname);
printf("ret after SetAttr for AutomaticQualityFlagExplanation.1 is %d\n", ret);

strcpy(informationname,"1997/12/23");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
}
"RangeBeginningDateTime", &informationname);
printf("ret after SetAttr for RangeBeginningDateTime is %d\n", ret);

strcpy(informationname, "1997.07/30");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "RangeBeginningDate", &informationname);
printf("ret after SetAttr for RangeBeginningDate is %d\n", ret);

strcpy(informationname, "ReprocessingplannINVENT");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ReprocessingPlanned", &informationname);
printf("ret after SetAttr for ReprocessingPlanned is %d\n", ret);

strcpy(informationname, "ReprocessingplannARCHIVE");
ret = PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "ReprocessingPlanned", &informationname);
printf("ret after SetAttr for ReprocessingPlanned is %d\n", ret);

strcpy(informationname, "Reprocessingin");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ReprocessingActual", &informationname);
printf("ret after SetAttr for ReprocessingActual is %d\n", ret);

strcpy(informationname, "ID1111");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "LocalGranuleID", &informationname);
printf("ret after SetAttr for LocalGranuleID is %d\n", ret);

strcpy(informationname, "version1234");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "LocalVersionID", &informationname);
printf("ret after SetAttr for LocalVersionID is %d\n", ret);

strcpy(informationname, "Flag1");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "DayNightFlag", &informationname);
printf("ret after SetAttr for DayNightFlag is %d\n", ret);

strcpy(informationname, "Flag1");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "DayNightFlag", &informationname);
printf("ret after SetAttr for DayNightFlag is %d\n", ret);

strcpy(informationname, "information1");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.1", &informationname);
printf("ret after SetAttr for ParameterName is %d\n", ret);

strcpy(informationname, "information2");
ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.2", &informationname);
printf("ret after SetAttr for ParameterName.2 is %d\n", ret);
strcpy(informationname,"information3");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.3", &informationname);
printf("ret after SetAttr for ParameterName is %d\n", ret);

strcpy(informationname,"information4");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.4", &informationname);
printf("ret after SetAttr for ParameterName is %d\n", ret);

dval=111.11;
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "WestBounding Coordinate", &dval);
printf("ret WestBounding Coordinate is %d %f\n", ret, dval);

dval=222.22;
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "northBounding Coordinate", &dval);
printf("ret northBounding Coordinate is %d %f\n", ret, dval);

dval=333.33;
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "EastBounding Coordinate", &dval);
printf("ret EastBounding Coordinate is %d %f\n", ret, dval);

dval=444.44;
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "SouthBounding Coordinate", &dval);
printf("ret SouthBounding Coordinate is %d %f\n", ret, dval);

dval=11.11;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "WestBounding Coordinate", &dval);
printf("ret WestBounding Coordinate is %d %f\n", ret, dval);

dval=22.22;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "northBounding Coordinate", &dval);
printf("ret northBounding Coordinate is %d %f\n", ret, dval);

dval=33.33;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "EastBounding Coordinate", &dval);
printf("ret EastBounding Coordinate is %d %f\n", ret, dval);

dval=44.44;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "SouthBounding Coordinate", &dval);
printf("ret SouthBounding Coordinate is %d %f\n", ret, dval);

/* Get the value of set attribute */
dval=11.11;
ret=PGS_MET_GetSetAttr(handles[INVENTORYMETADATA],
    "SouthBoundingCoordinate", &dval);

printf("after GetSetAttr: ret SouthBoundingCoordinate is %d %f
", ret, dval);

/* write metadata to HDF and ASCII files */
version = 1;
fileId = 5039;

ret = PGS_PC_GetReference(fileId, &version, my_HDF_file);
printf("after PGS_PC_GetReference ret =%d", ret);
printf("after PGS_PC_GetReference my_HDF_file = %s", my_HDF_file);

if (ret == PGS_S_SUCCESS)
{
    ret = PGS_MET_SDstart(my_HDF_file, HDF5_ACC_RDWR, &sdid1);
    printf("after PGS_MET_SDstart sdid1 =%d", sdid1);
}
else
{
    return (-1);
}

printf("After SDstart sdid1 is %d", sdid1);

/***** write INVENTORYMETADATA to HDF file *****/
ret=PGS_MET_Write(handles[INVENTORYMETADATA], "coremetadata", sdid1);
printf("ret after PGS_MET_Write is %d", ret);

if(ret !=PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET)
{
    if (ret == PGSMET_E_MAND_NOT_SET)
    {
        printf("some mandatory parameters were not set\n");
    }
    else
    {
        printf("HDF write failed\n");
    }
}

/***** write ARCHIVEDMETADAT to HDF file *****/
ret=PGS_MET_Write(handles[ARCHIVEDMETADATA], "archivemetadata",
    sdid1);
printf("ret after PGS_MET_Write is %d", ret);
if (ret != PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET) {
    if (ret == PGSMET_E_MAND_NOT_SET) {
        printf("some mandatory parameters were not set\n");
    } else {
        printf("HDF write failed\n");
    }
}

/***** write to non-HDF file *****/

fileId = 5804;
printf("non-hdf file to be written has fileId %d\n", fileId);
ret = PGS_MET_Write(handles[ODL_IN_MEMORY], NULL, fileId);
printf("ret after PGS_MET_Write is %d\n", ret);

if (ret != PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET) {
    if (ret == PGSMET_E_MAND_NOT_SET) {
        printf("some mandatory parameters were not set\n");
    } else {
        printf("ASCII write failed\n");
    }
}

/***** write to default non-HDF file *****/

ret = PGS_MET_Write(handles[ODL_IN_MEMORY], NULL, NULL);
printf("ret after PGS_MET_Write is %d\n", ret);

if (ret != PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET) {
    if (ret == PGSMET_E_MAND_NOT_SET) {
        printf("some mandatory parameters were not set\n");
    } else {
        printf("ASCII write failed\n");
    }
}

(void)PGS_MET_SDend(sdid1);
PGS_MET_Remove();
free(informationname);
8.3 The Metadata Configuration File (MCF) for Codes in Section 8.2

GROUP = INVENTORYMETADATA
GROUPTYPE = MASTERGROUP

/* ECSDataGranule */
GROUP = ECSDataGranule

/* Note: SizeMBECSDataGranule will be set by DSS, */
/* not by the science software. */
OBJECT = SizeMBECSDataGranule
    Data_Location = "DSS"
    NUM_VAL = 1
    TYPE = "DOUBLE"
    Mandatory = "FALSE"
END_OBJECT = SizeMBECSDataGranule

OBJECT = ReprocessingPlanned
    Data_Location = "PGE"
    NUM_VAL = 1
    TYPE = "STRING"
    Mandatory = "TRUE"
END_OBJECT = ReprocessingPlanned

OBJECT = ReprocessingActual
Data_Location = "PGE"
NUM_VAL = 1
TYPE = "STRING"
Mandatory = "TRUE"
END_OBJECT = ReprocessingActual

OBJECT = LocalGranuleID
Data_Location = "PGE"
NUM_VAL = 1
TYPE = "STRING"
Mandatory = "TRUE"
END_OBJECT = LocalGranuleID

OBJECT = DayNightFlag
Data_Location = "PGE"
NUM_VAL = 1
TYPE = "STRING"
Mandatory = "TRUE"
END_OBJECT = DayNightFlag

OBJECT = ProductionDateTime
Data_Location = "TK"
NUM_VAL = 1
TYPE = "DATETIME"
Mandatory = "TRUE"
END_OBJECT = ProductionDateTime

OBJECT = LocalVersionID
Data_Location = "PGE"
NUM_VAL = 1
TYPE = "STRING"
Mandatory = "TRUE"
END_OBJECT = LocalVersionID

END_GROUP = ECSDataGranule

GROUP = MeasuredParameter

OBJECT = MeasuredParameterContainer

Data_Location = "NONE"
Class = "M"
Mandatory = "TRUE"

OBJECT = ParameterName
Data_Location = "PGE"
Class = "M"
TYPE = "STRING"
NUM_VAL = 1
Mandatory = "TRUE"
END_OBJECT = ParameterName

GROUP = QAFlags
Class = "M"
OBJECT = AutomaticQualityFlag
Data_Location = "PGE"
Mandatory = "TRUE"
TYPE = "STRING"
NUM_VAL = 1
END_OBJECT = AutomaticQualityFlag

OBJECT = AutomaticQualityFlagExplanation
   Data_Location = "PGE"
   Mandatory = "TRUE"
   TYPE = "STRING"
   NUM_VAL = 1
END_OBJECT = AutomaticQualityFlagExplanation

OBJECT = OperationalQualityFlag
   Data_Location = "DAAC"
   Mandatory = "FALSE"
   TYPE = "STRING"
   NUM_VAL = 1
END_OBJECT = OperationalQualityFlag

OBJECT = OperationalQualityFlagExplanation
   Data_Location = "DAAC"
   Mandatory = "FALSE"
   TYPE = "STRING"
   NUM_VAL = 1
END_OBJECT = OperationalQualityFlagExplanation

OBJECT = ScienceQualityFlag
   Data_Location = "DP"
   Mandatory = "FALSE"
   TYPE = "STRING"
   NUM_VAL = 1
END_OBJECT = ScienceQualityFlag

OBJECT = ScienceQualityFlagExplanation
   Data_Location = "DP"
   Mandatory = "FALSE"
   TYPE = "STRING"
   NUM_VAL = 1
END_OBJECT = ScienceQualityFlagExplanation

END_GROUP = QAFlags

GROUP = QAStats
   Class = "M"

OBJECT = QAPercentInterpolatedData
   Data_Location = "PGE"
   NUM_VAL = 1
   TYPE = "INTEGER"
   Mandatory = "TRUE"
END_OBJECT = QAPercentInterpolatedData

OBJECT = QAPercentMissingData
   Data_Location = "PGE"
   NUM_VAL = 1
   TYPE = "INTEGER"
   Mandatory = "TRUE"
END_OBJECT = QAPercentMissingData

OBJECT = QAPercentOutOfBoundsData
   Data_Location = "PGE"
   NUM_VAL = 1
TYPE = "INTEGER"
Mandatory = "TRUE"
END_OBJECT = QAPercentOutOfBoundsData

OBJECT = QAPercentCloudCover
   Data_Location = "PGE"
   NUM_VAL = 1
   TYPE = "INTEGER"
   Mandatory = "TRUE"
END_OBJECT = QAPercentCloudCover
END_GROUP = QAStats
END_OBJECT = MeasuredParameterContainer

GROUP = OrbitCalculatedSpatialDomain
OBJECT = OrbitCalculatedSpatialDomainContainer
   Data_Location = "NONE"
   Class = "M"
   Mandatory = "TRUE"

OBJECT = OrbitalModelName
   Data_Location = "PGE"
   Mandatory = "TRUE"
   Class = "M"
   TYPE = "STRING"
   NUM_VAL = 1
END_OBJECT = OrbitalModelName

OBJECT = OrbitNumber
   Data_Location = "PGE"
   Mandatory = "TRUE"
   Class = "M"
   TYPE = "INTEGER"
   NUM_VAL = 1
END_OBJECT = OrbitNumber

OBJECT = Start Orbit Number
   Data_Location = "PGE"
   Mandatory = "TRUE"
   Class = "M"
   TYPE = "INTEGER"
   NUM_VAL = 1
END_OBJECT = StartOrbitNumber

OBJECT = StopOrbitNumber
   Data_Location = "PGE"
   Mandatory = "TRUE"
   Class = "M"
   TYPE = "INTEGER"
   NUM_VAL = 1
END_OBJECT = StopOrbitNumber

OBJECT = EquatorCrossingLongitude
   Data_Location = "PGE"
   Mandatory = "TRUE"
   Class = "M"
   TYPE = "DOUBLE"
   NUM_VAL = 1
END_OBJECT = EquatorCrossingLongitude

OBJECT = EquatorCrossingTime
   Data_Location = "PGE"
   Mandatory = "TRUE"
   Class = "M"
   TYPE = "TIME"
   NUM_VAL = 1
END_OBJECT = EquatorCrossingTime

OBJECT = EquatorCrossingDate
   Data_Location = "PGE"
   Mandatory = "TRUE"
   Class = "M"
   TYPE = "DATE"
   NUM_VAL = 1
END_OBJECT = EquatorCrossingDate

END_OBJECT = OrbitCalculatedSpatialDomainContainer
END_GROUP = OrbitCalculatedSpatialDomain

GROUP = CollectionDescriptionClass

OBJECT = ShortName
   Data_Location = "MCF"
   NUM_VAL = 1
   TYPE = "STRING"
   Mandatory = "TRUE"
   Value = "L7ORF1"
END_OBJECT = ShortName

OBJECT = VersionID
   Data_Location = "MCF"
   NUM_VAL = 1
   TYPE = "STRING"
   Mandatory = "TRUE"
   Value = "1"
END_OBJECT = VersionID

END_GROUP = CollectionDescriptionClass

GROUP = SpatialDomainContainer

GROUP = HorizontalSpatialDomainContainer

   /* ZoneIdentifierClass */
   GROUP = ZoneIdentifierClass
   OBJECT = ZoneIdentifier
      Data_Location = "PGE"
      NUM_VAL = 1
      TYPE = "STRING"
      Mandatory = "TRUE"
   END_OBJECT = ZoneIdentifier
   END_GROUP = ZoneIdentifierClass

   /* BoundingRectangle */
   GROUP = BoundingRectangle
   OBJECT = WestBoundingCoordinate
      Data_Location = "PGE"
NUM_VAL = 1
TYPE = "DOUBLE"
Mandatory = "TRUE"
END_OBJECT = WestBoundingCoordinate

OBJECT = NorthBoundingCoordinate
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "DOUBLE"
  Mandatory = "TRUE"
END_OBJECT = NorthBoundingCoordinate

OBJECT = EastBoundingCoordinate
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "DOUBLE"
  Mandatory = "TRUE"
END_OBJECT = EastBoundingCoordinate

OBJECT = SouthBoundingCoordinate
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "DOUBLE"
  Mandatory = "TRUE"
END_OBJECT = SouthBoundingCoordinate
END_GROUP = BoundingRectangle
END_GROUP = HorizontalSpatialDomainContainer
END_GROUP = SpatialDomainContainer

/* RangeDateTime */
GROUP = RangeDateTime

OBJECT = RangeBeginningTime
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "TIME"
  Mandatory = "TRUE"
END_OBJECT = RangeBeginningTime

OBJECT = RangeEndingTime
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "TIME"
  Mandatory = "TRUE"
END_OBJECT = RangeEndingTime

OBJECT = RangeBeginningDate
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "DATE"
  Mandatory = "TRUE"
END_OBJECT = RangeBeginningDate

OBJECT = RangeEndingDate
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "DATE"
  Mandatory = "TRUE"
END_OBJECT = RangeEndingDate
END_GROUP = RangeDateTime

GROUP = AdditionalAttributes
OBJECT = AdditionalAttributesContainer
  Data_Location = "NONE"
  Class = "M"
  Mandatory = "TRUE"

/* AdditionalAttributes */
OBJECT = AdditionalAttributeName
  Data_Location = "PGE"
  Mandatory = "TRUE"
  TYPE = "STRING"
  Class = "M"
  NUM_VAL = 1
END_OBJECT = AdditionalAttributeName

/* InformationContent */
GROUP = InformationContent
  Class = "M"

OBJECT = ParameterValue
  Data_Location = "PGE"
  Mandatory = "TRUE"
  TYPE = "STRING"
  NUM_VAL = 1
END_OBJECT = ParameterValue

END_GROUP = InformationContent
END_OBJECT = AdditionalAttributesContainer
END_GROUP = AdditionalAttributes

GROUP = OrbitParametersGranule
OBJECT = OrbitalParametersPointer
  Data_Location = "PGE"
  Mandatory = "TRUE"
  TYPE = "STRING"
  NUM_VAL = 1
END_OBJECT = OrbitalParametersPointer

END_GROUP = OrbitParametersGranule

/* StorageMediumClass */
GROUP = StorageMediumClass
OBJECT = StorageMedium
  Data_Location = "PGE"
  NUM_VAL = 10
  TYPE = "STRING"
  Mandatory = "TRUE"
END_OBJECT = StorageMedium
END_GROUP = StorageMediumClass
END_GROUP = INVENTORYMETADATA

GROUP = ARCHIVEDMETADATA
GROUPTYPE = MASTERGROUP

/* BoundingRectangle */
GROUP = BoundingRectangle
OBJECT = WestBoundingCoordinate
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "DOUBLE"
  Mandatory = "TRUE"
END_OBJECT = WestBoundingCoordinate

OBJECT = NorthBoundingCoordinate
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "DOUBLE"
  Mandatory = "TRUE"
END_OBJECT = NorthBoundingCoordinate

OBJECT = EastBoundingCoordinate
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "DOUBLE"
  Mandatory = "TRUE"
END_OBJECT = EastBoundingCoordinate

OBJECT = SouthBoundingCoordinate
  Data_Location = "PGE"
  NUM_VAL = 1
  TYPE = "DOUBLE"
  Mandatory = "TRUE"
END_OBJECT = SouthBoundingCoordinate
END_GROUP = BoundingRectangle
END_GROUP = ARCHIVEDMETADATA
END

8.4 The ODL Output File Which Results from Running Code in Section 8.2

******************************************************************************
******************************************************************************
/*
/* This is a working version of the MCF template that will be */
/* supplied with the next SDP Toolkit. This MCF template will */
/* NOT be official until the SDP Toolkit is released. All */
/* details are subject to change. */
/* */
/* This MCF file represents the ODL which is expected to be */
/* created when either Data Server or the MetaDataWorks tool */
/* uses the contents of an ESST's INVENTORYMETADATA section in */
/* order to generate an ESST-specific MCF. The level of */
/* metadata coverage presented here corresponds to the metadata */
/* requirement for granules in Full Class as described in */
/* Appendix B of DID 311 and Section 2.5 of the document 'BNF*/
/* Representation of the B.0 Earth Science Data Model for the */
GROUP                  = INVENTORYMETADATA
GROUPTYPE              = MASTERGROUP

/* ECSDataGranule */

GROUP                  = ECSDATAGRANULE

OBJECT                 = REPROCESSINGPLANNED
  NUM_VAL              = 1
  VALUE                = "ReprocessingplannINVENTR"
END_OBJECT             = REPROCESSINGPLANNED

OBJECT                 = REPROCESSINGACTUAL
  NUM_VAL              = 1
  VALUE                = "Reprocessin"
END_OBJECT             = REPROCESSINGACTUAL

OBJECT                 = LOCALGRANULEID
  NUM_VAL              = 1
  VALUE                = "ID1111"
END_OBJECT             = LOCALGRANULEID

OBJECT                 = DAYNIGHTFLAG
  NUM_VAL              = 1
  VALUE                = "Flag1"
END_OBJECT             = DAYNIGHTFLAG

OBJECT                 = PRODUCTIONDATETIME
  NUM_VAL              = 1
  VALUE                = "1999-11-23T18:16:01.000Z"
END_OBJECT             = PRODUCTIONDATETIME

OBJECT                 = LOCALVERSIONID
  NUM_VAL              = 1
  VALUE                = "version1234"
END_OBJECT             = LOCALVERSIONID

END_GROUP              = ECSDATAGRANULE

/* MeasuredParameter */

GROUP                  = MEASUREDPARAMETER

OBJECT                 = MEASUREDPARAMETERCONTAINER
  CLASS                = "1"

OBJECT                 = PARAMETERNAME
  CLASS                = "1"
  NUM_VAL              = 1
  VALUE                = "information1"
END_OBJECT = PARAMETERNAME

/* QAFlags */

GROUP = QAFLAGS
CLASS = "1"

OBJECT = AUTOMATICQUALITYFLAG
NUM_VAL = 1
CLASS = "1"
VALUE = "NOT SET"
END_OBJECT = AUTOMATICQUALITYFLAG

OBJECT = AUTOMATICQUALITYFLAGEXPLANATION
NUM_VAL = 1
CLASS = "1"
VALUE = "Exercise1"
END_OBJECT = AUTOMATICQUALITYFLAGEXPLANATION

END_GROUP = QAFLAGS

/* QAStats */

GROUP = QASTATS
CLASS = "1"

OBJECT = QAPERCENTINTERPOLATEDDATA
NUM_VAL = 1
CLASS = "1"
VALUE = 667788
END_OBJECT = QAPERCENTINTERPOLATEDDATA

OBJECT = QAPERCENTMISSINGDATA
NUM_VAL = 1
CLASS = "1"
VALUE = 12345
END_OBJECT = QAPERCENTMISSINGDATA

OBJECT = QAPERCENTOUTOFBOUNDSDATA
NUM_VAL = 1
CLASS = "1"
VALUE = 123
END_OBJECT = QAPERCENTOUTOFBOUNDSDATA

OBJECT = QAPERCENTCLOUDCOVER
NUM_VAL = 1
CLASS = "1"
VALUE = "NOT SET"
END_OBJECT = QAPERCENTCLOUDCOVER

END_GROUP = QASTATS

END_OBJECT = MEASUREDPARAMETERCONTAINER
OBJECT = MEASUREDPARAMETERCONTAINER
CLASS = "2"
OBJECT = PARAMETERNAME
CLASS = "2"
NUM VAL = 1
VALUE = "information2"
END_OBJECT = PARAMETERNAME

/* QAFlags */

GROUP = QAFLAGS
CLASS = "2"

OBJECT = AUTOMATICQUALITYFLAG
NUM VAL = 1
CLASS = "2"
VALUE = "NOT SET"
END_OBJECT = AUTOMATICQUALITYFLAG

OBJECT = AUTOMATICQUALITYFLAGEXPLANATION
NUM VAL = 1
CLASS = "2"
VALUE = "NOT SET"
END_OBJECT = AUTOMATICQUALITYFLAGEXPLANATION

END_GROUP = QAFLAGS

/* QAStats */

GROUP = QASTATS
CLASS = "2"

OBJECT = QAPERCENTINTERPOLATEDDATA
NUM VAL = 1
CLASS = "2"
VALUE = "NOT SET"
END_OBJECT = QAPERCENTINTERPOLATEDDATA

OBJECT = QAPERCENTMISSINGDATA
NUM VAL = 1
CLASS = "2"
VALUE = 56789
END_OBJECT = QAPERCENTMISSINGDATA

OBJECT = QAPERCENTOUTOFBOUNDSDATA
NUM VAL = 1
CLASS = "2"
VALUE = 23456
END_OBJECT = QAPERCENTOUTOFBOUNDSDATA

OBJECT = QAPERCENTCLOUDCOVER
NUM VAL = 1
CLASS = "2"
VALUE = "NOT SET"
END_OBJECT = QAPERCENTCLOUDCOVER

END_GROUP = QASTATS
END_OBJECT = MEASUREDPARAMETERCONTAINER

OBJECT = MEASUREDPARAMETERCONTAINER
CLASS = "3"

OBJECT = PARAMETERNAME
CLASS = "3"
NUM_VAL = 1
VALUE = "information3"
END_OBJECT = PARAMETERNAME

/* QAFlags */

GROUP = QAFLAGS
CLASS = "3"

OBJECT = AUTOMATICQUALITYFLAG
NUM_VAL = 1
CLASS = "3"
VALUE = "NOT SET"
END_OBJECT = AUTOMATICQUALITYFLAG

OBJECT = AUTOMATICQUALITYFLAGEXPLANATION
NUM_VAL = 1
CLASS = "3"
VALUE = "NOT SET"
END_OBJECT = AUTOMATICQUALITYFLAGEXPLANATION

END_GROUP = QAFLAGS

/* QAStats */

GROUP = QASTATS
CLASS = "3"

OBJECT = QAPERCENTINTERPOLATEDDATA
NUM_VAL = 1
CLASS = "3"
VALUE = "NOT SET"
END_OBJECT = QAPERCENTINTERPOLATEDDATA

OBJECT = QAPERCENTMISSINGDATA
NUM_VAL = 1
CLASS = "3"
VALUE = "NOT SET"
END_OBJECT = QAPERCENTMISSINGDATA

OBJECT = QAPERCENTOUTOFBOUNDSDATA
NUM_VAL = 1
CLASS = "3"
VALUE = "NOT SET"
END_OBJECT = QAPERCENTOUTOFBOUNDSDATA

OBJECT = QAPERCENTCLOUDCOVER
NUM_VAL = 1
CLASS = "3"
VALUE = "NOT SET"
END_OBJECT = QAPERCENTCLOUDCOVER
END_GROUP = QASTATS
END_OBJECT = MEASUREDPARAMETERCONTAINER

OBJECT = MEASUREDPARAMETERCONTAINER
CLASS = "4"

OBJECT = PARAMETERNAME
CLASS = "4"
NUM_VAL = 1
VALUE = "information4"
END_OBJECT = PARAMETERNAME

OBJECT = PARAMETERNAME
CLASS = "4"

/* QAFlags */

GROUP = QAFLAGS
CLASS = "4"

OBJECT = AUTOMATICQUALITYFLAG
NUM_VAL = 1
CLASS = "4"
VALUE = "NOT SET"
END_OBJECT = AUTOMATICQUALITYFLAG

OBJECT = AUTOMATICQUALITYFLAGEXPLANATION
NUM_VAL = 1
CLASS = "4"
VALUE = "NOT SET"
END_OBJECT = AUTOMATICQUALITYFLAGEXPLANATION

END_GROUP = QAFLAGS

/* QAStats */

GROUP = QASTATS
CLASS = "4"

OBJECT = QAPERCENTINTERPOLATEDDATA
NUM_VAL = 1
CLASS = "4"
VALUE = "NOT SET"
END_OBJECT = QAPERCENTINTERPOLATEDDATA

OBJECT = QAPERCENTMISSINGDATA
NUM_VAL = 1
CLASS = "4"
VALUE = "NOT SET"
END_OBJECT = QAPERCENTMISSINGDATA

OBJECT = QAPERCENTOUTOFBOUNDSDATA
NUM_VAL = 1
CLASS = "4"
VALUE = "NOT SET"
END_OBJECT = QAPERCENTOUTOFBOUNDSDATA

END_GROUP = QASTATS
OBJECT = QAPERCENTCLOUDCOVER
    NUM_VAL = 1
    CLASS = "4"
    VALUE = "NOT SET"
END_OBJECT = QAPERCENTCLOUDCOVER

END_GROUP = QASTATS

END_OBJECT = MEASUREDPARAMETERCONTAINER

END_GROUP = MEASUREDPARAMETER

GROUP = ORBITCALCULATEDSPATIALDOMAIN

OBJECT = ORBITCALCULATEDSPATIALDOMAINCONTAINER
    CLASS = "M"

OBJECT = ORBITALMODELNAME
    CLASS = "M"
    NUM_VAL = 1
    VALUE = "NOT SET"
END_OBJECT = ORBITALMODELNAME

OBJECT = ORBITNUMBER
    CLASS = "M"
    NUM_VAL = 1
    VALUE = "NOT SET"
END_OBJECT = ORBITNUMBER

OBJECT = STARTORBITNUMBER
    CLASS = "M"
    NUM_VAL = 1
    VALUE = "NOT SET"
END_OBJECT = STARTORBITNUMBER

OBJECT = STOPORBITNUMBER
    CLASS = "M"
    NUM_VAL = 1
    VALUE = "NOT SET"
END_OBJECT = STOPORBITNUMBER

OBJECT = EQUATORCROSSINGLONGITUDE
    CLASS = "M"
    NUM_VAL = 1
    VALUE = "NOT SET"
END_OBJECT = EQUATORCROSSINGLONGITUDE

OBJECT = EQUATORCROSSINGTIME
    CLASS = "M"
    NUM_VAL = 1
    VALUE = "NOT SET"
END_OBJECT = EQUATORCROSSINGTIME

OBJECT = EQUATORCROSSINGDATE
    CLASS = "M"
    NUM_VAL = 1
    VALUE = "NOT SET"
END_OBJECT = EQUATORCROSSINGDATE
END_OBJECT             = ORBITCALCULATEDSPATIALDOMAINCONTAINER
END_GROUP              = ORBITCALCULATEDSPATIALDOMAIN

/* CollectionDescriptionClass */
GROUP                  = COLLECTIONDESCRIPTIONCLASS
OBJECT                 = SHORTNAME
  NUM_VAL              = 1
  VALUE                = "L7ORF1"
END_OBJECT             = SHORTNAME

OBJECT                 = VERSIONID
  NUM_VAL              = 1
  VALUE                = "1"
END_OBJECT             = VERSIONID

END_GROUP              = COLLECTIONDESCRIPTIONCLASS

/* SpatialDomainContainer */
GROUP                  = SPATIALDOMAINCONTAINER

GROUP                  = HORIZONTALSPATIALDOMAINCONTAINER

/* ZoneIdentifierClass */
GROUP                  = ZONEIDENTIFIERCLASS
OBJECT                 = ZONEIDENTIFIER
  NUM_VAL              = 1
  VALUE                = "NOT SET"
END_OBJECT             = ZONEIDENTIFIER

END_GROUP              = ZONEIDENTIFIERCLASS

/* BoundingRectangle */
GROUP                  = BOUNDINGRECTANGLE

OBJECT                 = WESTBOUNDINGCOORDINATE
  NUM_VAL              = 1
  VALUE                = 11.110000
END_OBJECT             = WESTBOUNDINGCOORDINATE

OBJECT                 = NORTHBOUNDINGCOORDINATE
  NUM_VAL              = 1
  VALUE                = 22.220000
END_OBJECT             = NORTHBOUNDINGCOORDINATE

OBJECT                 = EASTBOUNDINGCOORDINATE
  NUM_VAL              = 1
  VALUE                = 33.330000
END_OBJECT             = EASTBOUNDINGCOORDINATE
OBJECT = SOUTHBOUNDINGCOORDINATE
NUM_VAL = 1
VALUE = 44.440000
END_OBJECT = SOUTHBOUNDINGCOORDINATE

END_GROUP = BOUNDRINGRECTANGLE

END_GROUP = HORIZONTALSPATIALDOMAINCONTAINER

END_GROUP = SPATIALDOMAINCONTAINER

/* RangeDateTime */

GROUP = RANGEDATETIME

OBJECT = RANGEBEGINNINGTIME
NUM_VAL = 1
VALUE = "NOT SET"
END_OBJECT = RANGEBEGINNINGTIME

OBJECT = RANGEENDINGTIME
NUM_VAL = 1
VALUE = "NOT SET"
END_OBJECT = RANGEENDINGTIME

OBJECT = RANGEBEGINNINGDATE
NUM_VAL = 1
VALUE = "1997.07/30"
END_OBJECT = RANGEBEGINNINGDATE

OBJECT = RANGEENDINGDATE
NUM_VAL = 1
VALUE = "NOT SET"
END_OBJECT = RANGEENDINGDATE

END_GROUP = RANGEDATETIME

GROUP = ADDITIONALATTRIBUTES

OBJECT = ADDITIONALATTRIBUTESCONTAINER
CLASS = "M"

/* AdditionalAttributes */

OBJECT = ADDITIONALATTRIBUTENAME
CLASS = "M"
NUM_VAL = 1
VALUE = "NOT SET"
END_OBJECT = ADDITIONALATTRIBUTENAME

/* InformationContent */

GROUP = INFORMATIONCONTENT
CLASS = "M"

OBJECT = PARAMETERVALUE
8.5 The files filetable.temp and PCF file

The following file are used by MTD TOOLKIT (Section 8.5.1) or SDP TOOLKIT (Section 8.5.2).

8.5.1 The file filetable.temp used for example in Section 8.2.1

This example is used by the MTD Toolkit:

```
# This file is needed for testing TIME tools. Only the Path for the files need to be changed.
# The following IDs are defined in the TOOLKIT and they SHOULD NOT be changed

10100|LogStatus|/tk/TOOLKIT_MTD/test/test_MET_HDF5/LogStatus
5000|configfile.dat|/tk/TOOLKIT_MTD/runtime/configfile.dat
10252|GetAttrtemp|/tk/TOOLKIT_MTD/test/test_MET_HDF5/GetAttrtemp
10254|MCFWrite.temp|/tk/TOOLKIT_MTD/test/test_MET_HDF5/MCFWrite.temp
10255|AsciiDump|/tk/TOOLKIT_MTD/test/test_MET_HDF5/AsciiDump
10256|temporary.MCF|/tk/TOOLKIT_MTD/test/test_MET_HDF5/temporary.MCF
10301|leapsec.dat|/tk/TOOLKIT_MTD/database/common/TD/leapsec.dat
```
The PCF file used for example in Section 8.2.2

This example is used by the SDP Toolkit:

```bash
# Process Control File: PCF.mypcf
# Remember to reset the environment variable PGS_PC_INFO_FILE
# to point to the instance of your PCF file
# Entries preceded by the comment: (DO NOT REMOVE THIS ENTRY)
# are deemed especially critical and should not be removed for
# any reason (although the values of the various fields of such an
# entry may be configurable).
#
# ---------------------------------------------------------------
# SYSTEM RUNTIME PARAMETERS
# ---------------------------------------------------------------
# This section contains unique identifiers used to track instances of
# a PGE run, versions of science software, etc. This section must
# contain exactly two entries. These values will be inserted by
# ECS just before a PGE is executed. At the SCF the values may be set
# to anything but these values are not normally user definable and user
# values will be ignored/overwritten at the DAAC.
#
# Production Run ID - unique production instance identifier
# (DO NOT REMOVE THIS ENTRY)
# ---------------------------------------------------------------
1
# Software ID - unique software configuration identifier
# (DO NOT REMOVE THIS ENTRY)
# ---------------------------------------------------------------
1
```
# PRODUCT INPUT FILES

This section is intended for standard product inputs, i.e., major input files such as Level 0 data files.

Each logical ID may have several file instances, as given by the version number in the last field.

Next non-comment line is the default location for PRODUCT INPUT FILES

WARNING! DO NOT MODIFY THIS LINE unless you have relocated these data set files to the location specified by the new setting.

The following are for the PGS_GCT tool only. The IDs are defined in the PGS_GCT.h file. These entries are essential for the State Plane Projection, but can otherwise be deleted or commented out.

file for Constant & Unit Conversion (CUC) tools

IMPORTANT NOTE: THIS FILE WILL BE SUPPLIED AFTER TK4 DELIVERY!

Metadata Configuration File (MCF) is a template to be filled in by the Instrument teams. MCFWrite.temp is a scratch file used to dump the MCF prior to writing to the hdf file. GetAttr.temp is similarly used to dump metadata from the hdf attributes and is used by PGS_MET_GetPCAttr. (DO NOT REMOVE THESE ENTRIES)

Metadata Configuration File (MCF) is a template to be filled in by the Instrument teams. MCFWrite.temp is a scratch file used to dump the MCF prior to writing to the hdf file. GetAttr.temp is similarly used to dump metadata from the hdf attributes and is used by PGS_MET_GetPCAttr. (DO NOT REMOVE THESE ENTRIES)

PRODUCT OUTPUT FILES

This section is intended for standard product outputs, i.e., HDF-EOS files generated by this PGE.

Each logical ID may have several file instances, as given by the version number in the last field.

Next line is the default location for PRODUCT OUTPUT FILES
This file is created when PGS_MET_Write is used with an intention
to write an ASCII representation of the MCF in memory. The user is
allowed to change the name and path if required.

# SUPPORT INPUT FILES

This section is intended for minor input files, e.g., calibration
files.

Each logical ID may have several file instances, as given by the
version number in the last field.

Next line is the default location for SUPPORT INPUT FILES
! ~/runtime

# leap seconds (TAI-UTC) file (DO NOT REMOVE THIS ENTRY)
10301|leapsec.dat|~/database/common/TD||1

# polar motion and UTC-UT1 file (DO NOT REMOVE THIS ENTRY)
10401|utcpole.dat|~/database/common/CSC||1

# earth model tags file (DO NOT REMOVE THIS ENTRY)
10402|earthfigure.dat|~/database/common/CSC||1

# JPL planetary ephemeris file (binary form) (DO NOT REMOVE THIS ENTRY)
10601|de200.eos|~/database/$BRAND/CBP||1

# spacecraft tag definition file (DO NOT REMOVE THIS ENTRY)
10801|sc_tags.dat|~/database/common/EPH||1

# units conversion definition file (DO NOT REMOVE THIS ENTRY)
10302|udunits.dat|~/database/common/CUC||1

# SUPPORT OUTPUT FILES


# This section is intended for minor output files, e.g., log files.
# Each logical ID may have several file instances, as given by the
# version number in the last field.
#
# Next line is default location for SUPPORT OUTPUT FILES
! ~/runtime
#
# These files support the SMF log functionality. Each run will cause
# status information to be written to 1 or more of the Log files. To
# simulate DAAC operations, remove the 3 Logfiles between test runs.
# Remember: all executables within a PGE will contribute status data to
# the same batch of log files. (DO NOT REMOVE THESE ENTRIES)
#
10100|LogStatus|||1
10101|LogReport|||1
10102|LogUser|||1
10103|TmpStatus|||1
10104|TmpReport|||1
10105|TmpUser|||1
10110|MailFile|||1
#
# ASCII file which stores pointers to runtime SMF files in lieu of
# loading them to shared memory, which is a TK5 enhancement.
# (DO NOT REMOVE THIS ENTRY)
#
10111|ShmMem|||1
#
# USER DEFINED RUNTIME PARAMETERS
#
# This section is intended for parameters used as PGE input.
# Note: these parameters may NOT be changed dynamically.
#
5804|ProductMetadataFile|100:1
10255|reference output product|102:1
#
# These parameters are required to support the PGS_SMF_Send...() tools.
# If the first parameter (TransmitFlag) is disabled, then none of the
# other parameters need to be set. By default, this functionality has been
# disabled. To enable, set TransmitFlag to 1 and supply the other 3
# parameters with local information. (DO NOT REMOVE THESE ENTRIES)
#
10109|TransmitFlag; 1=transmit,0=disable|0
10106|RemoteHost|sandcrab
10107|RemotePath|/usr/kwan/test/PC/data
10108|EmailAddresses|kwan@eos.hitc.com
# The following runtime parameters define various logging options.
# Parameters described as lists should be space (i.e. ' ') separated.
# The logical IDs 10117, 10118, 10119 listed below are for OPTIONAL
# control of SMF logging. Any of these logical IDs which is unused by a
# PGE may be safely commented out (e.g. if logging is not disabled for
# any status level, then the line beginning 10117 may be commented out
#   --------------------
# 10114|Logging Control; 0=disable logging, 1=enable logging|1
# 10115|Trace Control; 0=no trace, 1=error trace, 2=full trace|0
# 10116|Process ID logging; 0=don't log PID, 1=log PID|0
# 10117|Disabled status level list (e.g. W S F)|
# 10118|Disabled seed list|
# 10119|Disabled status code list|
#   --------------------
# Toolkit version for which this PCF was intended.
# DO NOT REMOVE THIS VERSION-ENTRY!
#   --------------------
# 10220|Toolkit version string|SCF TK5.2.7.3
#   --------------------
# The following parameters define the ADEOS-II TMDF values (all values
# are assumed to be floating point types). The ground reference time
# should be in TAI93 format (SI seconds since 12 AM UTC 1993-01-01).
# These formats are only prototypes and are subject to change when
# the ADEOS-II TMDF values are clearly defined. PGEs that do not access
# ADEOS-II L0 data files do not require these parameters. In this case
# they may be safely commented out, otherwise appropriate values should
# be supplied.
#   --------------------
# 10120|ADEOS-II s/c reference time|
# 10121|ADEOS-II ground reference time|
# 10122|ADEOS-II s/c clock period|
#   --------------------
# The following parameter defines the TRMM UTCF value (the value is
# assumed to be a floating point type). PGEs that do not access TRMM
# data of any sort do not require this parameter. In this case it may be
# safely commented out, otherwise an appropriate value should be
# supplied.
#   --------------------
# 10123|TRMM UTCF value|
#   --------------------
# The following parameter defines the Epoch date to be used for the
# interpretation (conversion) of NASA PB5C times (the Epoch date should
# be specified here in CCSDS ASCII format--A or B) (reserved for future
# use--this quantity is not referenced in TK 5.2). This entry may be
# safely commented out or deleted.
#   --------------------
# 10124|NASA PB5C time Epoch date (ASCII UTC)|
The following parameter is a "mask" for the attitude data quality flag. The value should be specified as an unsigned integer specifying those bits of the attitude data quality flag that should be considered fatal (i.e. the attitude data associated with the quality flag should be REJECTED/IGNORED).

ECS DPS trigger for PGE debug runs

NOTICE TO PGE DEVELOPERS: PGEs which have a debug mode need to examine this parameter to evaluate activation rule (DO NOT REMOVE THIS ENTRY)

This entry defines the IP address of the processing host and is used by the Toolkit when generating unique Intermediate and Temporary file names. The Toolkit no longer relies on the PGS_HOST_PATH environment variable to obtain this information. (DO NOT REMOVE THIS ENTRY)

Local IP Address of 'ether' | 155.157.31.87

INTERMEDIATE INPUT

This section is intended for intermediate input files, i.e., files which are output by an earlier PGE but which are not standard products.
Each logical ID may have only one file instance.
Last field on the line is ignored.

Next line is default location for INTERMEDIATE INPUT FILES
~/runtime

INTERMEDIATE OUTPUT

This section is intended for intermediate output files, i.e., files which are to be input to later PGEs, but which are not standard products.
Each logical ID may have only one file instance.
Last field on the line is ignored.

Next line is default location for INTERMEDIATE OUTPUT FILES
~/runtime
# TEMPORARY I/O
# This section is intended for temporary files, i.e., files
# which are generated during a PGE run and deleted at PGE termination.
# Entries in this section are generated internally by the Toolkit.
# DO NOT MAKE MANUAL ENTRIES IN THIS SECTION.
#
# Next line is default location for TEMPORARY FILES
! ~/runtime
#
? END
9. Zonal Average Data

9.1 Introduction

This section will describe the routines available for storage and manipulation of HDF-EOS Zonal Average Data. A Zonal Average data set is similar to a swath in that it contains a series of data fields of two or more dimensions. The main difference between a Zonal Average and a Swath is that a Zonal Average is not associated with specific geolocation information.

A standard Zonal Average is made up of two primary parts: data fields and dimensions. Each of the parts of a Zonal Average is described in detail in the following subsections.

9.1.1 Data Fields

Data fields are the main part of a Zonal Average from a science perspective. Data fields usually contain the raw data (often as counts) taken by the sensor or parameters derived from that data on a value-for-value basis. All the other parts of the Zonal Average exist to provide information about the data fields or to support particular types of access to them. Data fields typically are two-dimensional arrays, but can have as few as one dimension or as many as eight, in the current library implementation. They can have any valid C data type.

9.1.2 Dimensions

Dimensions define the axes of the data fields by giving them names and sizes. In using the library, dimensions must be defined before they can be used to describe data fields.

Every axis of each data field must have a dimension associated with it. However, there is no requirement that they all be unique. In other words, different data fields may share the same named dimension. In fact, sharing dimension names allows the Zonal Average interface to make some assumptions about the data fields involved which can reduce the complexity of the file and simplify the program creating or reading the file.

9.2 Applicability

The Zonal Average data model is most useful for satellite [or similar] data at a low level of processing. The Zonal Average model is best suited to data at EOS processing levels 1A, 1B, and 2.

9.3 The Zonal Average Data Interface

The ZA interface consists of routines for storing, retrieving, and manipulating data in zonal average data sets.
9.3.1 ZA API Routines

All C routine names in the zonal average data interface have the prefix “HE5_ZA” and the equivalent FORTRAN routine names are prefixed by “he5_za”. The ZA routines are classified into the following categories:

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

The ZA function calls are listed in Table 9-1 and are described in detail in the Software Reference Guide that accompanies this document. The page number column in the following table refers to the Software Reference Guide.

<table>
<thead>
<tr>
<th>Category</th>
<th>Routine Name</th>
<th>Description</th>
<th>Page Nos.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HE5_ZAopen</td>
<td>Opens or creates HDF file in order to create, read, or write a zonal average</td>
<td>2-336</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAcreate</td>
<td>Creates a zonal average within the file</td>
<td>2-298</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAattach</td>
<td>Attaches to an existing zonal average within the file</td>
<td>2-292</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdetach</td>
<td>Detaches from zonal average interface</td>
<td>2-307</td>
</tr>
<tr>
<td>Access</td>
<td>HE5_ZAclose</td>
<td>Closes file</td>
<td>2-296</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdefdim</td>
<td>Defines a new dimension within the zonal average</td>
<td>2-304</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdefine</td>
<td>Defines a new data field within the zonal average</td>
<td>2-306</td>
</tr>
<tr>
<td>Definition</td>
<td>HE5_ZAdefcomp</td>
<td>Defines a field compression scheme</td>
<td>2-302</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdefchunk</td>
<td>Define chunking parameters</td>
<td>2-299</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdefcomchunk</td>
<td>Defines compression with automatic chunking</td>
<td>2-300</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAsetalias</td>
<td>Defines alias for data field</td>
<td>2-346</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdropalias</td>
<td>Removes alias from the list of field aliases</td>
<td>2-309</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAfieldrename</td>
<td>Changes the field name</td>
<td>2-311</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwrite</td>
<td>Writes data to a zonal average field</td>
<td>2-352</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAread</td>
<td>Reads data from a zonal average field</td>
<td>2-337</td>
</tr>
<tr>
<td>Basic I/O</td>
<td>HE5_ZAwriteattr</td>
<td>Writes/updates attribute in a zonal average</td>
<td>2-354</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAreadattr</td>
<td>Reads attribute from a zonal average</td>
<td>2-339</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwritegrattr</td>
<td>Writes/updates group attribute in a zonal average</td>
<td>2-360</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwritelocattr</td>
<td>Writes/updates group attribute in a zonal average</td>
<td>2-362</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAwrite datameta</td>
<td>Writes field metadata for an existing zonal average</td>
<td>2-356</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAread grattr</td>
<td>Reads attribute from a zonal average</td>
<td>2-344</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAread locattr</td>
<td>Reads attribute from a zonal average</td>
<td>2-345</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAset fillvalue</td>
<td>Sets fill value for the specified field</td>
<td>2-350</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAget fillvalue</td>
<td>Retrieves fill value for the specified field</td>
<td>2-316</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAalias info</td>
<td>Retrieves information about field aliases</td>
<td>2-291</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAget alias list</td>
<td>Retrieves list and number of aliases in a data group</td>
<td>2-312</td>
</tr>
<tr>
<td></td>
<td>HE5_ZAinquiry dims</td>
<td>Retrieves information about dimensions defined in zonal average</td>
<td>2-323</td>
</tr>
</tbody>
</table>
### Table 9-1. Summary of the Zonal Average 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>Inquiry</td>
<td>HE5_ZAinquire</td>
<td>he5_zainquire</td>
<td>Retrieves info about the data fields defined</td>
<td>2-331</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAinqattrs</td>
<td>he5_zainqattrs</td>
<td>Retrieves number and names of attributes defined</td>
<td>2-320</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAinqdatatype</td>
<td>he5_zaidtype</td>
<td>Returns data type information about specified fields in a zonal average</td>
<td>2-321</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAinqfldalias</td>
<td>he5_zainqfldalias</td>
<td>Returns information about data fields &amp; aliases defined in a zonal average</td>
<td>2-326</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAchunkinfo</td>
<td>he5_zachunkinfo</td>
<td>Retrieves chunking information</td>
<td>2-294</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAinqgrpattrs</td>
<td>he5_zainqgrpattrs</td>
<td>Retrieves information about group attributes defined in zonal average</td>
<td>2-328</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAinqlocattrs</td>
<td>he5_zainqlocattrs</td>
<td>Retrieves information about local attributes defined in zonal average</td>
<td>2-329</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAlocattrinfo</td>
<td>he5_zalocattrinfo</td>
<td>Returns information about a data field's local attribute(s)</td>
<td>2-333</td>
<td></td>
</tr>
<tr>
<td>Inquiry</td>
<td>HE5_ZAnentries</td>
<td>he5_zanentries</td>
<td>Returns number of entries and descriptive string buffer size for a specified entity</td>
<td>2-335</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAdiminfo</td>
<td>he5_zadiminfo</td>
<td>Retrieves size of specified dimension</td>
<td>2-308</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAattrinfo</td>
<td>he5_zattrinfo</td>
<td>Returns info about zonal average attributes</td>
<td>2-293</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAgrpattrinfo</td>
<td>he5_zagroupattrinfo</td>
<td>Returns information about a zonal average group attribute</td>
<td>2-317</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAinfo</td>
<td>he5_zainfo</td>
<td>Retrieves information about a specific data field</td>
<td>2-318</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAcompinfo</td>
<td>he5_zacomponentinfo</td>
<td>Retrieves compression information about a field</td>
<td>2-297</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAinfo</td>
<td>he5_zainfo</td>
<td>Retrieves number and names of zas in file</td>
<td>2-332</td>
<td></td>
</tr>
<tr>
<td>External Files</td>
<td>HE5_ZAsetextdata</td>
<td>he5_zasettextdata</td>
<td>Sets external data set</td>
<td>2-315</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAgetextdata</td>
<td>he5_zagettextdata</td>
<td>Gets external data set</td>
<td>2-347</td>
<td></td>
</tr>
<tr>
<td>External Data Sets</td>
<td>HE5_ZAsetdimscale</td>
<td>he5_zasetdimscale</td>
<td>Sets dimension scale for a field dimension within the ZA</td>
<td>2-347</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAgetdimscale</td>
<td>he5_zagetdimscale</td>
<td>Gets dimension scale for a field dimension within the ZA</td>
<td>2-313</td>
<td></td>
</tr>
<tr>
<td>Dimension Scale</td>
<td>HE5_ZAwritescaleattr</td>
<td>he5_zawritedscaleattr</td>
<td>Writes/Updates a dimension scale attribute in a specific ZA</td>
<td>2-357</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAreadscaleattr</td>
<td>he5_zareadscaleattr</td>
<td>Reads a dimension scale attribute from a specific dimension</td>
<td>2-340</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAinqscaleattr</td>
<td>he5_zainqscaleattr</td>
<td>Retrieves information about the attributes defined for a specific dimension scale</td>
<td>2-324</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HE5_ZAgetscaleattr</td>
<td>he5_zagetscaleattr</td>
<td>Returns information about attribute(s) in a specific dimension scale</td>
<td>2-310</td>
<td></td>
</tr>
</tbody>
</table>

#### 9.3.2 File Identifiers

As with all HDF-EOS interfaces, file identifiers in the HE5_ZA interface are of hid_t HDF5 type, each uniquely identifying one open data file. They are not interchangeable with other file identifiers created with other interfaces.

#### 9.3.3 Zonal Average Identifiers

Before a zonal average data set is accessed, it is identified by a name which is assigned to it upon its creation. The name is used to obtain a zonal average identifier. After a zonal average data set has been opened for access, it is uniquely identified by its zonal average identifier.
9.4 Programming Model

The programming model for accessing a zonal average data set through the HE5_ZA interface is as follows:

1. Open the file and initialize the HE5_ZA interface by obtaining a file ID from a file name.
2. Open or create a zonal average object by obtaining a zonal average ID from a zonal average name.
3. Perform desired operations on the data set.
4. Close the zonal average data set by disposing of the zonal average ID.
5. Terminate zonal average access to the file by disposing of the file ID.

/* In this example we (1) open an HDF-EOS file, (2) create the ZA object within the file, and (3) define the ZA field dimensions. */

Open a new HDF-EOS ZA file, "ZA.he5". Assuming that this file may not exist, we are using "H5F_ACC_TRUNC" access code. The "HE5_ZAopen" function returns the ZA file ID, zafid, which is used to identify the file in subsequent calls to the HDF-EOS library functions. */

zafid = HE5_ZAopen("ZA.he5", H5F_ACC_TRUNC);

/* Create the ZA, "ZA1", within the file */
ZAid = HE5_ZAcreate(zafid, "ZA1");

/* Define dimensions and specify their sizes */

status = HE5_ZAdefdim(ZAid, "MyTrack1", 20);
status = HE5_ZAdefdim(ZAid, "MyTrack2", 10);
status = HE5_ZAdefdim(ZAid, "Res2tr", 40);
status = HE5_ZAdefdim(ZAid, "Res2xtr", 20);
status = HE5_ZAdefdim(ZAid, "Bands", 15);

/* Define "External" Dimension */

status = HE5_ZAdefdim(ZAid, "ExtDim", 60);

/* Define "Unlimited" Dimension */

status = HE5_ZAdefdim(ZAid, "Unlim", H5S_UNLIMITED);

/* Close the ZA interface */
status = HE5_ZAdetach(ZAid);

/* Close the ZA file */
status = HE5_ZAclose(zafid);
Appendix A. Installation and Maintenance

A.1 Installation Procedures

A.1.1 Preliminary Step

Before installing HDFEOS, you must already have installed The HDF Group HDF, Version 5-1.8.8 on your host. You may also need to install jpeg-v6b, zlib-1.2.5 and szip-2.1 before HDF5 installation. The installation script will prompt for the paths to the HDF include and library directories. Please see the SDP Toolkit Users Guide for the EOSDIS Evolution and Development Contract, Section 5 for instructions on installing both the Toolkit and HDF. See also http://hdf.ncsa.uiuc.edu/ for instructions on how to access HDF libraries.

A.1.2 Unpacking the Distribution File

1) Select a location for the HDFEOS directory tree. Installing HDFEOS alone requires a disk partition with at least 35 Mb of free space.

2) Copy the file HDF-EOSv5.1.14.tar.Z to the target directory by typing the command:

   cp HDF-EOSv5.1.14.tar.Z <target-dir>

   where <target-dir> is the full pathname of your target directory.

3) Set your default directory to the target directory by typing the command:

   cd <target-dir>

4) Uncompress this file and extract the contents by typing the command:

   zcat HDF-EOS5.1.14.tar.Z | tar xvf -

This will create a subdirectory of the current directory called 'hdfeos5'. This is the top-level HDFEOS directory, which contains the full HDFEOS directory structure.

A.1.3 Starting the Installation Procedure

You may install hdfeos5 using the installation script (Section A.1.3.1) or using the built in autoconfiguration/automake similar to HDF5. The latter method is outlined in section A.1.3.2.

A.1.3.1 Installation Using Installation Scripts

1) Set your default directory to the top-level HDFEOS directory by typing the command:

   cd hdfeos5

2) Select installation options.
Now the library supports the option to build a thread-safe version:

```
<install-option> purpose
  -ts              to build a thread-safe version
  -ts_dbg          thread-safe version with enabled debug statements
```

Also, the user should specify the architecture options for the SGI Power Challenge platform. On the SGI Challenge, the default is to build HDFEOS in 64-bit mode, which is the same as the Toolkit. The following table gives the option to specify the appropriate architecture to be built:

```
<table>
<thead>
<tr>
<th>binary format</th>
<th>architecture</th>
<th>&lt;install-option&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>new 32-bit</td>
<td>sgi32</td>
<td>-sgi32</td>
</tr>
<tr>
<td>64 bit</td>
<td>sgi64</td>
<td>-sgi64</td>
</tr>
</tbody>
</table>
```

Please note that the old-32-bit mode has been dropped as the default because it is no longer being supported by SGI, it is therefore recommended that all users migrate to new-style 32 bit or 64 bit mode.

3) Run the installation script.

Please note that the installation script for this release of HDFEOS requires user interaction. Because of this, it should NOT be run as a background task.

3.0) If you wish to generate a log of this session, use the Unix 'script' command. This command runs a sub-shell that saves all terminal output to the specified file. To log the session, type:

```
script <logfile-name>
```

where `<logfile-name>` is the name of the log file

3.1) To run the installation script, type the command:

```
bin/INSTALL-HDFEOS <install-options>
```

where `<install-options>` is the list of options determined in the previous step.

The installation script will then run. It will output various startup messages, beginning with:

```
HDFEOS installation starting at <date/time>
```

3.2) Enter the full pathnames for the hdf5-1.8.8 library and include directory paths, when the script prompts for them. If there is an error in the supplied paths, the script will exit.

NOTE: If the environment variables HDFLIB and/or HDFINC are set in your shell, the script will use these for the default values. If this is not the first run of the script, the default values will be taken from the values used for the last run of the script. In either of these cases, the installation script will prompt with:

```
Current value of the HDF library directory is: <path>
Accept [y]/n:
```
A.1.3.2 Installation Using Autoconf/Automake

1) Quick Start (here we assume that the brand is linux. You may replace it with another supported brand name if you are installing hfeos5 in a different platform)

To build HDF-EOS5 from <target-dir> /hdfeos5 and install the HDF-EOS5 library into <target-dir> /hdfeos5/lib/linux:

$ cd <target-dir> /hdfeos5
$ ./configure --with-hdf5=/path/to/hdf5 --libdir=<target-dir> /hdfeos5/lib/linux
$ make install

2) Configuration

HDF-EOS5 uses the GNU autoconf system for configuration, which detects various features of the host system and creates the Makefiles. On most systems with HDF-EOS5 installed it should be sufficient to say:

$ ./configure
$ sh configure

The configuration process can be controlled through environment variables and command-line switches. For a complete list of switches type:

$ ./configure --help
Configure must be re-run for each platform, and the source tree can only be configured for one platform at a time.

In 64-bit machines one must set the desired compiler flag for the compilation mode before running ./configure. For example to install HDF-EOS5 in 32-bit mode in SGI one must set CC as:

```
setenv CC "cc -n32"
```

or to install HDF-EOS5 in 64-bit mode in SGI one must set CC as:

```
setenv CC "cc -64"
```

In a 64-bit linux the 32-bit flag is “-m32”. So for 32-bit installation one must set CC as:

```
setenv CC "gcc -m32"
```

The same is applied to the FORTRAN compiler flag.

3) Building against HDF5

On systems without HDF5 installed, where HDF5 is not found automatically, or to link against a different version of the HDF5 library, the user must specify the path to HDF5. This can be done either by giving the path to configure directly:

```
$ ./configure --with-hdf5=/path/to/hdf5
```

or by setting the environment variable CC to be the h5cc script installed with HDF5:

```
$ H5CC=/path/to/hdf5/bin/h5cc
$ ./configure
```

4) Building against ZLIB and SZLIB

HDF-EOS5 does not require the zlib and szlib libraries to build, but some of the tests in the testdrivers directory require them. Their paths can be given to configure using the --with-zlib and --with-szlib switches:

```
$ ./configure --with-zlib=/usr/local/zlib --with-szlib=/usr/local/szlib
```

5) Specifying install locations

The location where the HDF-EOS5 library will be installed is controlled by
the --libdir switch. To set the install location to
<target-dir>/hdfeos5/lib/linux:

$ ./configure --libdir=<target-dir>/hdfeos5/lib/linux

HDF-EOS5 traditionally installs libraries into the hdfeos5/lib/* directories
and does not install header files. Users who wish to install both libraries
and header files should use the --enable-install-include switch to enable
this feature and the --prefix switch to control where they are installed.
To install into /usr/local/hdfeos5/include and /usr/local/hdfeos5/lib:

$ ./configure --enable-install-include --prefix=/usr/local/hdfeos5

The default installation location if no flags are specified is a directory
named hdfeos5 in the current directory, with libraries in hdfeos5/lib and
include files in hdfeos5/include (if installing include files is enabled).

6) Building and Installing

Once HDF-EOS5 has been configured, its makefiles can be used to build, test,
and install. To build the library:

$ make

To run tests (if present):

$ make check

To install to the location specified during configure:

$ make install

These commands do not need to be run in order; if the library has not been built, 'make install'
will build it before installing. However, configure must always have been run on the current
system before running make.

7) The Testdrivers Directory

The testdrivers directory contains test to verify that HDF-EOS5 has built correctly. Users who
wish to run these tests should copy the testdrivers directory into the hdfeos5 directory before
running configure. Configure will detect the presence of this directory and 'make check' will run
all the tests it contains.
8) For More Information

For more information about using autoconf and automake, see the documentation online at http://sources.redhat.com/autobook/autobook/autobook.html or HDF5's documentation.

A.1.4 User Account Setup

Once HDFEOS has been installed, the accounts of HDFEOS users must be set up to define environment variables needed to compile and run code with HDFEOS (see parts 2 and 3 of the Notes section, below). The type of setup depends on the user's login shell.

1A) C shell (csh) Users:

Edit the HDFEOS user's .cshrc file to include the following line:

```
source <HDFEOS-home-dir>/bin/$BRAND/hdfeos_env.csh
```

where <HDFEOS-home-dir> is the full path of the HDFEOS home directory, and $BRAND is an architecture-specific value for your host. Please refer to part 2 of the Notes section, below, to determine the correct value.

The script hdfeos_env.csh sets up all the variables discussed in parts 2 and 3 of the Notes section, below, and it adds the HDFEOS bin directory to the user path.

The environment variables will become available during all subsequent login sessions. To activate them for the current session, simply type one of the two lines listed above, at the Unix prompt.

Note regarding path setup with hdfeos_env.csh:

The script hdfeos_env.csh also makes available a variable called hdfeos_path. This can be added to the user's path to ensure that it accesses the directories necessary for the compilers and other utilities used to generate executable programs. It is not added to the user path by default, because in many cases it adds unnecessary complexity to the user path. To add hdfeos_path to the user path, modify the HDFEOS user's .cshrc file to include the following:

```
set my_path = ($path)
source <HDFEOS-HOME-DIR>/bin/$BRAND/hdfeos_env.csh # HDFEOS setup
set path = ($my_path $hdfeos_path)
```

INSTEAD OF the line listed at the beginning of this step.

Note that it is the user's responsibility to set up his or her own path so that it doesn't duplicate the directories set up in hdfeos_path. Please also note that the hdfeos_path is added AFTER the user's path. This way, the user's directories will be searched first when running Unix commands.
1B) Korn shell (ksh) Users:

Edit the HDFEOS user's .profile file to include the following line:

```bash
. <HDFEOS-home-dir>/bin/$BRAND/hdfeos_env.ksh
```

where <HDFEOS-home-dir> is the full path of the HDFEOS home directory, and $BRAND is an architecture-specific value for your host. Please refer to part 2 of the Notes section, below, to determine the correct value.

The script hdfeos_env.ksh sets up all the variables discussed in part 2 and 3 of the Notes section, below, and it adds the HDFEOS bin directory to the user path.

The environment variables will become available during all subsequent login sessions. To activate them for the current session, simply type one of the two lines listed above, at the Unix prompt.

Note regarding path setup with hdfeos_env.ksh:

The script hdfeos_env.ksh also makes available a variable called hdfeos_path. This can be added to the user's path to ensure that it accesses the directories necessary for the compilers and other utilities used to generate executable programs. It is not added to the user path by default, because in many cases it adds unnecessary complexity to the user path. To add hdfeos_path to the user path, modify the HDFEOS user's .profile file to include the following:

```bash
my_path="$PATH"
# save path
. <HDFEOS-HOME-DIR>/bin/$BRAND/hdfeos_env.ksh # HDFEOS setup
PATH="$my_path:$hdfeos_path" ; export PATH # add hdfeos_path
```

INSTEAD OF the line listed at the beginning of this step.

Note that it is the user's responsibility to set up his or her own path so that it doesn't duplicate the directories set up in hdfeos_path. Please also note that the hdfeos_path is added AFTER the user's path. This way, the user's directories will be searched first when running Unix commands.

1C) Bourne shell (sh) Users:

Set up the required HDFEOS environment variables by appending the contents of the file <HDFEOS-home-dir>/bin/$BRAND/hdfeos_env.ksh to the end of the HDFEOS user's .profile, where <HDFEOS-home-dir> is the full path of the HDFEOS home directory, and $BRAND is an architecture-specific value for your host. Please refer to part 2 of the Notes section, below, to determine the correct value.

The environment variables will become available during all subsequent login sessions. To activate them, log out and then log back in.

A.1.5 File Cleanup

Once HDFEOS has been built and tested, you can delete certain temporary files and directories to save some disk space. Note that once these files have been removed, you will need to unpack the original distribution file in order to re-do the installation.
To remove these files:
```
cd <HDFEOS-home-dir>/bin
rm -rf tmp
cd <HDFEOS-home-dir>/lib
rm -rf tmp
```

A.1.6 Compiling and Linking with HDFEOS5

In order to compile and link programs with the HDFEOS5 you must access the HDFEOS5 include and library files. To do this be sure that your makefiles include something like the following:

```
INCLUDE = -I -I$(HDFEOS5_INC) -I$(HDF5INC) -I$(JPEGINC) -I$(ZLIBINC)
           -I$(SZIPINC)
LIBRARY = -L -L$(HDFEOS5_LIB) -L$(HDF5LIB) -L$(JPEGLIB) -L$(ZLIBLIB)
           -L$(SZIPLIB)
LDFLAGS = -lhe5_hdfeos -lGctp -lhdf5 -lhdf5_hl -lnsl -lz -ljpeg -lsz -lpthread -lm (Sun platform)
           LDFLAGS = -lhe5_hdfeos -lGctp -lhdf5 -lhdf5_hl -lz -ljpeg -lsz -lpthread -lm (others)
```

The environment variables HDFEOS5_INC, HDFEOS5_LIB, HDF5INC and HDF5LIB are set up by the HDFEOS5 environment scripts (see User Setup, above). They refer to the include and library directories for HDFEOS5 and HDF5, respectively.

The INCLUDE macro should be included in all compilation statements. The LIBRARY an LDFLAGS macros should be included in all link statements.

The flag -D_HDFEOS5_THREADSAFE should be included in your makefile for HDF-EOS5 thread-safe version.

A.2 Notes

1) Approved Platforms

HDFEOS5 was built and tested in a multi-platform environment. The list of approved platforms, which includes information about operating system and compiler versions, may be found in the HDFEOS5 User's Guide and is also listed below.
<table>
<thead>
<tr>
<th>Platform</th>
<th>OS</th>
<th>Version</th>
<th>C Compiler</th>
<th>FORTRAN77</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun Sparc</td>
<td>Solaris</td>
<td>5.10</td>
<td>Sun C 5.7</td>
<td>Sun FORTRAN 8.1 (F95)</td>
</tr>
<tr>
<td>Power Mac OS X</td>
<td>Darwin</td>
<td>8.10.0</td>
<td>gcc 4.0.1</td>
<td>gfortran 4.3.0</td>
</tr>
<tr>
<td>Intel Mac (32-bit)</td>
<td>Darwin</td>
<td>9.8.0</td>
<td>gcc 4.0.1</td>
<td>gfortran 4.3.0</td>
</tr>
<tr>
<td>Intel Mac (64-bit)</td>
<td>Darwin</td>
<td>9.8.0</td>
<td>gcc 4.0.1</td>
<td>gfortran 4.3.0</td>
</tr>
<tr>
<td>SGI Power Challenge</td>
<td>IRIX</td>
<td>6.5.9</td>
<td>SGI C 7.4.2m</td>
<td>SGI FORTRAN 7.4.2m</td>
</tr>
<tr>
<td>Linux 32-bit</td>
<td>Red Hat Linux</td>
<td>2.6.18-128.1.1.e15 #1 SMP x86_64</td>
<td>gcc 4.1.2-42</td>
<td>g77 3.4.6-4</td>
</tr>
<tr>
<td>Linux 64-bit</td>
<td>Red Hat Linux</td>
<td>2.6.18-128.1.1.e15 #1 SMP x86_64</td>
<td>gcc 4.1.2-42</td>
<td>g77 3.4.6-4</td>
</tr>
<tr>
<td>Linux 64-bit (Itanium)</td>
<td>ia64 SUSE Linux</td>
<td>2.6.16.60-0.66.1- default #1 SMP</td>
<td>gcc 4.1.2</td>
<td>ifort 10.0</td>
</tr>
<tr>
<td>Windows</td>
<td>XP</td>
<td>MS VS .net 2008</td>
<td>Visual c++</td>
<td>Intel Visual Fortran 11.1</td>
</tr>
<tr>
<td>Cygwin</td>
<td>Cygwin</td>
<td>1.5.19</td>
<td>gcc 3.4.4</td>
<td>g77 3.4.4</td>
</tr>
</tbody>
</table>

Note: The compilers are supplied by the vendor. The SGI Power Challenge (64-bit mode) is not supported by HDF. Therefore it cannot be supported for HDF-EOS.

2) Architecture Type Names

To track architecture dependencies, HDFEOS defines the environment variable $BRAND. Following is a list of valid values for this variable, which is referred to throughout this document(though some platforms such as dec, hp, hp11, ibm, sun5.8, and sgi old-style 32-bit mode are not supported with this or some old versions of HDF-EOS),:

```
$BRAND                      Architecture
dec                         DEC Alpha
ibm                         IBM AIX
hp                          HP 9000
hp                          HP-UX11 9000/785
sgi                         SGI Power Challenge (old-style 32-bit mode)
sgi32                       SGI Power Challenge (new-style 32-bit mode)
sgi64                       SGI Power Challenge (64-bit mode)
sun5.8, sun5.9, sun5.10     Sun:SunOS 5.8, OS5.9, OS5.10
linux                       LINUX Platforms
linux32                     64-bit LINUX Platforms for 32-bit mode
linux64                     64-bit LINUX Platforms for 64-bit mode
macintel (32-bit and 64-bit) Macintosh platforms with Intel chip
macintosh                   Macintosh Power PC
cygwin                      Cygwin 32-bit Platform
```

A-9 175-EED-001, Revision 01
3) Directory and File Environment Variables

In order to use the HDFEOS5 library and utilities, a number of environment variables MUST be set up to point to HDFEOS5 directories and files. These variables are automatically set up in the User Account Setup section of the installation instructions. They are listed here for reference:

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFEOS5_HOME</td>
<td>&lt;install-path&gt;/hdfeos5</td>
<td>top-level directory</td>
</tr>
<tr>
<td></td>
<td>(where &lt;install-path&gt; is the absolute directory path above hdfeos5)</td>
<td></td>
</tr>
<tr>
<td>HDFEOS5_BIN</td>
<td>$HDFEOS5_HOME/bin/$BRAND</td>
<td>executable files</td>
</tr>
<tr>
<td>HDFEOS5_INC</td>
<td>$HDFEOS5_HOME/include</td>
<td>header files</td>
</tr>
<tr>
<td>HDFEOS5_LIB</td>
<td>$HDFEOS5_HOME/lib/$BRAND</td>
<td>library files</td>
</tr>
<tr>
<td>HDFEOS5_OBJ</td>
<td>$HDFEOS5_HOME/obj/$BRAND</td>
<td>object files</td>
</tr>
<tr>
<td>HDFEOS5_SRC</td>
<td>$HDFEOS5_HOME/src</td>
<td>source files</td>
</tr>
</tbody>
</table>

4) Other Environment Variables

In addition, the makefiles which are used to build the library require certain machine-specific environment variables. These set compilers, compilation flags and libraries, allowing a single set of makefiles to serve on multiple platforms. The User Account Setup section of the installation instructions explains how to set them up. They are listed here for reference:

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>C compiler</td>
</tr>
<tr>
<td>CFLAGS</td>
<td>default C flags (optimize, ANSI)</td>
</tr>
<tr>
<td>C_CFH</td>
<td>C w/ cfortran.h callable from FORTRAN</td>
</tr>
<tr>
<td>CFHFLAGS</td>
<td>CFLAGS + C CfH</td>
</tr>
<tr>
<td>C_F77_CFH</td>
<td>C w/ cfortran.h calling FORTRAN</td>
</tr>
<tr>
<td>C_F77_LIB</td>
<td>FORTRAN lib called by C main</td>
</tr>
<tr>
<td>F77</td>
<td>FORTRAN compiler</td>
</tr>
<tr>
<td>F77FLAGS</td>
<td>common FORTRAN flags</td>
</tr>
<tr>
<td>F77_CFH</td>
<td>FORTRAN callable from C w/ cfortran.h</td>
</tr>
<tr>
<td>F77_C_CFH</td>
<td>FORTRAN calling C w/ cfortran.h</td>
</tr>
<tr>
<td>CFH_F77</td>
<td>same as F77_C_CFH</td>
</tr>
<tr>
<td>F77_C_LIB</td>
<td>C lib called by FORTRAN main</td>
</tr>
</tbody>
</table>

5) Tools Provided with This Release

For a complete list of the tools provided with this release of HDFEOS5, please refer to Section 7 of this document.

6) Copyright Notice for cfortran.h

HDFEOS functions are written in C. These C-based tools include the file cfortran.h, using it to generate machine-independent FORTRAN bindings. The cfortran.h facility includes the following notice which must accompany distributions that use it:
A.3 Test Drivers

Also included with this software delivery is a tar file containing test driver programs. These test programs are provided to aid the user in the development of software using the HDF-EOS5 library. The user may run the same test cases as included in this file to verify that the software is functioning correctly. These programs were written to support the internal testing and are not an official part of the delivery. Users make use of them at their own risk. No support will be provided to the user of these programs. The tar file contains source code for a drivers in C and FORTRAN for each tool; sample output files; and input files and/or shell scripts, where applicable.

The UNIX command: “zcat HDF-EOS5.1.14_TestDrivers.tar.Z | tar xvf -” will create a directory called test_drivers beneath the current directory containing all these test files.
A.4 User Feedback Mechanism

The mechanism for handling user feedback, documentation and software discrepancies, and bug reports follows:

1) The following accounts at the ECSRiverdale facility have been set up for user response:
   
   - Landover_PGSTLKIT@raytheon.com and

2) Users will e-mail problem reports and comments to the above account. Responses will be prioritized based on the severity of the problem and the available resources. Simple bug fixes will be turned around sooner, while requested functional enhancements to the Toolkit will be placed in a recommended requirements database (RRDB) and handled more formally.

3) In order to help expedite responses, we request the following information be supplied with problem reports:
   
   - Name:
   - Date:
   - EOS Affiliation (DAAC, Instrument, Earth Science Data and Information System (ESDIS), etc.):
   - Phone No.:
   - Development Environment:
   - Computing Platform:
   - Operating System:
   - Compiler and Compiler Flags:
   - Tool Name:
   - Problem Description:
   
   (Please include exact inputs to and outputs from the toolkit call, including error code returned by the function, plus exact error message returned where applicable.)

   Suggested Resolution (include code fixes or workarounds if applicable):

3) In addition to the email response mechanism, a phone answering machine is also provided. The telephone number is: 301–851–8373. Calls will be returned as soon as possible. We note, however, that email is our preferred method of responding to users.
# 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>EOSPM</td>
<td>EOS PM Project (afternoon spacecraft series)</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</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>NGDC</td>
<td>National Geophysical Data Center</td>
</tr>
<tr>
<td>NMC</td>
<td>National Meteorological Center (NOAA)</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</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>SDF</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>THG</td>
<td>The HDF Group</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>
<tr>
<td>URL</td>
<td>universal reference locator</td>
</tr>
<tr>
<td>USNO</td>
<td>United States Naval Observatory</td>
</tr>
</tbody>
</table>
UT  universal time
UTC  Coordinated Universal Time
UTCF  universal time correlation factor
UTM  universal transverse mercator
VPF  vector product format
WWW  World Wide Web
ZA  Zonal Average