EED2-175-001

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

Technical Paper

December 2017

Prepared Under Contract #NNG15HZ39C

RESPONSIBLE ENGINEER

E. Moghaddam Taaheri 12/21/17

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

SUBMITTED BY

Kristene Rodrigues 12/21/17

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

Raytheon Company
Riverdale, Maryland

This page intentionally left blank.

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 5.1.16. 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 V5.1.16 incorporates HDF5, and keeps the familiar HDF4-based interface. There are a few exceptions and these exceptions are described in this document. Note that the major functional difference is that Version 5.1.16 of the HDF-EOS library is a thread-safe.

HDF is the scientific data format standard selected by NASA as the baseline standard for EOS. This Users Guide accompanies Version 5.1.16 software, which is available to the user community on the EDHS1 server. This library is aimed at EOS data producers and consumers, who will develop their data into increasingly higher order products. These products range from calibrated Level 1 to Level 4 model data. The primary use of the HDF-EOS library will be to create structures for associating geolocation data with their associated science data. This association is specified by producers through use of the supplied library. Most EOS data products which have been identified, fall into categories of point, grid, swath or zonal average structures, the latter two of which are implemented in the current version of the library. Services based on geolocation information will be built on HDF-EOS structures. Producers of products not covered by these structures, e.g. non-geolocated data, can use the standard HDF libraries.

In the ECS (EOS Core System) production system, the HDF-EOS library will be used in conjunction with SDP (Science Data Processing) Toolkit software. The primary tools used in conjunction with HDF-EOS library will be those for metadata handling, process control and status message handling. Metadata tools will be used to write ECS inventory and granule specific metadata into HDF-EOS files, while the process control tools will be used to access physical file handles used by the HDF tools (SDP Toolkit Users Guide for the EOSDIS Evolution and Development-2 (EED-2) Contract, December 2017, 333-EED2-001, Revision 01.)

HDF-EOS is an extension of The HDF Group (THG) HDF5 and uses HDF5 library calls as an underlying basis. Version 5-1.8.19 of HDF5 is used. The library tools are written in the C language and a FORTRAN interface is provided. The current version contains software for creating, accessing and manipulating Grid, Point, Swath and Zonal Average structures. This document includes overviews of the interfaces, and code examples. The HDF-EOS plug-in for HDFView, the HDF-EOS viewing 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:

RVL_PGSTLKIT@raytheon.com

Any questions should be addressed to:

Data Management Office
EOSDIS Evolution and Development-2 (EED-2) Contract
Raytheon Company
5700 Rivertech Court
Riverdale, MD 20737

Abstract

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

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

This page intentionally left blank.

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 V2.20 (HDF4 based) and HDF-EOS V5.1.16 (HDF5 based) 3-3

3.6 Different Types of Attributes in HDF-EOS5 3-6

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

5.3.1 Swath IDentifiers 5-8

5.4 Programming Model 5-9

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 6-6
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 7-93
7.4.1 Creating a Simple Zonal Average 7-93

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 8-31
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 9-4

9.3.3 Zonal Average Identifiers 9-4

9.4 Programming Model 9-4

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 5.1.16 Current Modifications to the HDF4-Based HDF-EOS Library 2.20 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

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-2 (EED-2) Contract (NNG15HZ39C).

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:

EED2-175-001	HDF-EOS Interface Based on HDF5, Volume 1: Overview and Examples
EED2-175- 002	HDF-EOS Interface Based on HDF5, Volume 2: Function Reference Guide
EED2-170- 001	The HDF-EOS Library Users Guide for the EOSDIS Evolution and Development (EED) Contract, Volume 1: Overview and Examples
EED2-170- 002	The HDF-EOS Library Users Guide for the EOSDIS Evolution and Development (EED) Contract, 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.

EED2-333- 001	SDP Toolkit Users Guide for the EOSDIS Evolution and Development-2 (EED-2) Contract
163-WP-001	An ECS Data Provider's Guide to Metadata in Release A, A White Paper for the ECS Project
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	HDF5: API Specification Reference Manual, October, 1999

none	Getting Started with HDF, Version 3.2, University of Illinois, May 1993
none	NCSA HDF Reference Manual, Version 3.3, University of Illinois, February 1994
none	NCSA HDF Specification and Developer's Guide, Version 3.2, University of Illinois, September 1993
none	NCSA HDF User's Guide, Version 4.0, University of Illinois, February 1996
none	NCSA HDF User's Guide, Version 3.3, University of Illinois, March 1994
none	An Album of Map Projections, USGS Professional Paper 1453, Snyder and Voxland, 1989
none	Map Projections - A Working Manual, USGS Professional Paper 1395, Snyder, 1987
none	The WMO Format for the Storage of Weather Product Information and the Exchange of Weather Product Messages in Gridded Binary Form, John D. Stackpole, Office Note 388, GRIB Edition 1, U.S. Dept. of Commerce, NOAA, National Weather Service National Meteorological Center, Automation Division, Section 1, pp. 9-12, July 1, 1994.
none	The Michigan Earth Grid: Description, Registration Method for SSM/I Data, and Derivative Map Projection, John F. Galntowicz, Anthony W. England, The University of Michigan, Radiation Laboratory, Ann Arbor, Michigan, Feb. 1991.
none	Selection of a Map Grid for Data Analysis and Archive, William B. Rossow, and Leonid Garder, American Meteorological Society Notes, pp. 1253-1257, Aug. 1984.
none	Level-3 SeaWiFS Data Products: Spatial and Temporal Binning Algorithms, Janet W. Campbell, John M. Blaisdell, and Michael Darzi, NASA Technical Memorandum 104566, GSFC, Volume 32, Appendix A, Jan. 13, 1995.

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 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, December 2017, 333-EED2-001, 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 5.1.16. The former library is currently designated HDF-EOS 2.20. (The HDF-EOS Library Users Guide for the EED2 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 5.1.16 support for HDF5 will have the same function calls as the current HDF-EOS 2.20 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 V2.20 (HDF4 based) and HDF-EOS V5.1.16 (HDF5 based)

There are several important differences between the Versions 2.20 and 5.1.16 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 3-1. HDF-EOS 5.1.16 Current Modifications to the HDF4-Based HDF-EOS Library 2.20

Function	Change Description
Point Access	Modify to implement HDF5 Compound data type.
Point Definition	Modify to implement HDF5 Compound data type.
Point Input/output	Add an additional parameter, ‘size’, a dataset size. Add an additional parameter – data structure in HE5_Ptreadlevel() to pass information about data fields to read.
Swath Access	Modify to implement new HDF5 file structure. Create a swath with a specified name
Swath Definition	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())
Swath Inquiry	Add a new function HE5_SWchunkinfo() to retrieve chunking information. Add two new functions HE5_SWinqdfldalias() and HE5_SWinqgfldalias() to retrieve information about data or geolocation fields and aliases in swath
Swath Input/output	Add unsigned long type to store large number of elements in attributes. Change 'start,stride,edge' parameter type to allow for very large numbers.
Swath Subset	Add a new function HE5_SWindexinfo() to retrieve the indices information about a subsetted region.
New interfaces	HE5_SWinqdatatype() retrieve information about data type of a data field HE5_PRdefine(),HE5_PRread(),HE5_PRwrite(), HE5_PRinquire(), HE5_PRinfo(), HE5_PRreclaimspace()- support profile structures. HE5_SWsetextdata(), HE5_SWgetextdata() – support external data files.
Grid Access	Modify to implement new HDF5 file structure. Create a grid with a specified name
Grid Definition	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_GDdefcomp() and HE5_GDdefcomtile().
New interfaces	HE5_GDsetextdata(), HE5_GDgetextdata() – support external data files.
Grid Input/Output	Add unsigned long type to store large number of elements in attributes.
Grid Inquiry	Add a new function HE5_GDinqfldalias() to retrieve information about data fields and aliases in grid.
Grid Subset	No new functionality or interface changes
Grid Tiling	Add a new function HE5_GDtileinfo() to retrieve tiling information for fields.
Utilities	Added a function for determining whether a file is HDF4 or HDF5 based. Added checking for illegal characters (“,” “;” “/”) in the HDF-EOS object names. Added interfaces for the global “File Attributes”: HE5_EHwriteglbattr(), HE5_EHreadglbattr(), HE5_EHglbattrinfo(), HE5_EHinqglbattrs(),HE5_EHinqglbdatatype().
HE5View	Modified EOS View to accommodate new HDF5-based HDF-EOS library.

Version 5.1.16 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 V5.1.16 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_EHinqglbdatatype(), 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_SWwritegeogrpattr(), HE5_SWreadgeogrpattr(), HE5_SWgeogrpattrinfo(), and HE5_SWinqgeogrpattrs() 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 "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" {
```

```
      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 { ( 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),
          (2000, 2001, 2002, 2003, 2004, 2005, 2006, ........,
           ........., 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:

```
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 *)mallloc((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.

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

Figure 4-1. A Simple Point Data Example

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.

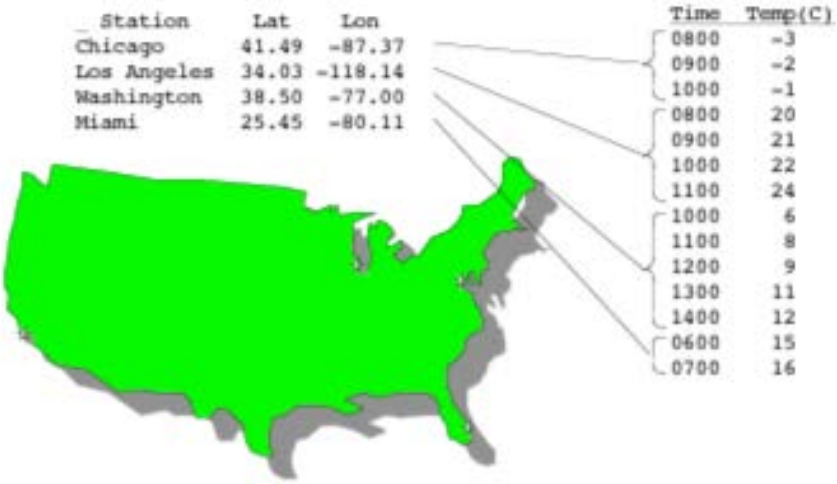

Figure 4-2. Recording Points Over Time

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.

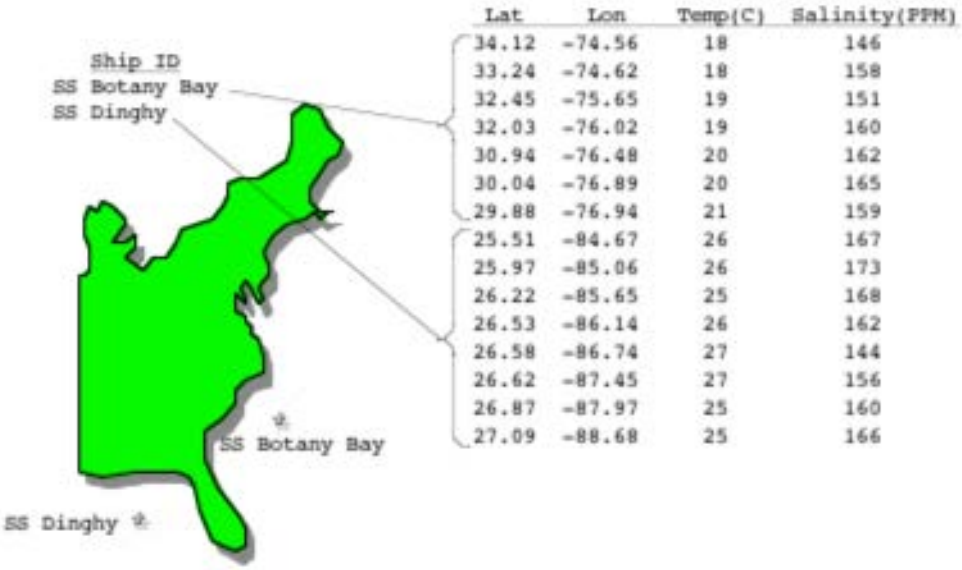

Figure 4-3. Point Data from a Moving Platform

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 4-1. Summary of the Point Interface (1 of 2)

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Access	HE5_PTopen	he5_ptopen	Creates a new file or opens an existing one	2-32
	HE5_PTcreate	he5_ptcreate	Creates a new point data set and returns a handle	2-06
	HE5_PTattach	he5_ptattach	Attaches to an existing point data set	2-02
	HE5_PTdetach	he5_ptdetach	Releases a point data set and frees memory	2-14
	HE5_PTclose	he5_ptclose	Closes the HDF-EOS file and deactivates the point interface	2-05
Definition	HE5_PTdeflevel	he5_ptdeflevel	Defines a level within the point data set	2-07
	HE5_PTdeflinkage	he5_ptdeflinkage	Defines link field to use between two levels	2-13
Basic I/O	HE5_PTwritelevel	he5_ptwritelevel	Writes (appends) full records to a level	2-42
	HE5_PTreadlevel	he5_ptreadlevel	Reads data from the specified fields and records of a level	2-36
	HE5_PTupdatelevel	he5_ptupdatelevel	Updates the specified fields and records of a level	2-37
	HE5_PTwriteattr	he5_ptwriteattr	Creates or updates an attribute of the point data set	2-38
	HE5_PTwritegrpattr	he5_ptwritegrpattr	Creates or updates group attribute	2-40
	HE5_PTwritelocattr	he5_ptwritelocattr	Creates or updates local attribute	2-43
	HE5_PTreadattr	he5_ptreadattr	Reads existing attribute of point data set	2-33
Inquiry	HE5_PTreadgrpattr	he5_ptreadgrpattr	Reads group attribute	2-34
	HE5_PTreadlocattr	he5_ptreadlocattr	Reads local attribute	2-35
	HE5_PTnlevels	he5_ptnlevels	Returns the number of levels in a point data set	2-30
	HE5_PTnrecs	he5_ptnrecs	Returns the number of records in a level	2-31
	HE5_PTnfields	he5_ptnfields	Returns number of fields defined in a level	2-29
	HE5_PTlevelinfo	he5_ptlevelinfo	Returns information about a given level	2-27
	HE5_PTlevelindx	he5_ptlevelindx	Returns index number for a named level	2-26
	HE5_PTbcklinkinfo	he5_ptbcklinkinfo	Returns link field to previous level	2-04
	HE5_PTfwdlinkinfo	he5_ptfwdlinkinfo	Returns link field to following level	2-15
	HE5_PTgetlevelname	he5_ptgetlevelname	Returns level name given level number	2-16
	HE5_PTgetrecnums	None	Retrieves number of records in one level corresponding to a group of records in a different level	2-17
	HE5_PTattrinfo	he5_ptattrinfo	Returns information about point attributes	2-03
	HE5_PTattrinfo2	he5_ptattrinfo2	Same as above, but also returns buffer size of attribute element.	2-03
	HE5_PTgrpattrinfo	he5_ptgrpattrinfo	Returns information about group attribute	2-18
	HE5_PTgrpattrinfo2	he5_ptgrpattrinfo2	Same as above, but also returns buffer size of attribute element.	2-18
	HE5_PTlocattrinfo	he5_ptlocattrinfo	Returns information about local (level) attribute	2-28
	HE5_PTlocattrinfo2	he5_ptlocattrinfo2	Same as above, but also returns buffer size of attribute element.	2-28
	HE5_PTinqattrs	he5_ptinqattrs	Retrieves number and names of attributes defined	2-19

Table 4-1 Summary of the Point Interface (2 of 2)

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Inquiry	HE5_PTinqgrpattrs	he5_ptinqgrpattrs	Retrieves information about group attributes	2-22
	HE5_PTinqlocattrs	he5_ptinqlocattrs	Retrieves information about local (level) attributes	2-23
	HE5_PTinqpoint	he5_ptinqpoint	Retrieves number and names of points in file	2-25
	HE5_PTinqdatatype	he5_ptinqdatatype	Returns data type information about specified level in point	2-20

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.

```
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_PTopen(filename_1, access_mode);
ptid_1 = HE5_PTattach(ptfid_1, point_name_1);
ptfid_2 = HE5_PTopen(filename_2, access_mode);
ptid_2 = HE5_PTattach(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.

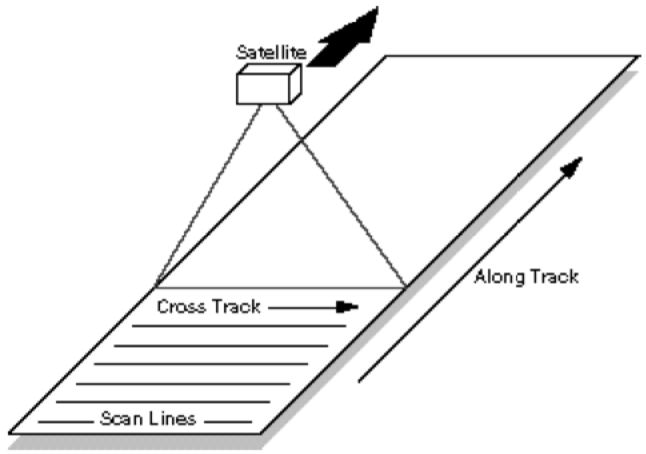

Figure 5-1. A Typical Satellite Swath: Scanning Instrument

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.

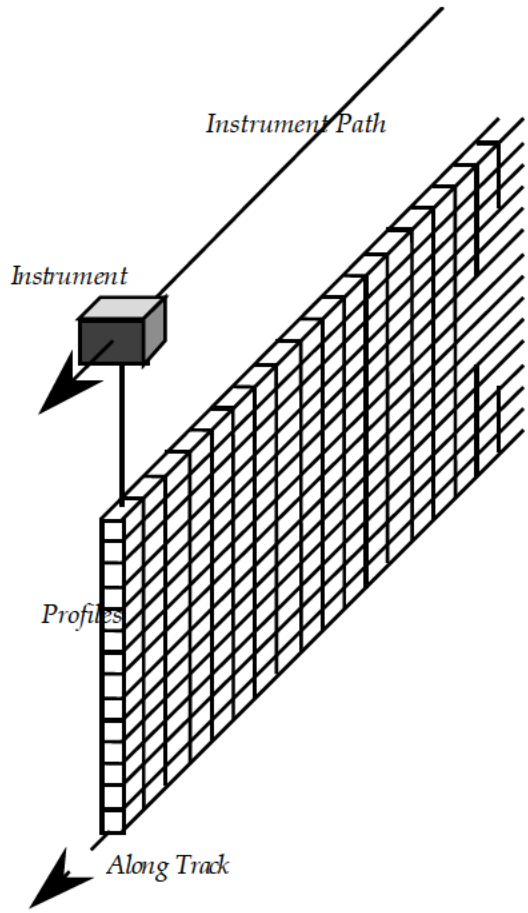

Figure 5-2. A Swath Derived from a Profiling Instrument

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"[1] 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 on 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."

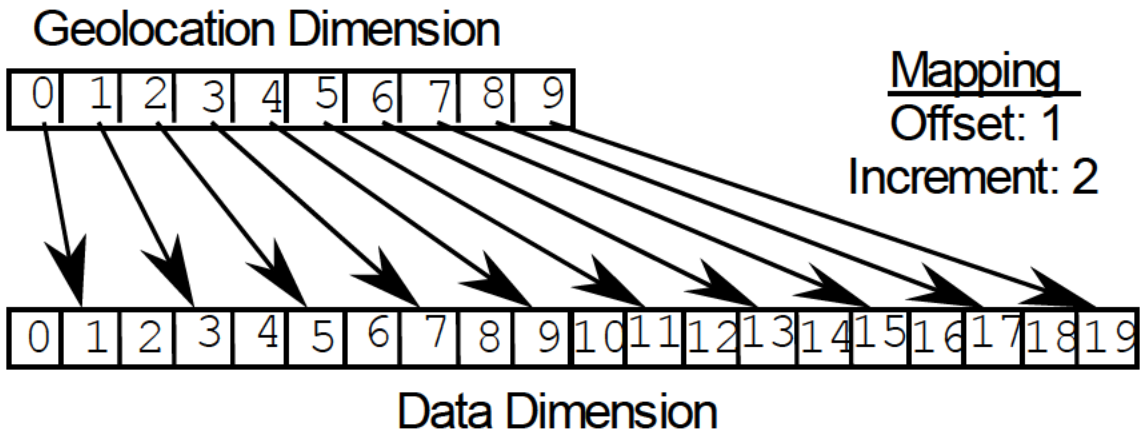

Figure 5-3. A "Normal" Dimension Map

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.

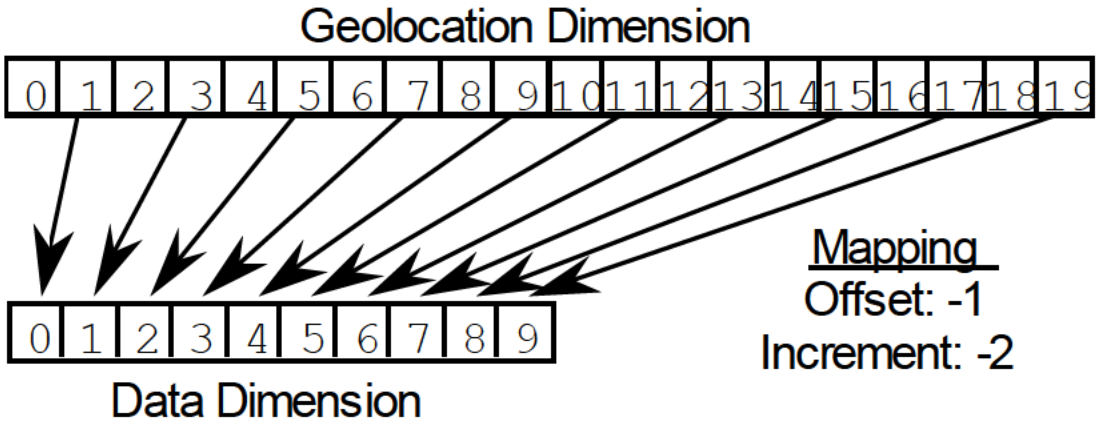

Figure 5-4. A "Backwards" Dimension Map

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 5-1. Summary of the Swath Interface (1 of 3)

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Access	HE5_SWopen	he5_swopen	Opens or creates HDF file in order to create, read, or write a swath	2-123
	HE5_SWcreate	he5_swcreate	Creates a swath within the file	2-53
	HE5_SWattach	he5_swattach	Attaches to an existing swath within the file	2-47
	HE5_SWdetach	he5_swdetach	Detaches from swath interface	2-76
	HE5_SWclose	he5_swclose	Closes file	2-51
	HE5_SWdefdim	he5_swdefdim	Defines a new dimension within the swath	2-64
Definition	HE5_SWdefdimmap	he5_swdefmap	Defines the mapping between the geolocation and data dimensions	2-66
	HE5_SWdefidxmap	he5_swdefimap	Defines a non-regular mapping between the geolocation and data dimension	2-70
	HE5_SWdefgeofield	he5_swdefgfld	Defines a new geolocation field within the swath	2-68
	HE5_SWdefdatafield	he5_swdefdfld	Defines a new data field within the swath	2-62
	HE5_SWdefcomp	he5_swdefcomp	Defines a field compression scheme	2-59
	HE5_SWdefchunk	he5_swdefchunk	Define chunking parameters	2-56
	HE5_SWdefcomchunk	he5_swdefcomch	Defines compression with automatic chunking	2-57
	HE5_SWsetalias	he5_swsetalias	Defines alias for data field	2-140
	HE5_SWdropalias	he5_swdrpalias	Removes alias from the list of field aliases	2-78
	HE5_SWfldrename	he5_swfldrnm	Changes the field name	2-87
Basic I/O	HE5_SWwritefield	he5_swwrfld	Writes data to a swath field	2-155
	HE5_SWwritegeometa	he5_swwrgmeta	Writes field metadata for an existing swath geolocation field	2-160
	HE5_SWwritedatameta	he5_swwrdmeta	Writes field metadata for an existing swath data field	2-151
	HE5_SWreadfield	he5_swrdfld	Reads data from a swath field.	2-131
	HE5_SWwriteattr	he5_swwrattr	Writes/updates attribute in a swath	2-149
	HE5_SWreadattr	he5_swrdattr	Reads attribute from a swath	2-126
	HE5_SWwritegeogrpattr	he5_swwrgeogattr	Writes/updates group Geolocation Fields attribute in a swath	2-158
	HE5_SWwritegrpattr	he5_swwrgattr	Writes/updates group Data Fields attribute in a swath	2-161
	HE5_SWwritelocattr	he5_swwrlattr	Write/updates local attribute in a swath	2-163
	HE5_SWreadgeogrpattr	he5_swrdgeogattr	Reads attribute in Geolocation Fields from swath	2-133
	HE5_SWreadgrpattr	he5_swrdgattr	Reads attribute in Data Fields from a swath	2-134
	HE5_SWreadlocattr	he5_swrdlattr	Reads attribute from a swath	2-135
	HE5_SWsetfillvalue	he5_swsetfill	Sets fill value for the specified field	2-144
	HE5_SWgetfillvalue	he5_swgetfill	Retrieves fill value for the specified field	2-94
Inquiry	HE5_SWaliasinfo	he5_swaliasinfo	Retrieves information about field aliases	2-46
	HE5_SWgetaliaslist	he5_swgetaliaslist	Retrieves list and number of aliases in a geo or data group	2-90
	HE5_SWinqdims	he5_swinqdims	Retrieves information about dimensions defined in swath	2-105
	HE5_SWinqmaps	he5_swinqmaps	Retrieves information about the geolocation relations defined	2-117
	HE5_SWinqidxmaps	he5_swinqimaps	Retrieves information about the indexed geolocation/data mappings defined	2-114
	HE5_SWinqgeofields	he5_swinqgflds	Retrieves information about the geolocation fields defined	2-108
	HE5_SWinqdatafields	he5_swinqdflds	Retrieves information about the data fields defined	2-100
	HE5_SWinqattrs	he5_swinqattrs	Retrieves number and names of attributes defined	2-99
	HE5_SWinqdatatype	he5_swidtype	Returns data type information about specified fields in swath	2-101
	HE5_SWinqdfldalias	he5_swinqdfldalias	Returns information about data fields & aliases defined in swath	2-103

Table 5-1. Summary of the Swath Interface (2 of 3)

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
	HE5_SWinqgfldalias	he5_swinqgfldalias	Returns information about geolocation fields & aliases defined in swath	2-111
	HE5_SWinqgeogrpattrs	he5_swinqgeogattrs	Retrieve information about group Geolocation Fields attributes defined in swath	2-109
	HE5_SWinqgrpattrs	he5_swinqgattrs	Retrieve information about group Data Fields attributes defined in swath	2-113
	HE5_SWinqlocattrs	he5_swinqlattrs	Retrieve information about local attributes defined in swath	2-115
	HE5_SWlocattrinfo	he5_swlocattrinfo	Returns information about a data field's local attribute(s)	2-119
	HE5_SWlocattrinfo2	he5_swlocattrinfo2	Same as above, but also returns buffer size of attribute element.	2-119
	HE5_SWnentries	he5_swnentries	Returns number of entries and descriptive string buffer size for a specified entity	2-122
	HE5_SWdiminfo	he5_swdiminfo	Retrieve size of specified dimension	2-77
	HE5_SWchunkinfo	he5_swchunkinfo	Retrieve chunking information	2-49
	HE5_SWmapinfo	he5_swmapinfo	Retrieve offset and increment of specified geolocation mapping	2-120
	HE5_SWidxmapinfo	he5_swimapinfo	Retrieve offset and increment of specified geolocation mapping	2-96
	HE5_SWattrinfo	he5_swattrinfo	Returns information about swath attributes	2-48
	HE5_SWattrinfo2	he5_swattrinfo2	Same as above, but also returns buffer size of attribute element.	2-48
	HE5_SWgeogrpattrinfo	he5_swgeogattrinfo	Returns information about group Geolocation Fields attribute	2-88
	HE5_SWgeogrpattrinfo2	he5_swgeogattrinfo2	Same as above, but also returns buffer size of attribute element.	2-88
	HE5_SWgrpattrinfo	he5_swgattrinfo	Returns information about group Data Fields attribute	2-95
	HE5_SWgrpattrinfo2	he5_swgattrinfo2	Same as above, but also returns buffer size of attribute element.	2-95
	HE5_SWfieldinfo	he5_swfldinfo	Retrieve information about a specific geolocation or data field	2-85
	HE5_SWcompinfo	he5_swcompinfo	Retrieve compression information about a field	2-52
	HE5_SWinqswath	he5_swinqswath	Retrieves number and names of swaths in file	2-118
	HE5_SWregionindex	he5_swregidx	Returns information about the swath region ID	2-136
	HE5_SWupdateidxmap	he5_swupimap	Update map index for a specified region	2-147
	HE5_SWgeomapinfo	he5_swgmapinfo	Retrieve type of dimension mapping for a dimension	2-89
Subset	HE5_SWdefboxregion	he5_swdefboxreg	Define region of interest by latitude/longitude	2-54
	HE5_SWregioninfo	he5_swreginfo	Returns information about defined region	2-138
	HE5_SWextractregion	he5_swextreg	Read a region of interest from a field	2-84
	HE5_SWdeftimeperiod	he5_swdeftmeper	Define a time period of interest	2-71
	HE5_SWperiodinfo	he5_swperinfo	Returns information about a defined time period	2-124
	HE5_SWextractperiod	he5_swextper	Extract a defined time period	2-82
	HE5_SWdefvrtregion	he5_swdefvrtreg	Define a region of interest by vertical field	2-73
	HE5_SWindexinfo	he5_swindexinfo	Returns the indices about a subsetted region	2-96
	HE5_SWdupregion	he5_swdupreg	Duplicate a region or time period	2-81

Table 5-1. Summary of the Swath Interface (3 of 3)

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Profile	HE5_PRdefine	he5_prdefine	Defines profile data structure	2-165
	HE5_PRread	he5_prread	Reads profile data	2-171
	HE5_PRwrite	he5_prwrite	Writes profile data	2-175
	HE5_PRinquire	he5_prinquire	Retrieves information about profiles	2-170
	HE5_PRinfo	he5_prinfo	Return information about profile	2-167
	HE5_PRgrpattrinfo	he5_prgattrinfo	Returns information about a group Profile Fields attribute	2-166
	HE5_PRgrpattrinfo2	he5_prgattrinfo2	Same as above, but also returns buffer size of attribute element.	2-166
External Files	HE5_SWmountexternal	Not available	Mount external data file	2-121
	HE5_SWreadexternal	Not available	Read external data set	2-130
	HE5_SWunmount	Not available	Dismount external data file	2-146
External Data Sets	HE5_SWsetextdata	he5_swsetxdat	Set external data set	2-143
	HE5_SWgetextdata	he5_swgetxdat	Get external data set	2-93
Dimension Scale	HE5_SWsetdimscale	he5_swsetdimscale	Sets dimension scale for a field dimension within the swath	2-141
	HE5_SWdefdimscale	he5_swdefdimscale	Sets dimension scale for a dimension for all fields within the swath	2-141
	HE5_SWgetdimscale	he5_swgetdimscale	Gets dimension scale for a field dimension within the swath	2-91
	HE5_SWwritedscaleattr	he5_swwritedscaleattr	Writes/Updates a dimension scale attribute in a specific swath	2-152
	HE5_SWreaddscaleattr	he5_swreaddscaleattr	Reads a dimension scale attribute from a specific dimension	2-127
	HE5_SWinqdscaleattrs	he5_swinqdscaleattrs	Retrieve information about the attributes defined for a specific dimension scale	2-106
	HE5_SWdscaleattrinfo	he5_swdscaleattrinfo	Returns information about attribute(s) in a specific dimension scale	2-79
	HE5_SWdscaleattrinfo2	he5_swdscaleattrinfo2	Same as above, but also returns buffer size of attribute element.	2-79

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.

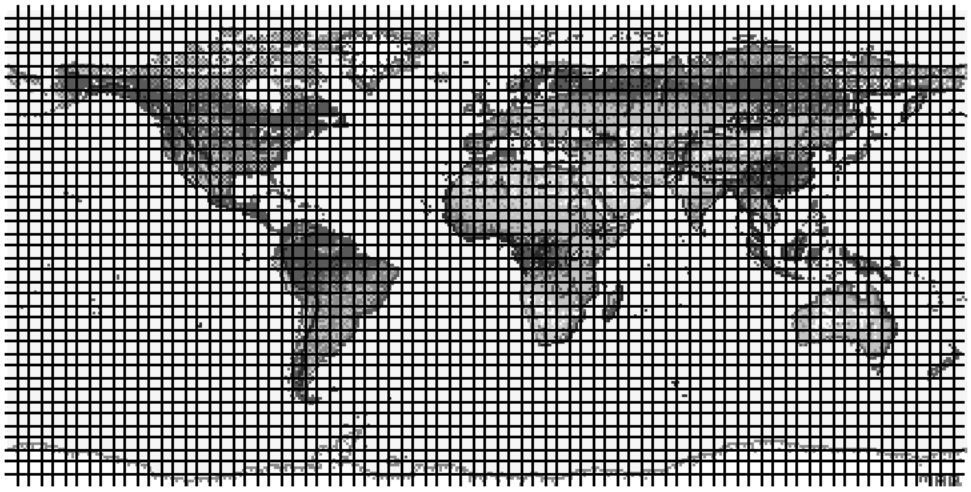

Figure 6-1. A Data Field in a Mercator-projected Grid

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.

Figure 6-2. A Data Field in an Interrupted Goode's Homolosine-Projected Grid

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 my 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 6-1. Summary of the Grid Interface (1 of 2)

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Access	HE5_GDopen	he5_gdopen	Creates a new file or opens an existing one	2-244
	HE5_GDcreate	he5_gdcreate	Creates a new grid in the file	2-187
	HE5_GDattach	he5_gdattach	Attaches to a grid	2-181
	HE5_GDdetach	he5_gddetach	Detaches from grid interface	2-209
	HE5_GDclose	he5_gdclose	Closes file	2-185
Definition	HE5_GDdeforigin	he5_gddeforigin	Defines origin of grid pixel	2-198
	HE5_GDdefdim	he5_gddefdim	Defines dimensions for a grid	2-195
	HE5_GDdefproj	he5_gddefproj	Defines projection of grid	2-200
	HE5_GDdefpixreg	he5_gddefpixreg	Defines pixel registration within grid cell	2-199
	HE5_GDdeffield	he5_gddeffld	Defines data fields to be stored in a grid	2-196
	HE5_GDdefcomp	he5_gddefcomp	Defines a field compression scheme	2-191
	HE5_GDblkSOMoffset	None	This is a special function for SOM MISR data. Write block SOM offset values.	2-183
	HE5_GDdefcomtile	he5_gddefcomtle	Defines compression with automatic tiling	2-194
	HE5_GDsetalias	he5_gdsetalias	Defines alias for data field	2-259
	HE5_GDdropalias	he5_gddrpalias	Removes alias from a list of field aliases	2-211
Basic I/O	HE5_GDwritefieldmeta	he5_gdwrmeta	Writes metadata for field already existing in file	2-275
	HE5_GDwritefield	he5_gdwrfld	Writes data to a grid field.	2-272
	HE5_GDreadfield	he5_gdrdfld	Reads data from a grid field	2-253
	HE5_GDwriteattr	he5_gdwrattr	Writes/updates attribute in a grid.	2-267
	HE5_GDwritelocattr	he5_gdwrlattr	Writes/updates local attribute in a grid	2-278
	HE5_GDwritegrpattr	he5_gdwrgattr	Writes/updates group attribute in a grid	2-276
	HE5_GDreadattr	he5_gdrdattr	Reads attribute from a grid	2-249
	HE5_GDreadgrpattr	he5_gdrdgattr	Reads group attribute from a grid	2-255
	HE5_GDreadlocattr	he5_gdrdlattr	Reads local attribute from a grid	2-256
	HE5_GDsetfillvalue	he5_gdsetfill	Sets fill value for the specified field	2-264
	HE5_GDgetfillvalue	he5_gdgetfill	Retrieves fill value for the specified field	2-221
Inquiry	HE5_GDgetaliaslist	he5_gdgetaliaslist	Retrieves list and number of aliases in a data group	2-217
	HE5_GDinqdims	he5_gdinqdims	Retrieves information about dimensions defined in grid	2-231
	HE5_GDinqfields	he5_gdinqdflds	Retrieves information about the data fields defined in grid	2-234
	HE5_GDinqattrs	he5_gdinqattrs	Retrieves number and names of attributes defined	2-228
	HE5_GDinqlocattrs	he5_gdinqlattrs	Retrieves information about local attributes defined for a field	2-238
	HE5_GDinqgrpattrs	he5_gdinqgattrs	Retrieves information about group attributes defined in grid	2-237
	HE5_GDnentries	he5_gdnentries	Returns number of entries and descriptive string buffer size for a specified entity	2-243
	HE5_GDaliasinfo	he5_gdaliasinfo	Retrieves information about aliases	2-180
	HE5_GDgridinfo	he5_gdgridinfo	Returns dimensions of grid and X-Y coordinates of corners	2-226
	HE5_GDprojinfo	he5_gdprojinfo	Returns all GCTP projection information	2-248
	HE5_GDdiminfo	he5_gddiminfo	Retrieves size of specified dimension.	2-210

Table 6-1. Summary of the Grid Interface (2 of 2)

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
	HE5_GDcompinfo	he5_gdcompinfo	Retrieves compression info about a field	2-186
	HE5_GDfieldinfo	he5_gdfldinfo	Retrieves information of a specific field in the grid	2-215
	HE5_GDinqgrid	he5_gdinqgrid	Retrieves number and names of grids in file	2-236
	HE5_GDinqfldalias	he5_gdinqfldalias	Returns information about data fields & aliases defined in grid	2-235
	HE5_GDinqdatatype	he5_gdinqdatatype	Returns data type information about specified fields in grid	2-229
	HE5_GDattrinfo	he5_gdattrinfo	Returns information about grid attributes	2-182
	HE5_GDattrinfo2	he5_gdattrinfo2	Same as above, but also returns buffer size of attribute element.	2-182
	HE5_GDgrpattrinfo	he5_gdgatinfo	Returns info about a grid group attribute	2-227
	HE5_GDgrpattrinfo2	he5_gdgatinfo2	Same as above, but also returns buffer size of attribute element.	2-227
	HE5_GDlocattrinfo	he5_gdlatinfo	Returns information about a Data Field's local attribute(s)	2-242
	HE5_GDlocattrinfo2	he5_gdlatinfo2	Same as above, but also returns buffer size of attribute element.	2-242
	HE5_GDorigininfo	he5_gdorginfo	Returns information about grid pixel origin	2-246
	HE5_GDpixreginfo	he5_gdpreginfo	Returns pixel registration info for given grid	2-247
Subset	HE5_GDdefboxregion	he5_gddefboxreg	Defines region of interest by latitude/longitude	2-190
	HE5_GDregioninfo	he5_gdreginfo	Returns information about a defined region	2-257
	HE5_GDextractregion	he5_gdextrreg	Read a region of interest from a field	2-214
	HE5_GDdeftimeperiod	he5_gddeftmeper	Define a time period of interest	2-205
	HE5_GDdefvrtregion	he5_gddefvrtreg	Define a region of interest by vertical field	2-207
	HE5_GDgetpixels	he5_gdgetpix	Get row/columns for lon/lat pairs	2-222
	HE5_GDgetpixvalues	he5_gdgetpixval	Get field values for specified pixels	2-224
	HE5_GDinterpolate	he5_gdinterpolate	Perform bilinear interpolation on a grid field	2-240
	HE5_GDdupregion	he5_gddupreg	Duplicate a region or time period	2-213
Tiling	HE5_GDdeftile	he5_gddeftle	Define a tiling scheme	2-202
	HE5_GDtileinfo	he5_gdtileinfo	Retrieve tiling information	2-265
Utility	HE5_GDij2ll	he5_Gdij2ll	convert (i,j) coordinates to (lon,lat) for a grid	2-364
	HE5_GDll2ij	he5_Gdll2ij	convert (lon,lat) coordinates to (i,j) for a grid	2-367
	HE5_GDrs2ll	he5_gdrs2ll	Convert (r,s) coordinates to (lon,lat) of EASE grid	2-370
External Data Sets	HE5_GDsetextdata	he5_gdsetxdat	Set external data set	2-263
	HE5_GDgetextdata	he5_gdgetxdat	Get external data set	2-220
Dimension Scale	HE5_GDsetdimscale	he5_gdsetdimscale	Sets dimension scale for a field dimension within the grid	2-260
	HE5_GDdefdimscale	he5_gddefdimscale	Sets dimension scale for a dimension of all fields within the grid	2-260
	HE5_GDgetdimscale	he5_gdgetdimscale	Gets dimension scale for a field dimension in a grid	2-218
	HE5_GDwritedscaleattr	he5_gdwritedscaleattr	Writes/Updates a dimension scale attribute in a specific grid	2-269
	HE5_GDreaddscaleattr	he5_gdreaddscaleattr	Reads a dimension scale attribute from a specific dimension	2-250
	HE5_GDinqdscaleattrs	he5_gdinqdscaleattrs	Retrieve information about the attributes defined for a specific dimension scale	2-232
	HE5_GDdscaleattrinfo	he5_gddscaleattrinfo	Returns information about attribute(s) in a specific dimension scale	2-212
	HE5_GDdscaleattrinfo2	he5_gddscaleattrinfo2	Same as above, but also returns buffer size of attribute element.	2-212

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:

```
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 described 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_SPCS         (2)   State Plane Coordinate System
GCTP_ALBERS       (3)   Albers Conical Equal_Area
GCTP_LAMCC        (4)   Lambert Conformal Conic
GCTP_MERCAT       (5)   Mercator
GCTP_PS           (6)   Polar Stereographic
GCTP_POLYC        (7)   Polyconic
GCTP_TM           (9)   Transverse Mercator
GCTP_LAMAZ        (11)  Lambert Azimuthal Equal Area
GCTP_SNSOID       (16)  Sinusoidal
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 Intergerized Sinusoidal Projection is not part of the original GCTP package. It has been added by ECS. See *Level-3 SeaWiFS Data Products: Spatial and Temporal Binning Algorithms*. Additional references are provided in Section 2.

** 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, December 2017, (333-EED2-001, 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.

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

6.5.3 GCTP Spheroid Codes

Clarke 1866 (default)	(0)
Clarke 1880	(1)
Bessel	(2)
International 1967	(3)
International 1909	(4)
WGS 72	(5)
Everest	(6)
WGS 66	(7)
GRS 1980	(8)
Airy	(9)
Modified Airy	(10)
Modified Everest	(11)
WGS 84	(12)
Southeast Asia	(13)
Australian National	(14)
Krassovsky	(15)
Hough	(16)
Mercury 1960	(17)
Modified Mercury 1968	(18)
Sphere of Radius 6370997m	(19)
Sphere of Radius 6371228m	(20)
Sphere of Radius 6371007.181m	(21)

6.5.4 Projection Parameters

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

Table 6-2. Projection Transformation Package Projection Parameters

	Array Element							
Code & Projection Id	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0 Geographic								
1 U T M	Lon/Z	Lat/Z						
2 PGSd_SPCS			Spheroid	Zone				
3 Albers Conical Equal_Area	Smajor	Sminor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
4 Lambert Conformal C	Smajor	Sminor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
5 Mercator	Smajor	Sminor			CentMer	TrueScale	FE	FN
6 Polar Stereographic	Smajor	Sminor			LongPol	TrueScale	FE	FN
7 Polyconic	Smajor	Sminor			CentMer	OriginLat	FE	FN
9 Transverse Mercator	Smajor	Sminor	Factor		CentMer	OriginLat	FE	FN
11 Lambert Azimuthal**	Smajor	Sminor			CentLon	CenterLat	FE	FN
Lambert Azimuthal	Sphere				CentLon	CenterLat	FE	FN
16 PGSd_SNSOID**	Smajor	Sminor			CentMer			
PGSd_SNSOID	Sphere				CentMer			
20 Hotin Oblique Merc A	Smajor	Sminor	Factor			OriginLat	FE	FN
20 Hotin Oblique Merc B	Smajor	Sminor	Factor	AziAng	AzmthPt	OriginLat	FE	FN
22 Space Oblique Merc A	Smajor	Sminor		IncAng	AscLong		FE	FN
22 Space Oblique Merc B	Smajor	Sminor	Satnum	Path			FE	FN
24 Interrupted Goode	Sphere							
97 CEA utilized by EASE grid (see Notes)	Smajor	Sminor			CentMer	TrueScale	FE	FN
98 BCEA utilized by EASE grid (see Notes)	Smajor	Sminor			CentMer	TrueScale	FE	FN

** Lambert Azimuthal and Sinusoidal support both spherical and WGS84 ellipsoidal Earth model

Table 6-3. Projection Transformation Package Projection Parameters Elements

	Array Element				
Code & Projection Id	**9**	**10**	**11**	**12**	**13**
0 Geographic					
1 U T M					
2 PGSd_SPCS					
3 Albers Conical Equal_Area					
4 Lambert Conformal C					
5 Mercator					
6 Polar Stereographic					
7 Polyconic					
9 Transverse Mercator					
11 Lambert Azimuthal					
Lambert Azimuthal					
16 PGSd_SNSOID					
PGSd_SNSOID					
20 Hotin Oblique Merc A	Long1	Lat1	Long2	Lat2	zero
20 Hotin Oblique Merc B					one
22 Space Oblique Merc A	PSRev	SRat	PFlag	HDF-EOS Para	zero
22 Space Oblique Merc B				HDF-EOS Para	one
24 Interrupted Goode					
31 & 99 Integerized Sinusoidal	NZone		RFlag		
97 CEA utilized by EASE grid (see Notes)					
98 BCEA utilized by EASE grid (see Notes)					

Where,

Lon/Z Longitude of any point in the UTM zone or zero. If zero, a zone code must be specified.

Lat/Z Latitude of any point in the UTM zone or zero. If zero, a zone code must be specified.

Smajor Semi-major axis of ellipsoid. If zero, Clarke 1866 in meters is assumed. It is recommended that explicit value, rather than zero, is used for Smajor.

Sminor	Eccentricity squared of the ellipsoid if less than one, if zero, a spherical form is assumed, or if greater than one, the semi-minor axis of ellipsoid. It should be noted that a negative sphere code should be used in order to have user specified Smajor and Sminor be accepted by GCTP, otherwise default ellipsoid Smajor and Sminor will be used.
Sphere	Radius of reference sphere. If zero, 6370997 meters is used. It is recommended that explicit value, rather than zero, is used for Sphere.
STDPR1	Latitude of the first standard parallel.
STDPR2	Latitude of the second standard parallel.
CentMer	Longitude of the central meridian.
OriginLat	Latitude of the projection origin.
FE	False easting in the same units as the semi-major axis.
FN	False northing in the same units as the semi-major axis.
TrueScale	Latitude of true scale.
LongPol	Longitude down below pole of map.
Factor	Scale factor at central meridian (Transverse Mercator) or center of projection (Hotine Oblique Mercator).
CentLon	Longitude of center of projection.
CenterLat	Latitude of center of projection.
Long1	Longitude of first point on center line (Hotine Oblique Mercator, format A).
Long2	Longitude of second point on center line (Hotine Oblique Mercator, frmt A).
Lat1	Latitude of first point on center line (Hotine Oblique Mercator, format A).
Lat2	Latitude of second point on center line (Hotine Oblique Mercator, format A).
AziAng	Azimuth angle east of north of center line (Hotine Oblique Mercator, frmt B).
AzmthPt	Longitude of point on central meridian where azimuth occurs (Hotine Oblique Mercator, format B).
IncAng	Inclination of orbit at ascending node, counter-clockwise from equator (SOM, format A).
AscLong	Longitude of ascending orbit at equator (SOM, format A).
PSRev	Period of satellite revolution in minutes (SOM, format A).
SRat	Satellite ratio to specify the start and end point of x,y values on earth surface (SOM, format A -- for Landsat use 0.5201613).
PFlag	End of path flag for Landsat: 0 = start of path, 1 = end of path (SOM, frmt A).

Satnum Landsat Satellite Number (SOM, format B).

Path Landsat Path Number (Use WRS-1 for Landsat 1, 2 and 3 and WRS-2 for Landsat 4 and 5.) (SOM, format B).

Nzone Number of equally spaced latitudinal zones (rows); must be two or larger and even.

Rflag Right justify columns flag is used to indicate what to do in zones with an odd number of columns. If it has a value of 0 or 1, it indicates the extra column is on the right (zero) 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 projetion 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 desides the 25km global EASE grid and having freedom to use different appropriate projection parameters. One can create the above mentioned 25km global EASE grid using:

Grid Dimensions:

Width 1383

Height 586

Grid Extent:

UpLeft Latitude 86.72

LowRight Latitude –86.72

UpLeft Longitude –180.00

LowRight Longitude 180.00

Projection Parameters:

1) 6371.2280/25.067525 = 254.16263

2) 6371.2280/25.067525 = 254.16263

5) 0.0

6) 30000000.0

7) 691.0

8) –292.5

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

Grid Dimensions:

Width 2766

Height 1171

Grid Extent:

UpLeft Latitude 85.95

LowRight Latitude –85.95

UpLeft :Longitude –179.93

LowRight Longitude 180.07

Projection Parameters:

1) 6371.2280 / (25.067525/2) = 508.325253

2) 6371.2280 / (25.067525/2) = 508.325253

5) 0.0

6) 30000000.0

7) 1382.0

8) –585.0

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

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

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_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);

  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
    {
      char        label[8];
      double      lon;
      double      lat;
      int         date;
      char        id[8];
    } InputData2;


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 struvcture */
      /* ---------------------------- */
      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 withing the C data structure */

        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 struvcture */
        /* ---------------------------- */
        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 withing 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[3] = H5T_NATIVE_INT;
        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 withing 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[2] = 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 withing 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;
        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);
      }
  else
    {
        printf("File ID returned by HE5_PTopen() : %d \n", ptfid);
    }

  return 0;
}
```

Example 3

```
/*
 * In this example we will write data to the specified level of Point
 */



/* he5_pt_writedata */



#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;

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);

  /* Write to Simple Point */
  /* --------------------- */
  PTid = HE5_PTattach(ptfid, "Simple Point");

  /* Open input data file */
  /* -------------------- */
  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];
        memmove(datbuf_1[n].Spc, spc, sizeof(char)*strlen(spc));
        datbuf_1[n].Spc[strlen(spc)] = 0;
        n++;
    }

  fclose(fp);

  datasize = (size_t)sizeof(CmpData_1);
  count[0] = n;

  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);


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


  /* Write First (0th) Level */
  /* ----------------------- */
  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;
        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("%lf  %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;
  buf_1.Conc[2] = 3.33;
  buf_1.Conc[3] = 4.44;

  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

```
      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)

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


c     Open the HDF point file, "Point.he5"
c     ------------------------------------
      ptfid = he5_ptopen('Point.he5',HE5F_ACC_TRUNC)
      write(*,*) 'File ID returned by he5_ptopen():  ',ptfid


      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


c ---------------------------------------------------------------------------
c
c     We now close the point interface with the he5_ptdetach routine.
c
c     This step is necessary to properly store the point information within
c
c     the file.
c
```

```
c ------------------------------------------------------------------------------
c

      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
```

```
      character*240 fieldlist4
      character*240 fieldlist5
      character*80  levelname
      character*20  parent
      character*20  child
      character*20  linkfield

      integer       HE5F_ACC_RDWR
      parameter     (HE5F_ACC_RDWR=100)

      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)



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     Read Simple Point
c     -----------------
      ptid = HE5_PTattach(ptfid, "Simple Point")
      write(*,*) 'Point ID returned by he5_ptattach():  ',ptid


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

      levelname = 'Sensor'

      rank_1(1)  = 1
      rank_1(2)  = 1
      rank_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(ptid, levelname, rank_1, fieldlist1,
     1dims_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

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


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

      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,
     1dims_2, dtype_2, array_2)
      write(*,*) 'Status returned by he5_ptdeflevel():  ',status


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

      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
```

```
      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,
     1dims_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

c.....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,
     1dims_4, dtype_4, array_4)
      write(*,*) 'Status returned by he5_ptdeflevel():  ',status

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

      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,
     1dims_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

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
```

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_ptwrbckptr
      integer       he5_ptwrfwdptr
      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
c     -----------------
      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),
     1                                  concentration_tt(2),
     2                                  concentration_tt(3),
     3                                  concentration_tt(4),
     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)
  110 format(F13.1,F6.2,F6.2,F6.2,F6.2,2X,A8)

      count(1) = n

      fieldname      = 'Time'
      datatype       = HE5T_NATIVE_DOUBLE

      status = he5_ptwritelevel(ptid, 0, count, fieldname,
     1datatype, time)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

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 dimention order from
c FORTRAN to C

      status    = he5_ptfort2c(dimens, rank, datatype, conc,
     1outconc)
      write(*,*) 'Status returned by he5_ptfort2c():  ',status

      fieldname      = 'Concentration'

      status = he5_ptwritelevel(ptid, 0, count, fieldname,
     1datatype, outconc)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status


      fieldname      = 'Species'
      datatype       = HE5T_NATIVE_CHAR
```

```
      status = he5_ptwritelevel(ptid, 0, count, fieldname,
     1datatype, spc)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

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

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

c     Read FixedBuoy Point
c     --------------------
      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,
     1id_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,
     1datatype, desc)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

      fieldname      = 'Longitude'
      datatype       = HE5T_NATIVE_DOUBLE

      status = he5_ptwritelevel(ptid, 0, count, fieldname,
     1datatype, lon)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

      fieldname      = 'Latitude'
      datatype       = HE5T_NATIVE_DOUBLE

      status = he5_ptwritelevel(ptid, 0, count, fieldname,
     1datatype, lat)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

      fieldname      = 'DeployDate'
      datatype       = HE5T_NATIVE_INT
```

```
      status = he5_ptwritelevel(ptid, 0, count, fieldname,
     1datatype, date)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

      fieldname      = 'ID'
      datatype       = HE5T_NATIVE_CHAR

      status = he5_ptwritelevel(ptid, 0, count, fieldname,
     1datatype, id)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

c     Read FixedBuoy Point - Level 1
c     ------------------------------
      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,
     1datatype, time3)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

      fieldname      = 'Rainfall'
      datatype       = HE5T_NATIVE_FLOAT

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

      fieldname      = 'Temperature'
      datatype       = HE5T_NATIVE_FLOAT

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

      fieldname      = 'ID'
      datatype       = HE5T_NATIVE_CHAR

      status = he5_ptwritelevel(ptid, 1, count, fieldname,
     1datatype, 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"
c     --------------------------------------
      attrname       = 'GlobalAttribute_int'
      ntype          = HE5T_NATIVE_INT
      fortcount(1)   = 1
      attr           = 9999

      status   = he5_ptwriteattr(ptid,attrname,ntype,fortcount,
     1attr)
      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
c     ------------------
      ptid = he5_ptattach(ptfid, "FloatBuoy Point")
      write(*,*) 'Point ID returned by he5_ptattach():  ',ptid

c     Read FloatBuoy Point - Level 0
c     ------------------------------
      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,
     1datatype, desc4)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

      fieldname      = 'DeployDate'
      datatype       = HE5T_NATIVE_INT
```

```
      status = he5_ptwritelevel(ptid, 0, count, fieldname,
     1datatype, date4)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

      fieldname      = 'Weight'
      datatype       = HE5T_NATIVE_INT

      status = he5_ptwritelevel(ptid, 0, count, fieldname,
     1datatype, wgt)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

      fieldname      = 'ID'
      datatype       = HE5T_NATIVE_CHAR

      status = he5_ptwritelevel(ptid, 0, count, fieldname,
     1datatype, id4)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

c     Read FixedBuoy Point - Level 1
c     ------------------------------
      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,
     1rain5_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,
     1datatype, time5)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

      fieldname      = 'Longitude'
      datatype       = HE5T_NATIVE_DOUBLE

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

      fieldname      = 'Latitude'
      datatype       = HE5T_NATIVE_DOUBLE

      status = he5_ptwritelevel(ptid, 1, count, fieldname,
```

```
     1datatype, lat5)
      write(*,*) 'Status returned by he5_ptwritelevel():  ',status

      fieldname      = 'Rainfall'
      datatype       = HE5T_NATIVE_FLOAT

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

      fieldname      = 'Temperature'
      datatype       = HE5T_NATIVE_FLOAT

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

      fieldname      = 'ID'
      datatype       = HE5T_NATIVE_CHAR

      status = he5_ptwritelevel(ptid, 1, count, fieldname,
     1datatype, 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"
c     -------------------------------------
      attrname       = 'GlobalAttribute_float'
      ntype          = HE5T_NATIVE_FLOAT
      fortcount(1)   = 1
      flt            = -7.5

      status   = he5_ptwriteattr(ptid,attrname,
     1ntype,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
```

Example 4

```
      program       he5_pt_readdataF_32
```

```
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_ptinqattrs
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)
```

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

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


c     Get level information
c     ---------------------
      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

c     Get the number of records in level
c     ----------------------------------
      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

c     Populate input information structure
c     ------------------------------------
      dtype(1)     =  HE5T_NATIVE_DOUBLE
      dtype(2)     =  HE5T_NATIVE_FLOAT
      dtype(3)     =  HE5T_NATIVE_CHAR

c     Read the level data and print out
c     ---------------------------------
      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

c     Convert 'C' array to Fortran order
c     ----------------------------------
```

```
      dimens(1) = 15
      dimens(2) = 4
      arr_rank  = 2
      datatype  = HE5T_NATIVE_FLOAT

c This is a call to utility routine reversing dimention order from
c 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),
     1dtype(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
c     --------------------
      ptid = he5_ptattach(ptfid, "FixedBuoy Point")
      write(*,*) 'Point ID returned by he5_ptattach():  ',ptid

c     Global Attributes
c     -----------------
      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
```

```
      stop
      end
```

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)


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

      dtype(1)     =  HE5T_NATIVE_DOUBLE
      dtype(2)     =  HE5T_NATIVE_FLOAT
```

```
      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,
     1recs, dtype(2), conc_tt)
      write(*,*) 'Status returned by he5_ptupdatelevel():  ',status

      fieldname = 'Time'

      time_tt   = 13131313.0

      status = he5_ptupdatelevel(ptid, level, fieldname, nrec,
     1recs, dtype(1), time_tt)
      write(*,*) 'Status returned by he5_ptupdatelevel():  ',status

      fieldname = 'Species'

      spc_tt    = 'AM'

      status = he5_ptupdatelevel(ptid, level, fieldname, nrec,
     1recs, 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
```

Example 6

```
      program       he5_pt_datainfoF_32

      implicit      none

      integer       status
      integer       ptfid
      integer       ptid
```

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

      integer*4     size

      character*1   null_char_0
      character*80  fieldname
      character*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"
c     -------------------------------------
      ptfid = he5_ptopen('Point.he5',HE5F_ACC_RDONLY)
      write(*,*) 'File ID returned by he5_ptopen():  ',ptfid

c     Read Simple Point
c     -----------------
      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,
     1dtype,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,
     1dtype,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,
     1dtype,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,
     1dtype,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

```
/*
 *   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);
```

```
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);

/* Define Unlimited Dimension */
/* -------------------------- */
status = HE5_SWdefdim(SWid, "Unlim", H5S_UNLIMITED);
printf("Status returned by HE5_SWdefdim():      %d \n", status);

/*
 * 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 theHE5_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".
 *
 * Note that there is no relationship between the geolocation dimensions
 * and the "Bands" dimension.
 */

/* 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);
```

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


  return 0;
}
```

Example 2

```
/*
 *     In this example we (1) open the "Swath.he5" HDF-EOS file,
 *   (2) attach to the "Swath1" swath, and (3) define the fields
 */

/* he5_sw_definefields */


#include <HE5_HdfEosDef.h>

#define  RANK   1


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_SWdefgeofield(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,GeoXtrack",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("\tStatus returned by HE5_SWdefchunk() :
%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;

              /* 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() :
%d\n",status);

              status = HE5_SWdefdatafield(SWid, "Count", "GeoTrack", "Unlim",
H5T_NATIVE_INT, 0);
              printf("Status returned by HE5_SWdefdatafield(...\"Count\",...)
:       %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            attr5[4] = {1111111L,2222222L,3333333L,4444444L};/* Global
'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 longitute field */
              /* ---------------------- */
              status = HE5_SWwritefield(SWid, "Longitude", start, NULL, count,
lng);
              printf("status returned by HE5_SWwritefield(\"Longitude\"):
%d\n", status);

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

              /* Write Spectra Field */
              /* ------------------- */
              start[0] = 0;      count[0] = 15;
              start[1] = 0;      count[1] = 40;
              start[2] = 0;      count[2] = 20;

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

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

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

```
  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 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("\tThe %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("\t\t %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("\t\t %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("\t\t %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("\t\t %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("\t\t %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("\t %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("\t\t %lf \n", datbuf[i]);

              free(datbuf);
              free(ntype);
            }
    }

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

  return 0;
}
```

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, nflds, 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_SWinqdims(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_SWnentries(SWid, HE5_HDFE_NENTMAP, &strbufsize);

off = (long *)calloc(nmaps, sizeof(long));
inc = (long *)calloc(nmaps, sizeof(long));
dimmap = (char *)calloc(strbufsize + 1, 1);

nmaps = HE5_SWinqmaps(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_SWnentries(SWid, HE5_HDFE_NENTIMAP, &strbufsize);
sizes = (hsize_t *) calloc(nmaps, sizeof(hsize_t));
dimmap = (char *) calloc(strbufsize + 1, 1);
nmaps = HE5_SWinqidxmaps(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_SWnentries(SWid, HE5_HDFE_NENTGFLD, &strbufsize);
rank = (int *)calloc(nflds, sizeof(int));
fieldlist = (char *) calloc(strbufsize + 1, 1);
nflds = HE5_SWinqgeofields(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_SWnentries(SWid, HE5_HDFE_NENTDFLD, &strbufsize);
rank = (int *) calloc(nflds, sizeof(int));
fieldlist = (char *) calloc(strbufsize + 1, 1);
nflds = HE5_SWinqdatafields(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 */
            /* ---------------------------------------------- */
            dimsize = HE5_SWdiminfo(SWid, "IndxTrack");
            indx = (long *) calloc(dimsize, 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("\t\t Data type:          %d\n", dtype);
            printf("\t\t 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("\t\t Data type:          %d\n", dtype);
            printf("\t\t 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("Local Attribute:\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_SWinqgrpattrs(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_SWinqgrpattrs(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

```
c  In this example we (1) open an HDF-EOS file, (2) create the
c  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            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/


c     Open the HDF-EOS file, "swath.he5" using HE5F_ACC_TRUNC access code
c     ---------------------------------------------------------------------
      swfid = he5_swopen("swath.he5",HE5F_ACC_TRUNC)

c     Create the swath, "Swath1", within the file
c     -------------------------------------------
      swid = he5_swcreate(swfid, "Swath1")
```

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

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

c     ----------------------------------------------------------------------
c     Once the dimensions are defined, the relationship (mapping)between the
c     geolocation dimensions, such as track and cross track, and the data
c     dimensions, must be established.  This is done through the SWdefdimmap
c     routine.  It takes as input the swath id, the names of the dimensions
c     designating the geolocation and data dimensions, respectively, and the
c     offset and increment defining the relation.
c
c     In the first example we relate the "GeoTrack" and "Res2tr" dimensions
c     with an offset of 0 and an increment of 2.  Thus the ith element of
c     "Geotrack" corresponds to the 2 * ith element of "Res2tr".
c
c     In the second example, the ith element of "GeoXtrack" corresponds to
c     the 2 * ith + 1 element of "Res2xtr".
c     ----------------------------------------------------------------------

c     Define dimension mappings
c     -------------------------
      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
c     --------------------------------
      status = he5_swdefimap(swid, "IndxTrack", "Res2tr", indx)

c     Detach from the swath
c     ---------------------
      status = he5_swdetach(swid)

c     Close the swath file
c     --------------------
      status = he5_swclose(swfid)

      stop
      end
```

Example 2

```
c     In this example we (1) open the "swath.he5" HDF-EOS file, (2)
c     attach to the "Swath1" swath, and (3) define the swath fields.
c
      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
c     ------------------------------------------------------------------
      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
```

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

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


      endif
      endif

c     Detach from the swath
c     ---------------------
      status = he5_swdetach(swid)

c     Close the file
c     --------------
      status = he5_swclose(swfid)

      stop
      end
```

Example 3

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


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

c     Open HDF-EOS file, "swath.h5"
c     -----------------------------
      swfid = he5_swopen("swath.h5", HE5F_ACC_RDWR)
      if (swfid .NE. FAIL) then
             SWid = he5_swattach(swfid, "Swath1")
             if (swid .NE. FAIL) then

c     Write data starting at the beginning of each cross track
c     --------------------------------------------------------
                   start(1)  = 0
```

```
                 stride(1) = 1
                 stride(2) = 1
                 count(1)  = 10
                 count(2)  = 1

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

c     Write Spectra one plane at a time
c     Value is 100 * track index + band index (0-based)
c     -------------------------------------------------
                 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,
     1             stride,count,plane)
                 enddo

c     Populate data buffer and write data to the Profile Field
c     --------------------------------------------------------
      datasize = 0
      counter  = 0
      do i=1,4

         len(i)    = i*25
         datasize  = datasize + len(i)
```

```
         do j=1,(25*i)
            counter = counter + 1
            buffer(counter) = (i)*1000 + j - 1
         enddo
      enddo

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

      status = he5_prwrite(swid,"Profile-2000",start,stride,count,
     1datasize,len,buffer)
      write(*,*) 'Status returned by he5_prwrite():  ',status

c     Write User defined Attribute
c     ----------------------------
                  attr(1) = 3
                  attr(2) = 5
                  attr(3) = 7
                  attr(4) = 11
                  count(1) = 4
                  status = he5_swwrattr(swid,"TestAttr",HE5T_NATIVE_INT,
     1           count,attr)

             endif
        endif

c     Detach from the swath
c     ---------------------
      status = he5_swdetach(swid)
      write(*,*) 'Status returned by he5_swdetach():  ',status

c     Close the file
c     --------------
      status = he5_swclose(swfid)
      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_swrdattr
      integer            he5_prread
      integer            he5_swdetach
```

```
      integer            he5_swclose
      integer            swfid, swid
      integer            buffer_out(250)
      integer            i,j,j1
      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)

c     Open HDF swath file, "swath.h5"
c     -------------------------------
        swfid = he5_swopen("swath.h5",HE5F_ACC_RDWR)
        if (swfid .NE. FAIL) then
             swid = he5_swattach(swfid, "Swath1")
             if (swid .NE. FAIL) then


c     Read the entire Longitude field
c     -------------------------------
                  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 Longitude ',i,j,lng(j,i)
                     enddo
                  enddo

c     Read data from the Profile
c     --------------------------
                  start(1)  = 0
                  stride(1) = 1
                  count(1)  = 4

      status = he5_prread(swid,"Profile-2000",start,stride,count,
     1len,buffer_out)
      write(*,*) '   '
```

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

c     Display the Profile data
c     ------------------------
      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(*,*) '   '

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

            endif
        endif
```

```
c     Detach from swath
c     -----------------
        status = he5_swdetach(swid)
      write(*,*) 'Status from he5_swdetach:  ',status

c     Close the file
c     --------------
        status = he5_swclose(swfid)
      write(*,*) 'Status from he5_swclose:  ',status

        stop
        end
```

Example 5

```
c  In this example we (1) open the "swath.he5" HDF-EOS file, (2) attach to
c  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_swdeftmeper
      integer     he5_swextreg
      integer     he5_swdetach
      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      tmebuf(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)


c     Open HDF-EOS swath file, "swath.he5"
```

```
c     ------------------------------------
      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.

c           Define box region
c           -----------------
            regionid = he5_swdefboxreg(swid,cornerlon,
     1cornerlat,HE5_HDFE_MIDPOINT)
            write(*,*) regionid,swid

            status = he5_swreginfo(swid,regionid,"Spectra",ntype,
     1rank,dims,size)
            write(*,*) dims(1), dims(2), dims(3), rank, ntype, size

c           Extract region data
c           -------------------
            status = he5_swextreg(swid,regionid,"Spectra",
     1HE5_HDFE_INTERNAL,datbuf)

c     Time Subsetting
c     ---------------
      t1  = 35232487.2d0
      t2  = 36609898.1d0
      periodid = he5_swdeftmeper(swid,t1,t2,HE5_HDFE_MIDPOINT)
      write(*,*) 'Time Subset: ', periodid,swid

      status = he5_swperinfo(swid,periodid,"Time",ntype,rank,
     1dims,size)
      write(*,*) 'Time Subset: ', rank, dims(1), size

c           Extract Time data
c           -----------------
            status = he5_swextper(swid,periodid,"Time",
     1HE5_HDFE_INTERNAL,tmebuf)

            do 10 i=1,size/8
                  write(*,*) i, tmebuf(i)
 10         continue

        endif

c        Detach from swath
c        -----------------
         status = he5_swdetach(swid)

c        Close the file
c        --------------
         status = he5_swclose(swfid)

      endif

      stop
      end
```

Example 6

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

      program         he5_sw_infoF_32

      implicit        none

        integer         i
      integer         status
        integer         swfid, swid
        integer         he5_swopen
        integer         he5_swattach
        integer         he5_swfldinfo
        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    maxdimlst
      character*72    fieldlist

        integer         HE5F_ACC_RDONLY
        parameter       (HE5F_ACC_RDONLY=101)

        integer         FAIL
        parameter       (FAIL=-1)
```

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

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

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

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

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

c     Inquire Geolocation Fields
c     --------------------------
                  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(*,*)

c     Inquire Data Fields
c     -------------------
                  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(*,*)

c       Get info on "GeoTrack" dim
c     --------------------------
```

```
            dimsize = he5_swdiminfo(swid, "GeoTrack")
            write(*,*) 'Size of GeoTrack: ', dimsize
            write(*,*)


c       Get info on "GeoTrack/Res2tr" mapping
c     -------------------------------------
            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
c     ----------------------------------------------
            n = he5_swimapinfo(swid, "IndxTrack", "Res2tr", indx)
            do i=1,n
                   write(*,*) 'Index Mapping Entry ', i, indx(i)
            enddo
            write(*,*)

c       Get info on "Longitude" Field
c     -----------------------------
            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)
            enddo

         endif
        endif

c     Detach from swath
c     -----------------
        status = he5_swdetach(swid)

c     Close the file
c     --------------
        status = he5_swclose(swfid)

        stop
        end
```

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     gdfid  = 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, gdfid,
   * which is used to identify the file in subsequent routines in the
   * library.
   */
  gdfid = 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]  =  813931.10959;
  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 DDDMMMSSS.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(...\"HE5_GCTP_PS\"...) :   %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(...\"Bands\"...) :      %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(...\"HE5_GCTP_GEO\"...) :  %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 the grid file cannot be found, HE5_GDopen will return -1 for the file
   * handle (gdfid).  We there check that this is not the case before
   * proceeding with the other routines.
   *
   * The HE5_GDattach routine returns the handle to the existing grid "Grid1",
   * GDid.  If the grid is not found, HE5_GDattach returns -1 for the handle.
   */
  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(..., \"Pollution\",...) :
%d \n", status);

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

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

```
        GDid2 = HE5_GDattach(gdfid, "PolarGrid");

        status = HE5_GDdeffield(GDid2, "Temperature", "YDim,XDim", NULL,
H5T_NATIVE_FLOAT, 0);
        printf("Status returned by HE5_GDdeffield(..., \"Temperature\",...) :
%d \n", status);

        status = HE5_GDdeffield(GDid2, "Pressure", "YDim,XDim", NULL,
H5T_NATIVE_FLOAT, 0);
        printf("Status returned by HE5_GDdeffield(..., \"Pressure\",...) :
%d \n", status);

        status = HE5_GDdeffield(GDid2, "Soil Dryness", "YDim,XDim", NULL,
H5T_NATIVE_FLOAT, 0);
        printf("Status returned by HE5_GDdeffield(..., \"Soil Dryness\",...) :
%d \n", status);

        status = HE5_GDdeffield(GDid2, "Spectra", "Bands,YDim,XDim", NULL,
H5T_NATIVE_DOUBLE, 0);
        printf("Status returned by HE5_GDdeffield(..., \"Spectra\",...) :
%d \n", status);

        status = HE5_GDsetfillvalue(GDid2, "Pressure", H5T_NATIVE_FLOAT,
&fillval2);
        printf("Status returned by HE5_GDsetfillvalue(...,\"Pressure\",...) :
%d \n", status);

        status = HE5_GDdetach(GDid1);
        status = HE5_GDdetach(GDid2);
    }

  status = HE5_GDclose(gdfid);

  return 0;
}
```

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);
```

```
              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;
}
```

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;
```

```
  float           flt     = -999.;                 /* global attribute */
  float           attr[4] = {-9.9,-9.9,-9.9,-9.9}; /* local 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\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\t %f \n", flt);

              status = HE5_GDreadgrpattr(GDid, "GroupAttribute", grpattr);
              printf("Status returned by HE5_GDreadgrpattr() :  %d \n", status
);
              printf("\tGroup attribute reads: \n");
              for (i = 0; i < 3; i++)
                  printf("\t\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\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.
 */

/* he5_gd_subset */



#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[1] = 59.;  cornerlat[1] = 35.;
              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:
%d\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\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\" \n", 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\n", dimname);
        for (i = 0; i < ndim; i++) printf("dim size: %lu \n", (unsigned
long)Dims[i]);

        ndim = HE5_GDinqdims(GDid2, dimname, Dims);
        printf("Dimension list (PolarGrid): %s\n", dimname);
        for (i = 0; i < ndim; i++) printf("dim size: %lu \n", (unsigned
long)Dims[i]);


        dimsize = HE5_GDdiminfo(GDid1, "Time");
        printf("Size of \"Time\" Array: %lu\n", (unsigned long)dimsize);

        dimsize = HE5_GDdiminfo(GDid2, "Bands");
        printf("Size of \"Bands\" Array: %lu\n", (unsigned long)dimsize);

        status = HE5_GDgridinfo(GDid1, &xdimsize, &ydimsize,upleftpt,
lowrightpt);
        printf("X dim size, Y dim size (UTMGrid): %li %li\n",  xdimsize,
ydimsize);
        printf("Up left pt (UTMGrid): %lf %lf\n", upleftpt[0], upleftpt[1]);
        printf("Low right pt (UTMGrid): %lf %lf\n",  lowrightpt[0],
lowrightpt[1]);

        status = HE5_GDgridinfo(GDid2, &xdimsize, &ydimsize, upleftpt,
lowrightpt);
        printf("X dim size, Y dim size (PolarGrid): %li %li\n",  xdimsize,
ydimsize);
        printf("Up left pt (PolarGrid): %lf %lf\n",  upleftpt[0],
upleftpt[1]);
        printf("Low right pt (PolarGrid): %lf %lf\n",  lowrightpt[0],
lowrightpt[1]);


        status = HE5_GDprojinfo(GDid1, &projcode, &zonecode, &spherecode,
NULL);
        printf("projcode , zonecode (UTMGrid): %d %d\n", projcode, zonecode);
```

```
        printf("spherecode (UTMGrid): %d\n", 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));
        nflds = 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("   %s\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_GDinqattrs(GDid1, NULL, &strbufsize);
        printf("\tNumber of attributes:                    %li \n", n);
        printf("\tSize (in bytes) of attribute list:       %li \n",
strbufsize);
        n = HE5_GDinqattrs(GDid1, attrlist, &strbufsize);
        printf("\tAttribute list:                          %s \n", attrlist);


        printf(" \n");
        printf("Group Attributes \n");
        n = HE5_GDinqgrpattrs(GDid1, NULL, &strbufsize);
        printf("\tNumber of attributes:                    %li \n", n);
        printf("\tSize (in bytes) of attribute list:       %li \n",
strbufsize);
        n = HE5_GDinqgrpattrs(GDid1, attrlist, &strbufsize);
        printf("\tAttribute list:                          %s \n", attrlist);
        printf(" \n");
        printf("Local Attributes \n");
        n = HE5_GDinqlocattrs(GDid1, "Vegetation", NULL, &strbufsize);
        printf("\tNumber of attributes:                    %li \n", n);
        printf("\tSize (in bytes) of attribute list:       %li \n",
strbufsize);
        n = HE5_GDinqlocattrs(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     gdid, gdid2, gdid3
      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"
c     ----------------------------------
      gdfid = he5_gdopen('grid.he5',HE5F_ACC_TRUNC)

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

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

      zonecode   = 40
      spherecode = 0

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

      xdim = 120
      ydim = 200

c     Define GCTP Projection Parameters
c     ---------------------------------
```

```
      do i=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
c     -----------------------
      dim    = 10
      status = he5_gddefdim(gdid, "Time", dim)

c     Create polar stereographic grid
c     -------------------------------

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

      xdim = 100
      ydim = 100

      spherecode = 3

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

c     Use default boundaries for Polar Stereographic (hemisphere)
c     -----------------------------------------------------------
      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
c     ------------------------
      dim    = 3
      status = he5_gddefdim(gdid2, "Bands", dim)

c     Create geographic (linear scale) grid
c     -------------------------------------

c ---------------------------------------------------- c
c   0-15 degrees longitude, 20-30 degrees latitude    c
c ---------------------------------------------------- c

      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,
     1projparm)

c     Detach from the created grids
c     -----------------------------
      status = he5_gddetach(gdid)
      status = he5_gddetach(gdid2)
      status = he5_gddetach(gdid3)

c     Finally, close the grid file
c     ----------------------------
      status = he5_gdclose(gdfid)

      stop
      end
```

Example 2

```
c  In this example we open an HDF-EOS file, attached to
c   the specified grids, define fields, and set fill
c     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_NATIVE_DOUBLE
      parameter (HE5T_NATIVE_DOUBLE=11)


        fillval1 = -7.0
        fillval2 = -9999.0
```

```
c     Open HDF-EOS file "grid.he5"
c     ----------------------------
        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_gdsetfill(gdid1, "Pollution", HE5T_NATIVE_FLOAT,
     1fillval1)

      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
c    Polar grid and write data to the “Temperature” field.


      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)


c     Create data buffers
c     -------------------
        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

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

c        Attach to the UTM grid
c        ----------------------
             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           Write data to the field "Vegetation"
c           ------------------------------------
                  status = he5_gdwrfld(gdid,"Vegetation",start,
     1      stride,count,veg)
                  f        = 1
```

```
                  count(1) = 1

c           Write global attribute "float"
c           ------------------------------
                  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

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

```
c In this example we open HDF-EOS grid file, attach to the UTM
c    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)

      real*4        f
      real*4        veg(120,200)

      integer       HE5F_ACC_RDWR
      parameter     (HE5F_ACC_RDWR=100)


c     Open HDF-EOS "grid.h5" file
c     ---------------------------
        gdfid = he5_gdopen("grid.he5", HE5F_ACC_RDWR)
        if (gdfid .ne. FAIL) then

c        Attach to the UTM grid
c        ----------------------
             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
c           -------------------------------------
                  status =
he5_gdrdfld(gdid,"Vegetation",start,stride,count,veg)

c           Read global attribute "float"
c           -----------------------------
                  status = he5_gdrdattr(gdid, "float", f)
                  write(*,*) 'global attribute value: ', f
             endif
        endif

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

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

        stop
        end
```

Example 5

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

      program      he5_gd_subsetF_32
```

```
      integer      status
      integer      gdfid
      integer      gdid
      integer      rgid
      integer      he5_gdopen
      integer      he5_gdattach
      integer      he5_gdreginfo
      integer      he5_gddefboxreg
      integer      he5_gdextreg
      integer      he5_gddetach
      integer      he5_gdclose
      integer      rk
      integer      nt

      integer*4    dims(8)
      integer*4    size

      real*8       cornerlon(2)
      real*8       cornerlat(2)
      real*8       upleft(2)
      real*8       lowright(2)

      real*4       datbuf(100*100)

      integer      HE5F_ACC_RDWR
      parameter    (HE5F_ACC_RDWR=100)


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

c        Attach to the POLAR grid
c        ------------------------
         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
c           -----------------
            rgid   = he5_gddefboxreg(gdid,cornerlon, cornerlat)

c           Get region information
c           ----------------------
            status = he5_gdreginfo(gdid,rgid,"Temperature",nt,
     1rk, dims, size, upleft, lowright)
            write(*,*) dims(1), dims(2), rk, nt

c           Extract region
c           --------------
            status = he5_gdextreg(gdid,rgid,"Temperature",datbuf)
         endif

c        Detach from the grid
```

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

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

      endif
      stop
      end
```

Example 6

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

      program       he5_gd_infoF_32

      integer       status
      integer       i
      integer       he5_gdopen
      integer       he5_gdattach
      integer       he5_gdfldinfo
      integer       he5_gdprojinfo
      integer       he5_gdgridinfo
      integer       he5_gdinqflds
      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_gdinqdims
      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
c     ----------------------------
      gdfid = he5_gdopen('grid.he5', HE5F_ACC_RDWR)
      if (gdfid .ne. FAIL) then

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

c        Inquire dimensions
c        ------------------
             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

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

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

c        Get grid parameters
c        -------------------
             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),
     2lowrightpt(2)

c        Get projection parameters
c        -------------------------
            status = he5_gdprojinfo(gdid1,projcode,zonecode,
     1spherecode,projparm)
            write(*,*) 'projcode,zonecode (UTMGrid): ',projcode,
     2zonecode
            write(*,*) 'spherecode (UTMGrid): ', spherecode

            status = he5_gdprojinfo(gdid2,projcode,zonecode,
     1spherecode,projparm)
          write(*,*) 'projcode (PolarGrid): ', projcode
            write(*,*) 'spherecode (PolarGrid): ', spherecode
            do i=1,13
                 write(*,*) 'Projection Parameter: ',i,projparm(i)
            enddo

c        Get information about fields
c        ----------------------------
            nflds = he5_gdinqflds(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_gdinqflds(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,
     1dimname,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

c     Get number of grid dimensions and dim. list length
c     --------------------------------------------------
      n = he5_gdnentries(gdid1, HE5_HDFE_NENTDIM, strbufsize)
      write(*,*) 'Number of dimension entries (UTMGrid): ', n
      write(*,*) 'Length of Dimension List (UTMGrid): ',strbufsize

c     Get number of data fields and field list length
c     -----------------------------------------------
      n = he5_gdnentries(gdid1, HE5_HDFE_NENTDFLD, strbufsize)
      write(*,*) 'Number of data fields (UTMGrid): ', n
```

```
      write(*,*) 'Length of Field List (UTMGrid): ',strbufsize

      endif

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
```

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);

  /* 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, "Bands", 15);
  printf("Status returned by HE5_ZAdefdim():      %d \n", status);

  status = HE5_ZAdefdim(ZAid, "IndxTrack", 12);
  printf("Status returned by HE5_ZAdefdim():      %d \n", status);

  /* Define "External" Dimension */
  /* --------------------------- */
  status = HE5_ZAdefdim(ZAid, "ExtDim", 60);
  printf("Status returned by HE5_ZAdefdim():      %d \n", status);

  /* Define Unlimited Dimension */
  /* -------------------------- */
  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);

  /* Close the za file */
  /* ----------------- */
  status = HE5_ZAclose(zafid);
  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);


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

              /*           First, define chunking          */
              /* (the appendable dataset must be chunked)  */
              /* ----------------------------------------  */
              chunk_dims[0] = 15;
              chunk_dims[1] = 40;
              chunk_dims[2] = 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, "Spectra", "Bands,Res2tr,Res2xtr", NULL,
              H5T_NATIVE_FLOAT);
                 printf("Status returned by HE5_ZAdefine(...\"Spectra\",...) :
%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(...\"Count\",...) :
%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   */

  double          attr6[4] = {1.111111,2.222222,3.333333,4.444444};/* 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 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;
              start[2] = 0;      count[2] = 20;

              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_ZAwritelocattr(ZAid, "Density", "LocalAttribute",
H5T_NATIVE_INT, count, attr3);
```

```
               printf("status returned by
HE5_ZAwritelocattr(\"LocalAttribute\"):%d\n", status);


      }
      }

  status = HE5_ZAdetach(ZAid);
  status = HE5_ZAclose(zafid);

  return 0;
}
```

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;
              count[0] = 15;   count[1] = 40;  count[2] = 20;
              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("\t\t %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("\t\t %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("\t\t %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("Global attribute values:\n");
              printf("\t\t %s \n",attr4);

              /* Read Global 'long' Attribute */
              /* ---------------------------- */
              status = HE5_ZAreadattr(ZAid, "GLOBAL_LONG_ATTR", attr5);
              printf("Status returned by HE5_ZAreadattr() :     %d \n",
status);
              printf("Global attribute values:\n");
              for (i = 0; i < 4; i++)
                  printf("\t\t %li \n",attr5[i]);

              /* Read Global 'double' Attribute */
              /* ------------------------------ */
```

```
                status = HE5_ZAreadattr(ZAid, "GLOBAL_DOUBLE_ATTR", attr6);
                printf("Status returned by HE5_ZAreadattr() :     %d \n", status);
                printf("Global attribute values:\n");
                for (i = 0; i < 4; i++)
                    printf("\t\t %f \n",attr6[i]);

            }
        }

  status = HE5_ZAdetach(ZAid);
  status = HE5_ZAclose(zafid);

  return 0;
}
```

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 */
      /* ------------------- */
      nflds = HE5_ZAnentries(ZAid, HE5_HDFE_NENTDFLD, &strbufsize);
      rank = (int *) calloc(nflds, sizeof(int));
      fieldlist = (char *) calloc(strbufsize + 1, 1);
      nflds = HE5_ZAinquire(ZAid, 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 "MyTrack1" dim */
      /* -------------------------- */
      dimsize = HE5_ZAdiminfo(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\t Data type:          %d\n", dtype);
      printf("\t\t Number of elements: %lu \n", (unsigned long)nelem);

      nelem = 0;
      dtype = FAIL;
      /* Get info about Group Attributes */
      /* ------------------------------- */
      printf("Group Attribute:\n");
      status = HE5_ZAgrpattrinfo(ZAid,"GroupAttribute",&dtype,&nelem);
      printf("\t\t Data type:          %d\n", dtype);
      printf("\t\t Number of elements: %lu \n", (unsigned long)nelem);

      nelem = 777;
      dtype = FAIL;
      /* Get info about Local Attributes */
      /* ------------------------------- */
```

```
                printf("Local Attribute:\n");
                status = HE5_ZAlocattrinfo(ZAid,"Density",
"LocalAttribute",&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_ZAinqattrs(ZAid, NULL, &strbufsize);
                printf("\t\t Number of attributes:             %li \n", nattr);
                printf("\t\t String length of attribute list:  %li \n", strbufsize);
                n = HE5_ZAinqattrs(ZAid, attrlist, &strbufsize);
                printf("\t\t Attribute list:                   %s \n", attrlist);

                /* Inquire Group Attributes */
                /* ------------------------ */
                strbufsize = 0;
                printf("\n");
                printf("Group Attributes:\n");
                nattr = HE5_ZAinqgrpattrs(ZAid, 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_ZAinqgrpattrs(ZAid, attrlist, &strbufsize);
                printf("\t\t Attribute 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 Number of attributes:             %li \n", nattr);
                printf("\t\t String length of attribute list:  %li \n", strbufsize);
                strcpy(attrlist,"");
                nattr = HE5_ZAinqlocattrs(ZAid, "Density", attrlist, &strbufsize);
                printf("\t\t Attribute list:                   %s \n", attrlist);


            }
    }

  status = HE5_ZAdetach(ZAid);
  status = HE5_ZAclose(zafid);

  return 0;
}
```

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;
  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);
      }


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

  /* Close the file */
  /* -------------- */
  status = HE5_ZAclose(fid);
  printf("Status returned by HE5_ZAclose() :         %d \n", status);

  return(0);

}
```

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
!     ==========================================================

      program    he5_za_setupF_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
```

```
!     Open the HDF-EOS file, "za.he5" using "TRUNC" file access code
!     ---------------------------------------------------------------
      zafid = he5_zaopen('za.he5',HE5F_ACC_TRUNC)
      write(*,*) 'File ID returned by he5_zaopen():  ',zafid

!     Create the za, "ZA1", within the file
!     -------------------------------------
      zaid = he5_zacreate(zafid, "ZA1")
      write(*,*) 'ZA ID returned by he5_zacreate():  ',zaid

!     Define Data dimensions
!     ----------------------
      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
!     ---------------------------
      extdata = 60
      status = he5_zadefdim(zaid, "ExtDim", extdata)
      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
```

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_za_definefieldsF_32

      implicit    none

      include     'hdfeos5.inc'

      integer     status
      integer     he5_zaopen
      integer     he5_zaattach
      integer     he5_zadefine
      integer     he5_zadetach
      integer     he5_zaclose
      integer     zafid, zaid

      integer     FAIL
      parameter   (FAIL=-1)

!     Open the HDF-EOS file, "za.he5" using "READ/WRITE" access code
!     --------------------------------------------------------------
      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
         if (zaid .NE. FAIL) then

!     Define Data fields
!     ------------------

            status = he5_zadefine(zaid, "Density",
     1           "MyTrack1"," ", HE5T_NATIVE_FLOAT)
            write(*,*) 'Status returned by he5_zadefine():  ',status

            status = he5_zadefine(zaid, "Temperature",
     1           "MyTrack2,MyTrack1",
     2           " ", HE5T_NATIVE_FLOAT)
            write(*,*) 'Status returned by he5_zadefine():  ',status

            status = he5_zadefine(zaid, "Pressure",
     1           "Res2xtr,Res2tr",
     2           " ", HE5T_NATIVE_FLOAT)
            write(*,*) 'Status returned by he5_zadefine():  ',status

            status = he5_zadefine(zaid, "Spectra",
     1           "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
      end
```

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

      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

              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,
     1              stride,count,plane)
                 enddo
      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,
     1           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
!     ==========================================================

      program            he5_za_readdataF_32

      implicit           none

      include            'hdfeos5.inc'

      integer            status
      integer            he5_zaopen
      integer            he5_zaattach
      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
              zaid = he5_zaattach(zafid, "ZA1")
              write(*,*) 'ZA ID returned by he5_zaattach():  ',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
```

```
                                plane((j-1)*20+k) = (j-1)*100 + i-1
                                  enddo
                            enddo
                            status = he5_zaread(zaid,"Spectra",start,
     1              stride,count,plane)
                        enddo


!     Read Attribute
!     --------------
                    status = he5_zardattr(zaid, "TestAttr", attr)
                    do i=1,4
                       write(*,*) 'Attribute Element', i, ':', attr(i)
                    enddo

              endif
        endif

!    Detach from the za
!    ------------------
        status = he5_zadetach(zaid)
      write(*,*) 'Status returned from he5_zadetach:  ',status

!     Close the file
!     --------------
        status = he5_zaclose(zafid)
      write(*,*) 'Status returned from he5_zaclose:  ',status

        stop
        end
```

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_zaopen("za.he5", HE5F_ACC_RDONLY)
      write(*,*) 'File ID returned by he5_zaopen():  ',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,
     1         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
```

This page intentionally left blank.

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 TOOLKITcan 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 5.2.20 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. Otherwiase 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[PGSd_MET_FILE_PATH_MAX]="";
    char                      fileName2[PGSd_MET_FILE_PATH_MAX]="";
    char                      my_HDF_file[PGSd_MET_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];

    /*******************************************************************/
    /* 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");
    }

    /*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,"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],
```

```
              "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 ARCHIVEDMETADAT 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,"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],
              "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);

/* 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 ARCHIVEDMETADAT 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;
}
```

8.3 The Metadata Configuration File (MCF) for Codes 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 ESDT's INVENTORYMETADATA section in  */
/*  order to generate an ESDT-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   */
/*  ECS Project' {420-TP-016-001).                               */
/*                                                               */
/*  This MCF file's contents were based on the ESDT Descriptor   */
/*  file template Ver-1.6, 3/31/97.                              */
/*                                                               */
/*****************************************************************/
/*****************************************************************/


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
END_GROUP = MeasuredParameter

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 = StartOrbitNumber
                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 ESDT's INVENTORYMETADATA section in */
/*  order to generate an ESDT-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 */
/*  ECS Project' {420-TP-016-001). */
/* */
/*  This MCF file's contents were based on the ESDT Descriptor */
/*  file template Ver-1.6, 3/31/97. */
/* */
/*****************************************************************/
/*****************************************************************/

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


  /*  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
```

```
      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               = BOUNDINGRECTANGLE

    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
          NUM_VAL               = 1
          CLASS                 = M
          VALUE                 = "NOT SET"
        END_OBJECT              = PARAMETERVALUE

      END_GROUP               = INFORMATIONCONTENT

    END_OBJECT              = ADDITIONALATTRIBUTESCONTAINER

  END_GROUP               = ADDITIONALATTRIBUTES


    GROUP                   = ORBITPARAMETERSGRANULE

    OBJECT                  = ORBITALPARAMETERSPOINTER
      NUM_VAL               = 1
      VALUE                 = "NOT SET"
    END_OBJECT              = ORBITALPARAMETERSPOINTER

  END_GROUP               = ORBITPARAMETERSGRANULE


    GROUP                   = STORAGEMEDIUMCLASS

    OBJECT                  = STORAGEMEDIUM
      NUM_VAL               = 10
      VALUE                 = "NOT SET"
    END_OBJECT              = STORAGEMEDIUM

  END_GROUP               = STORAGEMEDIUMCLASS

END_GROUP               = INVENTORYMETADATA
END
```

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
10401|utcpole.dat|/tk/TOOLKIT_MTD/database/common/CSC/utcpole.dat
10402|earthfigure.dat|/tk/TOOLKIT_MTD/database/common/CSC/earthfigure.dat
10601|de200.eos|/tk/TOOLKIT_MTD/database/common/CBP/de200.eos
10801|sc_tags.dat|/tk/TOOLKIT_MTD/database/common/EPH/sc_tags.dat
10302|udunits.dat|/tk/TOOLKIT_MTD/database/common/CUC/udunits.dat
#################################################################
# Logical IDs assigned for input/output files can be changed BUT they
# should be diffrent from the IDs assigned above.
##################################################################
10250|MCF_File|/tk/TOOLKIT_MTD/test/test_MET_HDF5/MET_TestData/MCF_File
10251|data_dict|/tk/TOOLKIT_MTD/test/test_MET_HDF5/MET_TestData/data_dict
5039|Swath_h5 hdf|/tk/TOOLKIT_MTD/test/test_MET_HDF5/MET_TestData/Swath_h5 hdf
#################################################################
# files to check PGS_MET_InitNonMCF function
#################################################################
5804|NAT_File|/tk/TOOLKIT_MTD/test/test_MET_HDF5/MET_TestData/NAT_File
#################################################################
# End this table with next two lines. Last line should be ?
##################################################################
0|DUMMY|/tk/TOOLKIT_MTD/test/test_MET_HDF5/MET_TestData/DUMMY?
```

8.5.2 The PCF file used for example in Section 8.2.2

This example is used by the SDP Toolkit:

```
#  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
#   ny 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.
!  ~/runtime
#
# ----------------------------------------------------------------------
# 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.
# ----------------------------------------------------------------------
10200|nad27sp|~/database/common/GCT||||1
10201|nad83sp|~/database/common/GCT||||1
# ----------------------------------------------------------------------
# file for Constant & Unit Conversion (CUC) tools
# IMPORTANT NOTE: THIS FILE WILL BE SUPPLIED AFTER TK4 DELIVERY!
# ----------------------------------------------------------------------
10999|PGS_CUC_maths_parameters|~/database/common/CUC||||1
#
#
#----------------------------------------------------------------------
# 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)
#----------------------------------------------------------------------
10250|MCF|||||1
10251|data_dict|||||1
10252|GetAttr.temp|||||1
10254|MCFWrite.temp|||||1
#
#
#
?   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
!  ~/runtime
#
#---------------------------------------------------------------------
# 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.
#
#---------------------------------------------------------------------
102|asciidump|||||1
5039|Swath_h5.hdf|~/runtime|||Swath_h5.hdf|1
100|NAT_file|~/runtime||||1
# ---------------------------------------------------------------------
#
?   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 ephemeris data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the ephemeris data quality flag that
# should be considered fatal (i.e. the ephemeris data associated
# with the quality flag should be REJECTED/IGNORED).
# ----------------------------------------------------------------------
10507|ephemeris data quality flag mask|65536
#
# ----------------------------------------------------------------------
# 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).
# ----------------------------------------------------------------------
10508|attitude data quality flag mask|65536
#
# ----------------------------------------------------------------------
# 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)
# ----------------------------------------------------------------------
10911|ECS DEBUG; 0=normal, 1=debug|0
#
# ----------------------------------------------------------------------
# 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 otain this information. (DO NOT REMOVE THIS ENTRY)
# ----------------------------------------------------------------------
10099|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 9-1. Summary of the Zonal Average Interface (1 of 2)

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Access	HE5_ZAopen	he5_zaopen	Opens or creates HDF file in order to create, read, or write a zonal average	2-336
	HE5_ZAcreate	he5_zacreate	Creates a zonal average within the file	2-298
	HE5_ZAattach	he5_zaattach	Attaches to an existing zonal average within the file	2-292
	HE5_ZAdetach	he5_zadetach	Detaches from zonal average interface	2-307
	HE5_ZAclose	he5_zaclose	Closes file	2-296
Definition	HE5_ZAdefdim	he5_zadefdim	Defines a new dimension within the zonal average	2-304
	HE5_ZAdefine	he5_zadefine	Defines a new data field within the zonal average	2-306
	HE5_ZAdefcomp	he5_zadefcomp	Defines a field compression scheme	2-302
	HE5_ZAdefchunk	he5_zadefchunk	Define chunking parameters	2-299
	HE5_ZAdefcomchunk	he5_zadefcomch	Defines compression with automatic chunking	2-300
	HE5_ZAsetalias	he5_zasetalias	Defines alias for data field	2-346
	HE5_ZAdropalias	he5_zadrpalias	Removes alias from the list of field aliases	2-309
	HE5_ZAfldrename	he5_zafldrnm	Changes the field name	2-311
Basic I/O	HE5_ZAwrite	he5_zawrite	Writes data to a zonal average field	2-352
	HE5_ZAread	he5_zaread	Reads data from a zonal average field	2-337
	HE5_ZAwriteattr	he5_zawrattr	Writes/updates attribute in a zonal average	2-354
	HE5_ZAreadattr	he5_zardattr	Reads attribute from a zonal average	2-339
	HE5_ZAwritegrpattr	he5_zawrgattr	Writes/updates group attribute in a zonal average	2-360
	HE5_ZAwritelocattr	he5_zawrlattr	Writes/updates group attribute in a zonal average	2-362
	HE5_ZAwritedatameta	he5_zawrdmeta	Writes field metadata for an existing zonal average data field	2-356
	HE5_ZAreadgrpattr	he5_zardgattr	Reads attribute from a zonal average	2-344
	HE5_ZAreadlocattr	he5_zardlattr	Reads attribute from a zonal average	2-345
	HE5_ZAsetfillvalue	he5_zasetfill	Sets fill value for the specified field	2-350
	HE5_ZAgetfillvalue	he5_zagetfill	Retrieves fill value for the specified field	2-316
	HE5_ZAaliasinfo	he5_zaaliasinfo	Retrieves information about field aliases	2-291
	HE5_ZAgetaliaslist	he5_zagetaliaslist	Retriews list and number of aliases in a data group	2-312
	HE5_ZAinqdims	he5_zainqdims	Retrieves information about dimensions defined in zonal average	2-323

Table 9-1. Summary of the Zonal Average Interface (2 of 2)

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Inquiry	HE5_ZAinquire	he5_zainquire	Retrieves info about the data fields defined	2-331
	HE5_ZAinqattrs	he5_zainqattrs	Retrieves number and names of attributes defined	2-320
	HE5_ZAinqdatatype	he5_zaidtype	Returns data type information about specified fields in a zonal average	2-321
	HE5_ZAinqfldalias	he5_zainqfldalias	Returns information about data fields & aliases defined in a zonal average	2-326
	HE5_ZAchunkinfo	he5_zachunkinfo	Retrieves chunking information	2-294
	HE5_ZAinqgrpattrs	he5_zainqgattrs	Retrieves information about group attributes defined in zonal average	2-328
	HE5_ZAinqlocattrs	he5_zainqlattrs	Retrieves information about local attributes defined in zonal average	2-329
	HE5_ZAlocattrinfo	he5_zalocattrinfo	Returns information about a data field's local attribute(s)	2-333
	HE5_ZAlocattrinfo2	he5_zalocattrinfo2	Same as above, but also returns buffer size of attribute element	2-333
Inquiry	HE5_ZAnentries	he5_zanentries	Returns number of entries and descriptive string buffer size for a specified entity	2-335
	HE5_ZAdiminfo	he5_zadiminfo	Retrieves size of specified dimension	2-308
	HE5_ZAattrinfo	he5_zaattrinfo	Returns info about zonal average attributes	2-293
	HE5_ZAattrinfo2	he5_zaattrinfo2	Same as above, but also returns buffer size of attribute element	2-293
	HE5_ZAgrpattrinfo	he5_zagattrinfo	Returns information about a zonal average group attribute	2-317
	HE5_ZAgrpattrinfo2	he5_zagattrinfo2	Same as above, but also returns buffer size of attribute element	2-137
	HE5_ZAinfo	he5_zainfo	Retrieves information about a specific data field	2-318
	HE5_ZAcompinfo	he5_zacompinfo	Retrieves compression information about a field	2-297
	HE5_ZAinqza	he5_zainqza	Retrieves number and names of zas in file	2-332
External Files	HE5_ZAmountexternal	Not available	Mounts external data file	2-334
	HE5_ZAreadexternal	Not available	Reads external data set	2-343
	HE5_ZAunmount	Not available	Dismounts external data file	2-351
External Data Sets	HE5_ZAsetextdata	he5_zasetxdat	Sets external data set	2-349
	HE5_ZAgetextdata	he5_zagetxdat	Gets external data set	2-315
Dimension Scale	HE5_ZAsetdimscale	he5_zasetdimscale	Sets dimension scale for a field dimension within the ZA	2-347
	HE5_ZAdefdimscale	he5_zadefdimscale	Sets dimension scale for a dimension of all fields within the ZA	2-347
	HE5_ZAgetdimscale	he5_zagetdimscale	Gets dimension scale for a field dimension within the ZA	2-313
	HE5_ZAwritedscaleattr	he5_zawritedscaleattr	Writes/Updates a dimension scale attribute in a specific ZA	2-357
	HE5_ZAreaddscaleattr	he5_zareaddscaleattr	Reads a dimension scale attribute from a specific dimension	2-340
	HE5_ZAinqdscaleattrs	he5_zainqdscaleattrs	Retrieve information about the attributes defined for a specific dimension scale	2-324
	HE5_ZAdscaleattrinfo	he5_zadscaleattrinfo	Returns information about attribute(s) in a specific dimension scale	2-310
	HE5_ZAdscaleattrinfo2	he5_zadscaleattrinfo2	Same as above, but also returns buffer size of attribute element	2-310

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 */
```

This page intentionally left blank.

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 5-1.8.19 on your host. You may also need to install jpeg-v9b, zlib-1.2.11 and szip-2.1.1 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-EOSv5.1.16.tar.Z to the target directory by

typing the command:
cp HDF-EOSv5.1.16.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-EOS5.1.16.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 (Sction 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_dbug	thread-safe version with enabled debug statements

Also, the user should specify the architecture options for the SGI Power Challenge platform, Linux and Mac. On the SGI Challenge, Linux and Mac, 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:

binary format	architecture	<install-option>
new 32-bit	sgi32	-sgi32
64 bit	sgi64	-sgi64

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 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 hdf5-1.8.19 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:
and/or
Current value of the HDF include directory is: <path>
Accept [y]/n:

Make sure to type 'n' and hit return, if the defaults do not point to the correct directories. The script will then prompt for the new values.

3.3) The installation script will finish with the following message:

HDFEOS installation ending at <date/time>

3.4) (optional - see 3.0)

If you ran the Unix 'script' command to create a log file, then type 'exit' and hit return at the command prompt. This will exit the sub-shell stated by 'script' and save the log file.

Hint: The log file generated by the script command may contain 'hard return' characters at the end of each line. These appear in some text editors as "^M". They can be removed with the following command:

```
sed 's/.$//' <logfile-name> > <logfile-name>.new
```

where <logfile-name> is the name of the log file.

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 hdfeos5 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          OR
$ 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 tradionally 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 = ($path)                                      # save path
source <HDFEOS-HOME-DIR>/bin/$BRAND/hdfeos_env.csh  # HDFEOS setup
set path = ($my_path $hdfeos_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.

1B) Korn shell (ksh) Users:

Edit the HDFEOS user's .profile file to include the following line:

```
. <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:

```
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.

Platform	OS	Version	C Compiler	FORTRAN77
Sun Sparc	Solaris	5.11	gcc	gfortran
Intel Mac 32-bit	Darwin	14.5.0	gcc 4.2.1	gfortran 5.2.0
Intel Mac 64-bit	Darwin	14.5.0	gcc 4.2.1	gfortran 5.2.0
Linux 32-bit	Red Hat Linux 4.4.7-18	2.6.32-358.2.1.e16.x86_64 #1 SMP	gcc 4.4.7	Gfortran 4.4.7
Linux 64-bit	Red Hat Linux 4.4.7-18	2.6.32-358.2.1.e16.x86_64 #1 SMP	gcc 4.4.7	Gfortran 4.4.7
Windows	7, 10	MS VS .net 2008	Visual c++	Intel Visual Fortran 11.1
Cygwin	Cygwin	1.7.29	gcc 4.8.2	gfortran 4.8.2

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. Also with 2.16 release we did not test HDF-EOS5 in Sun platforms.

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-EOS5),:

$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, Sun5.11	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

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 User Account Setup section of the installation instructions. They are listed here for reference:

name	value	description
HDFEOS5_HOME	<install-path>/hdfeos5	top-level directory
(where <install-path> is the absolute directory path above hdfeos5)		
HDFEOS5_BIN	$HDFEOS5_HOME/bin/$BRAND	executable files
HDFEOS5_INC	$HDFEOS5_HOME/include	header files
HDFEOS5_LIB	$HDFEOS5_HOME/lib/$BRAND	library files
HDFEOS5_OBJ	$HDFEOS5_HOME/obj/$BRAND	object files
HDFEOS5_SRC	$HDFEOS5_HOME/src	source files

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:

name	description
CC	C compiler
CFLAGS	default C flags (optimize, ANSI)
C_CFH	C w/ cfortran.h callable from FORTRAN
CFHFLAGS	CFLAGS + C_CFH
C_F77_CFH	C w/ cfortran.h calling FORTRAN
C_F77_LIB	FORTRAN lib called by C main
F77	FORTRAN compiler
F77FLAGS	common FORTRAN flags
F77_CFH	FORTRAN callable from C w/ cfortran.h
F77_C_CFH	FORTRAN calling C w/ cfortran.h
CFH_F77	same as F77_C_CFH
F77_C_LIB	C lib called by FORTRAN main

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:

THIS PACKAGE, I.E. CFORTRAN.H, THIS DOCUMENT, AND THE CFORTRAN.H EXAMPLEPROGRAMS ARE PROPERTY OF THE AUTHOR WHO RESERVES ALL RIGHTS. THIS PACKAGE ANDTHE CODE IT PRODUCES MAY BE FREELY DISTRIBUTED WITHOUT FEES, SUBJECT TO THEFOLLOWING RESTRICTIONS:

- YOU MUST ACCOMPANY ANY COPIES OR DISTRIBUTION WITH THIS (UNALTERED) NOTICE.
- YOU MAY NOT RECEIVE MONEY FOR THE DISTRIBUTION OR FOR ITS MEDIA (E.G. TAPE, DISK, COMPUTER, PAPER.)
- YOU MAY NOT PREVENT OTHERS FROM COPYING IT FREELY.
- YOU MAY NOT DISTRIBUTE MODIFIED VERSIONS WITHOUT CLEARLY DOCUMENTING YOUR CHANGES AND NOTIFYING THE AUTHOR.
- YOU MAY NOT MISREPRESENTED THE ORIGIN OF THIS SOFTWARE, EITHER BY EXPLICIT CLAIM OR BY OMISSION.

THE INTENT OF THE ABOVE TERMS IS TO ENSURE THAT THE CFORTRAN.H PACKAGE NOT BEUSED FOR PROFIT MAKING ACTIVITIES UNLESS SOME ROYALTY ARRANGEMENT IS ENTERED INTO WITH ITS AUTHOR.

THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SOFTWARE IS WITH YOU. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. THE AUTHOR IS NOT RESPONSIBLE FOR ANY SUPPORT OR SERVICE OF THE CFORTRAN.H PACKAGE.

Burkhard Burow

burow@vxdesy.cern.ch

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-EOS5.1.16_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:

   – RVL_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 data base (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):

4) In addition to the email response mechanism, a phone answering machine is also provided. The telephone number is: 301–851–8122. Calls will be returned as soon as possible. We note, however, that email is our preferred method of responding to users. Also a list server has also been set up for users. The address is: SDPToolkit ListsServer.gsfc.nasa.gov.

Abbreviations and Acronyms

AI&T	algorithm integration & test
AIRS	Atmospheric Infrared Sounder
API	application program interface
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer
CCSDS	Consultative Committee on Space Data Systems
CDRL	Contract Data Requirements List
CDS	CCSDS day segmented time code
CERES	Clouds and Earth Radiant Energy System
CM	configuration management
COTS	commercial off–the–shelf software
CUC	constant and unit conversions
CUC	CCSDS unsegmented time code
DAAC	distributed active archive center
DBMS	database management system
DCE	distributed computing environment
DCW	Digital Chart of the World
DEM	digital elevation model
DTM	digital terrain model
ECR	Earth centered rotating
ECS	EOSDIS Core System
EDC	Earth Resources Observation Systems (EROS) Data Center
EDHS	ECS Data Handling System
EDOS	EOSDIS Data and Operations System
EOS	Earth Observing System
EOSAM	EOS AM Project (morning spacecraft series)
EOSDIS	Earth Observing System Data and Information System
EOSPM	EOS PM Project (afternoon spacecraft series)

ESDIS	Earth Science Data and Information System (GSFC Code 505)
FDF	flight dynamics facility
FOV	field of view
ftp	file transfer protocol
GCT	geo–coordinate transformation
GCTP	general cartographic transformation package
GD	grid
GPS	Global Positioning System
GSFC	Goddard Space Flight Center
HDF	hierarchical data format
HEG	HDF-EOS to GeoTIFF Conversion Tool
HITC	Hughes Information Technology Corporation
http	hypertext transport protocol
I&T	integration & test
ICD	interface control document
IDL	interactive data language
IP	Internet protocol
IWG	Investigator Working Group
JPL	Jet Propulsion Laboratory
LaRC	Langley Research Center
LIS	Lightening Imaging Sensor
M&O	maintenance and operations
MCF	metadata configuration file
MET	metadata
MODIS	Moderate–Resolution Imaging Spectroradiometer
MSFC	Marshall Space Flight Center
NASA	National Aeronautics and Space Administration
NCSA	National Center for Supercomputer Applications
netCDF	network common data format
NGDC	National Geophysical Data Center

NMC	National Meteorological Center (NOAA)
ODL	object description language
PC	process control
PCF	process control file
PDPS	planning & data production system
PGE	product generation executive (formerly product generation executable)
POSIX	Portable Operating System Interface for Computer Environments
PT	point
QA	quality assurance
RDBMS	relational data base management system
RPC	remote procedure call
RRDB	recommended requirements database
SCF	Science Computing Facility
SDP	science data production
SDPF	science data processing facility
SGI	Silicon Graphics Incorporated
SMF	status message file
SMAP	Soil Moisture Active Passive
SMP	Symmetric Multi–Processing
SOM	Space Oblique Mercator
SPSO	Science Processing Support Office
SSM/I	Special Sensor for Microwave/Imaging
SW	swath
TAI	International Atomic Time
TBD	to be determined
TDRSS	Tracking and Data Relay Satellite System
THG	The HDF Group
TRMM	Tropical Rainfall Measuring Mission (joint US – Japan)
UARS	Upper Atmosphere Research Satellite
UCAR	University Corporation for Atmospheric Research

URL	universal reference locator
USNO	United States Naval Observatory
UT	universal time
UTC	Coordinated Universal Time
UTCF	universal time correlation factor
UTM	universal transverse mercator
VPF	vector product format
WWW	World Wide Web
ZA	Zonal Average