170-TP-510-001

HDF-EOS Library User's Guide for the ECS Project, Volume 1: Overview and Examples

Technical Paper

Technical Paper--Not intended for formal review or government approval.

February 2000

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

Larry Klein /s/ 2/21/00

David Wynne, Abe Taaheri Date
Xin-Min Hua, Ray Milburn, and Larry Klein
EOSDIS Core System Project

SUBMITTED BY

Mary S. Armstrong /s/ 2/21/00

Mary S. Armstrong, Director of Development Date
EOSDIS Core System Project

Raytheon Systems Company
Upper Marlboro, Maryland

This page intentionally left blank.

Preface

This document is a Users Guide for HDF-EOS (Hierarchical Data Format - Earth Observing System) library tools. HDF is the scientific data format standard selected by NASA as the baseline standard for EOS. This Users Guide accompanies Version 2 software, which is available to the user community on the EDHS1 server. These 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 grid, point or swath structures, which are implemented in the current version of the library. Services based on geolocation information will be built on HDF-EOS structures. Producers of products not covered by these structures, e.g. non-geolocated data, can use the standard HDF libraries.

In the ECS (EOS Core System) production system, the HDF-EOS library will be used in conjunction with SDP (Science Data Processing) Toolkit software. The primary tools used in conjunction with HDF-EOS library will be those for metadata handling, process control and status message handling. Metadata tools will be used to write ECS inventory and granule specific metadata into HDF-EOS files, while the process control tools will be used to access physical file handles used by the HDF tools. (*SDP Toolkit Users Guide for the ECS Project, May, 1999, 333-CD-100-003).*

HDF-EOS is an extension of NCSA (National Center for Supercomputing Applications) HDF and uses HDF library calls as an underlying basis. Version 4.1r3 of HDF is used.The library tools are written in the C language and a Fortran interface is provided. The current version contains software for creating, accessing and manipulating Grid, Point and Swath structures. This document includes overviews of the interfaces, and code examples. EOSView, the HDF-EOS viewing tool, has been revised to accommodate the current version of the library.

Technical Points of Contact within EOS are:

Larry Klein, larry@eos.hitc.com
David Wynne, davidw@eos.hitc.com
Doug Ilg, dilg@ gsfc.nasa.gov

An email address has been provided for user help:

pgstlkit@eos.hitc.com

Any questions should be addressed to:

Data Management Office
The ECS Project Office
Raytheon Systems Company
1616 McCormick Drive
Upper Marlboro, MD 20774-5301

This page intentionally left blank.

Abstract

This document will serve as the user's guide to the HDF-EOS file access library. 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, and Grid – including overviews of the interfaces, and code examples. This document should be suitable for use by data producers and data users alike.

***Keywords**:* HDF-EOS, Metadata, Standard Data Format, Standard Data Product, Disk Format, Point, Grid, Swath, 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

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 4-4

4.3.2 File Identifiers 4-5
4.3.3 Point Identifiers....4-6
4.4 Programming Model 4-6

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 5-7
5.3.3 Swath Identifiers....5-7
5.4 Programming Model 5-7

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

7. Examples of HDF-EOS Library Usage

7.1 Point Examples 7-1

7.1.1 A C Example of Creating a Simple Point 7-1

7.1.2 A FORTRAN Example of Creating a Simple Point 7-9

7.2 Swath Examples 7-14

7.2.1 Creating a Simple Swath 7-14

7.2.2 A 'Real World' Example of a Swath Creation 7-29

7.3 Grid Examples 7-32

7.3.1 Creating a Simple Grid 7-32

7.4 Combining HDF and HDF-EOS Objects 7-46

7.4.1 Adding HDF-EOS Structures to an Existing HDF File 7-46

7.4.2 Adding an SDS to an Existing HDF File.....7-46

7.4.3 Writing HDF-EOS Structures and Native HDF Objects Simultaneously.....7-46

8. Examples of SDP Toolkit Usage

8.1 Opening a File 8-1

8.1.1 A C Example of a File Open Using the SDP Toolkit.....8-1

8.1.2 A FORTRAN Example of a File Open Using the SDP Toolkit.....8-2

8.2 Status Message Example 8-4

8.2.1 A C Example of SDP Toolkit Status Message Handling 8-6

8.2.2 A FORTRAN Example of SDP Toolkit Status Message Handling.....8-7

8.3 Writing ODL Metadata into HDF-EOS 8-8

8.3.1. A C Example of Metadata Write.....8-8

8.3.2 A FORTRAN Example of Metadata Write 8-44

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 4-3

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

4-1 Summary of the Point Interface ……4-5
5-1 Summary of the Swath Interface ……5-6
6-1 Summary of the Grid Interface ……6-5
6-2 Projection Transformation Package Projection Parameters ……6-10
6-3 Projection Transformation Package Projection Parameters Elements……6-11

Appendix A. Installation and Maintenance

Abbreviations and Acronyms

1. Introduction

1.1 Identification

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

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 NCSA 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

January 1996, Prototype Library Available

January 1996 Version 1 API Available

March 1996, Version 1 API Frozen.

April 1996 - Delivery of HDF-EOS Users Guide and Beta Software

June 1996 - Delivery of Version 1 HDF-EOS, Release A EOSView Available , Beta 1.9

November 1996 - Delivery of Version 1.5 of the HDF-EOS Library. Release A of EOSView, Beta 2.1 Available, Delivery of SDP Toolkit Version 5.1.1

May 1997 - Delivery of Version 2.0 of the HDF-EOS Library. Release B.0 of EOSView, Beta 2.3 Available, Delivery of SDP Toolkit Version 5.2

October 1997 - Delivery of Version 2.1 of the HDF-EOS Library. Release B.0 of EOSView, Delivery of SDP Toolkit Version 5.2.1

March 1998 - Delivery of Version 2.2 of the HDF-EOS Library. Release B.0 of EOSView, Delivery of SDP Toolkit Version 5.2.2

October 1998 - Delivery of Version 2.3 of the HDF-EOS Library. Release B.0 of EOSView, Delivery of SDP Toolkit Version 5.2.3

January 1999 - Delivery of Version 2.4 of the HDF-EOS Library. Release B.0 of EOSView, Delivery of SDP Toolkit Version 5.2.4

June 1999 - Delivery of Version 2.5 of the HDF-EOS Library. Release B.0 of EOSView, Delivery of SDP Toolkit Version 5.2.5

December 1999 - Delivery of Version 2.6 of the HDF-EOS Library. Release B.0 of EOSView, Delivery of SDP Toolkit Version 5.2.6

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 4 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.
- Appendix A Installation Instructions, Test Drivers, User Feedback
- Acronyms

The accompanying Function Reference Guide is organized as follows:

- Section 1 Introduction
- Section2 Function Reference - Specification of the HDF-EOS Pont, Swath and Grid
- APIs
- Acronyms

This page intentionally left blank.

2. Related Documentation

2.1 Parent Documents

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

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
170-WP-012	The HDF-EOS Point 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.

333-CD-003	Release A SCF Toolkit Users Guide for the ECS Project
163-WP-001	An ECS Data Provider's Guide to Metadata in Release A, A White Paper for the ECS Project
175-WP-001	HDF-EOS Primer for Version 1 EOSDIS, A White Paper for the ECS Project
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 Laborartory, Ann Arbor, Michigan, Feb. 1991.
none	Selection of a Map Grid for Data Analysis and Archiva, 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 National Center for Supercomputing Applications (NCSA) at the University of Illinois 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 NCSA HDF, a cursory knowledge of HDF may help the reader to understand the basic workings of HDF-EOS.

HDF 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 HDF can be found in the NCSA documentation referenced in Section 2.2 of this Guide). Many of the NCSA defined datatypes map well to EOS datatypes. Examples include raster images, multi-dimensional arrays, and text blocks. There are other EOS datatypes, however, that do not map directly to NCSA datatypes, particularly in the case of geolocated datatypes. Examples include projected grids, satellite swaths, and field campaign or point data. Therefore, some additions to traditional HDF are required to fully support these datatypes.

To bridge the gap between the needs of EOS data products and the capabilities of HDF, three new EOS specific datatypes – *point*, *swath*, and *grid* – 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.

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 HDF library and, as such, it follows the general design philosophy

and coding style of HDF. For more information on the design of HDF, please refer to the appropriate NCSA 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. NCSA HDF is a subroutine library freely available as source code from the National Center for Supercomputing Applications. 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 NCSA HDF for successful compiling and linking and will be widely available (at no charge) to all interested parties. Although at the time of this writing, the exact packaging has yet to be determined, the basic HDF library will also be available from the ECS project.

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. While the SDP Toolkit may be available to individual researchers, it is unlikely to be of great use outside of EOS DAACs and Science Computing Facilities (SCF). 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 ECS Project, Mar, 1998, 333-CD-100-001, 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.

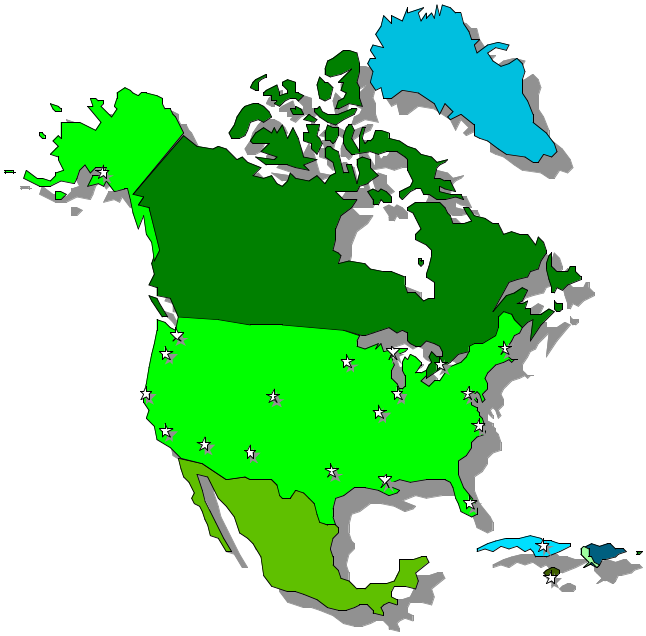
NCSA HDF libraries, on which HDF-EOS is based, is installed automatically with the SDP Toolkit installation script. Please refer to The SDP Toolkit Users Guide for the ECS Project, Section 5 for information pertaining installation and maintenance of the SDP Toolkit.

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.

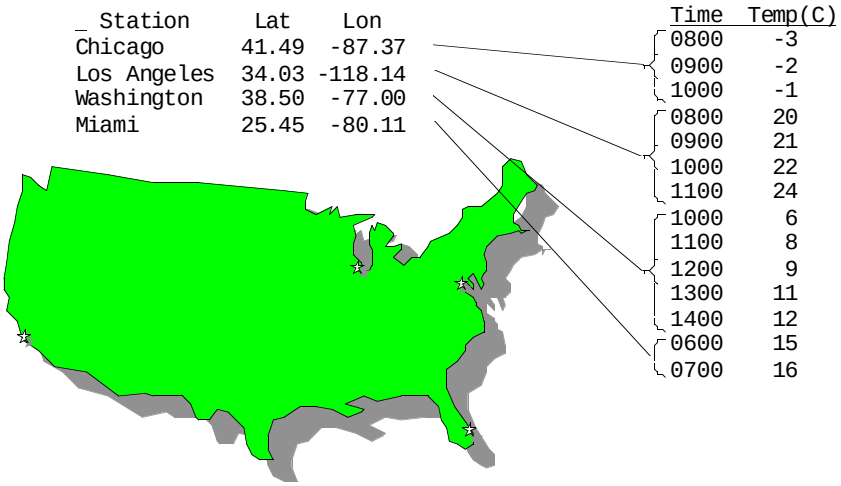


Figure 4-2. Recording Points Over Time

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.

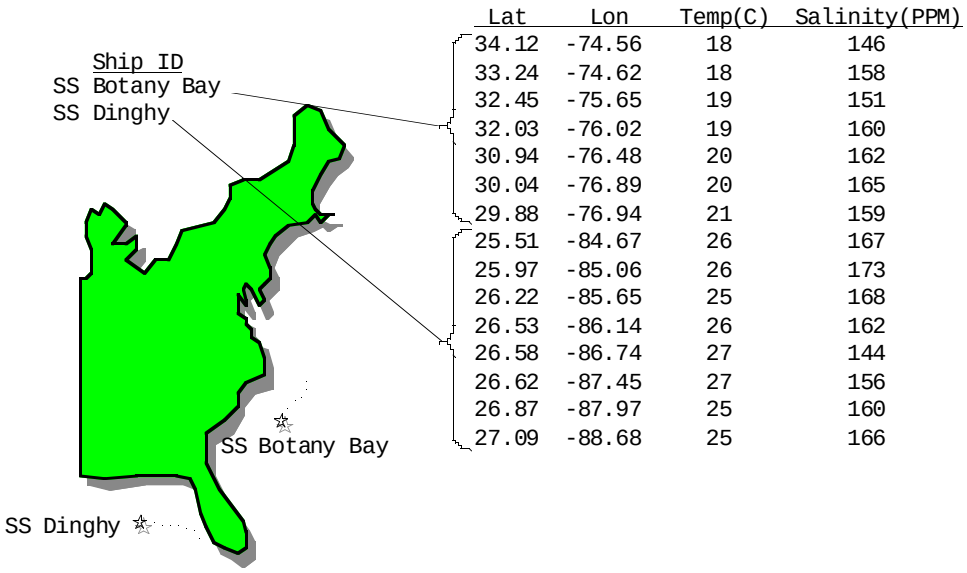


Figure 4-3. Point Data from a Moving Platform

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 PT 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 “PT” and the equivalent FORTRAN routine names are prefixed by “pt.” The PT routines are classified into the following categories:

- *Access routines* initialize and terminate access to the 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 PT function calls are listed in Table 4-1 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 4-1. Summary of the Point Interface

Category	Routine Name C	Routine Name FORTRAN	Description	Page Nos.
Access	PTopen	ptopen	creates a new file or opens an existing one	2-30
	Ptcreate	ptcreate	creates a new point data set and returns a handle	2-6
	PTattach	ptattach	attaches to an existing point data set	2-2
	PTdetach	ptdetach	releases a point data set and frees memory	2-15
	PTclose	ptclose	closes the HDF-EOS file and deactivates the point interface	2-5
Definition	PTdeflevel	ptdeflev	defines a level within the point data set	2-8
	PTdeflinkage	ptdeflink	defines link field to use between two levels	2-10
	Ptdefvrtregion	ptdefvrtreg	defines a vertical subset region	2-13
Basic I/O	PTwritelevel	ptwrlev	writes (appends) full records to a level	2-43
	PTreadlevel	ptrdlev	reads data from the specified fields and records of a level	2-34
	PTupdatelevel	ptuplev	updates the specified fields and records of a level	2-39
	PTwriteattr	ptwrattr	creates or updates an attribute of the point data set	2-41
	PTreadattr	ptrdattr	reads existing attribute of point data set	2-33
Inquiry	PTnlevels	ptnlevs	returns the number of levels in a point data set	2-28
	PTnrecs	ptnrecs	returns the number of records in a level	2-29
	PTnfields	ptnflds	returns number of fields defined in a level	2-27
	PTlevelinfo	ptnlevinfo	returns information about a given level	2-25
	PTlevelindx	ptlevidx	returns index number for a named level	2-24
	PTbcklinkinfo	ptblinkinfo	returns link field to previous level	2-4
	PTfwdlinkinfo	ptflinkinfo	returns link field to following level	2-18
	PTgetlevelname	ptgetlevname	returns level name given level number	2-19
	PTsizeof	ptsizeof	returns size in bytes for specified fields in a point	2-38
	PTattrinfo	ptattrinfo	returns information about point attributes	2-3
	PTinqattrs	ptinqattrs	retrieves number and names of attributes defined	2-22
	PTinqpoint	ptinqpoint	retrieves number and names of points in file	2-23
Utility	PTgetrecnums	ptgetrecnums	returns corresponding record numbers in a related level	2-20
Subset	PTdefboxregion	ptdefboxreg	define region of interest by latitude/longitude	2-7
	PTregioninfo	ptreginfo	returns information about defined region	2-36
	PTregionrecs	ptregrecs	returns # of records and record #s within region	2-37
	PTextractregion	ptextreg	read a region of interest from a set of fields in a single level	2-17
	PTdeftimeperiod	ptdeftmeper	define time period of interest	2-11
	PTperiodinfo	ptperinfo	returns information about defined time period	2-31
	PTperiodrecs	ptperrecs	returns # of records and record #s within time period	2-32
	PTextractperiod	ptextper	read a time period from a set of fields in a single level	2-16

4.3.2 File Identifiers

As with all HDF-EOS interfaces, file identifiers in the PT interface are 32-bit values, 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 data set through the PT interface is as follows:

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

To access a single point data set that already exists in an HDF-EOS file, the calling program must contain the following sequence of C calls:

```
file_id = PTopen(filename, access_mode);
pt_id = PTattach(file_id, point_name);
<Optional operations>
status = PTdetach(pt_id);
status = PTclose(file_id);
```

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 data set, a calling program must obtain a separate point id for each data set. For example, to open two data sets stored in two files, a program would execute the following series of C function calls:

```
file_id_1 = PTopen(filename_1, access_mode);
pt_id_1 = PTattach(file_id_1, point_name_1);
file_id_2 = PTopen(filename_2, access_mode);
pt_id_2 = PTattach(file_id_2, point_name_2);
<Optional operations>
status = PTdetach(pt_id_1);
status = PTclose(file_id_1);
status = PTdetach(pt_id_2);
status = PTclose(file_id_2);
```

Because each file and point data set is assigned its own identifier, the order in which files and data sets 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.

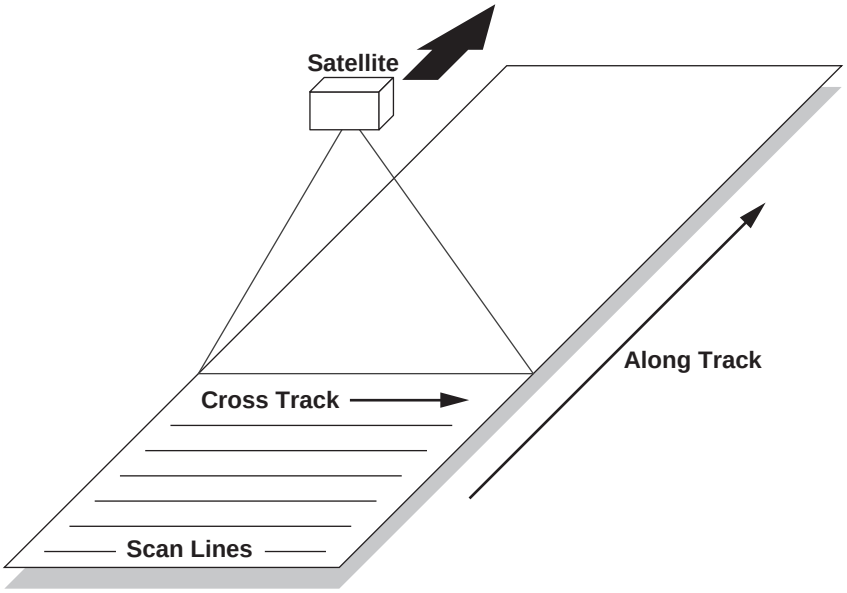


Figure 5-1. A Typical Satellite Swath: Scanning Instrument

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.

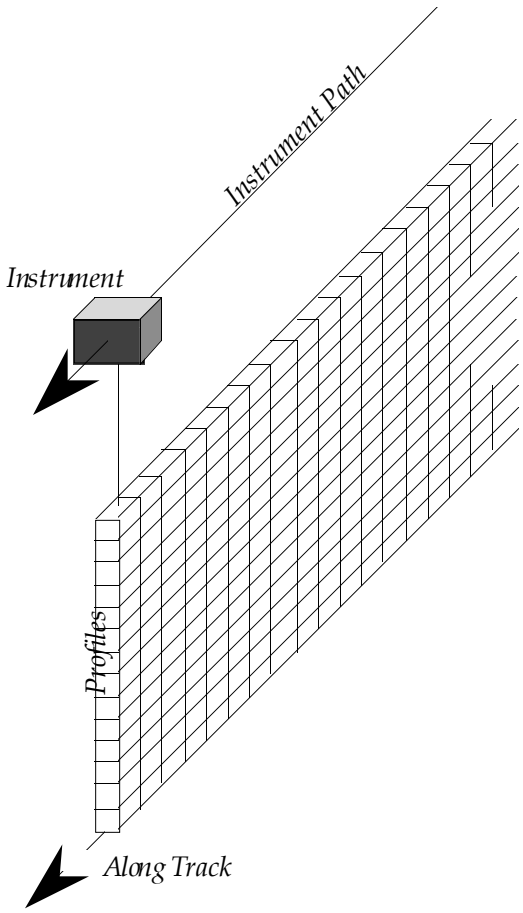


Figure 5-2. A Swath Derived from a Profiling Instrument

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.

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."

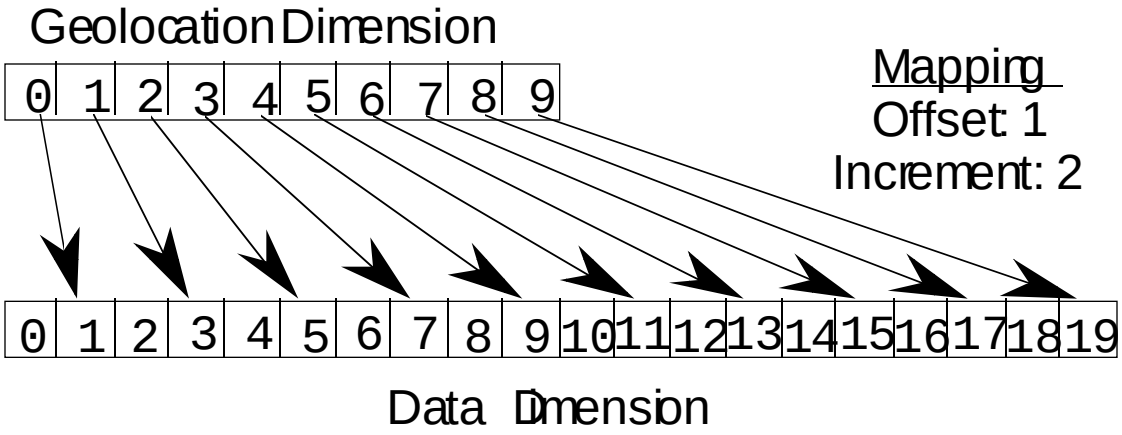


Figure 5-3. A "Normal" Dimension Map

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.

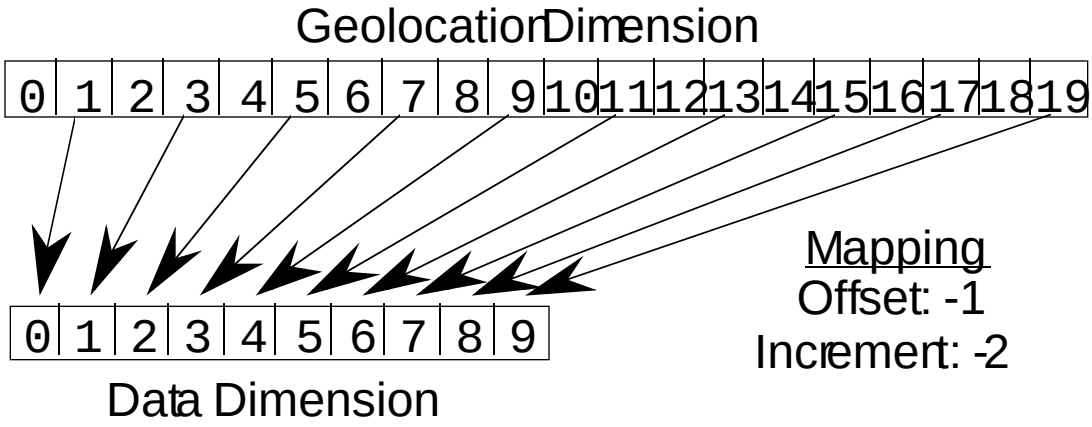


Figure 5-4. A "Backwards" Dimension Map

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 “SW” and the equivalent FORTRAN routine names are prefixed by “sw.” The SW routines are classified into the following categories:

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

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 2)

Category	Routine Name C	Routine Name FORTRAN	Description	Page Nos.
Access	SWopen	swopen	opens or creates HDF file in order to create, read, or write a swath	2-87
	SWcreate	swcreate	creates a swath within the file	2-48
	SWattach	swattach	attaches to an existing swath within the file	2-44
	SWdetach	swdetach	detaches from swath interface	2-66
	SWclose	swclose	closes file	2-46
Definition	SWdefdim	swdefdim	defines a new dimension within the swath	2-55
	SWdefdimmap	swdefmap	defines the mapping between the geolocation and data dimensions	2-56
	SWdefidxmap	swdefimap	defines a non-regular mapping between the geolocation and data dimension	2-60
	SWdefgeofield	swdefgfld	defines a new geolocation field within the swath	2-58
	SWdefdatafield	swdefdfld	defines a new data field within the swath	2-53
	SWdefcomp	swdefcomp	defines a field compression scheme	2-51
	SWwritegeometa	swwrgmeta	writes field metadata for an existing swath geolocation field	2-106
	SWwritedatameta	swwrdmeta	writes field metadata for an existing swath data field	2-102
Basic I/O	SWwritefield	swwrfld	writes data to a swath field	2-103
	SWreadfield	swrdfld	reads data from a swath field.	2-91
	SWwriteattr	swwrattr	writes/updates attribute in a swath	2-100
	SWreadattr	swrdattr	reads attribute from a swath	2-90
	SWsetfillvalue	swsetfill	sets fill value for the specified field	2-97
	SWgetfillvalue	swgetfill	retrieves fill value for the specified field	2-75
Inquiry	SWinqdims	swinqdims	retrieves information about dimensions defined in swath	2-80
	SWinqmaps	swinqmaps	retrieves information about the geolocation relations defined	2-83
	SWinqidxmaps	swinqimaps	retrieves information about the indexed geolocation/data mappings defined	2-82
	SWinqgeofields	swinqgflds	retrieves information about the geolocation fields defined	2-81
	SWinqdatafields	swinqdflds	retrieves information about the data fields defined	2-79
	SWinqattrs	swinqattrs	retrieves number and names of attributes defined	2-78
	SWnentries	swnentries	returns number of entries and descriptive string buffer size for a specified entity	2-86
	SWdiminfo	swdiminfo	retrieve size of specified dimension	2-67
	SWmapinfo	swmapinfo	retrieve offset and increment of specified geolocation mapping	2-85
	SWidxmapinfo	swimapinfo	retrieve offset and increment of specified geolocation mapping	2-77
	SWattrinfo	swattrinfo	returns information about swath attributes	2-45
	SWfieldinfo	swfldinfo	retrieve information about a specific geolocation or data field	2-73
	SWcompinfo	swcompinfo	retrieve compression information about a field	2-47
	SWinqswath	swinqswath	retrieves number and names of swaths in file	2-84
	Swregionindex	swregidx	Returns information about the swath region ID	2-93
	SWupdateidxmap	swupimap	update map index for a specified region	2-98

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

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Subset	Swgeomapinfo	swgmapinfo	retrieves type of dimension mapping when first dimension is geodim	2-76
	SWdefboxregion	swdefboxreg	define region of interest by latitude/longitude	2-49
	SWregioninfo	swreginfo	returns information about defined region	2-95
	SWextractregion	swextreg	read a region of interest from a field	2-71
	SWdeftimeperiod	swdeftmeper	define a time period of interest	2-61
	SWperiodinfo	swperinfo	retuns information about a defined time period	2-88
	SWextractperiod	swextper	extract a defined time period	2-70
	SWdefvrtregion	swdefvrtreg	define a region of interest by vertical field	2-63
	SWdupregion	swdupreg	duplicate a region or time period	2-68

5.3.2 File Identifiers

As with all HDF-EOS interfaces, file identifiers in the SW interface are 32-bit values, 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 SW interface is as follows:

1. Open the file and initialize the SW interface by obtaining a file id from a file name.
2. Open OR create a swath data set 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.

To access a single swath data set that already exists in an HDF-EOS file, the calling program must contain the following sequence of C calls:

```
file_id = SWopen(filename, access_mode);
sw_id = SWattach(file_id, swath_name);
<Optional operations>
status = SWdetach(sw_id);
status = SWclose(file_id);
```

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 swath data set, a calling program must obtain a separate swath id for each data set. For example, to open two data sets stored in two files, a program would execute the following series of C function calls:

```
file_id_1 = SWopen(filename_1, access_mode);
sw_id_1 = SWattach(file_id_1, swath_name_1);
file_id_2 = SWopen(filename_2, access_mode);
sw_id_2 = SWattach(file_id_2, swath_name_2);
<Optional operations>
status = SWdetach(sw_id_1);
status = SWclose(file_id_1);
status = SWdetach(sw_id_2);
status = SWclose(file_id_2);
```

Because each file and swath data set is assigned its own identifier, the order in which files and data sets 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. 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 5, 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.

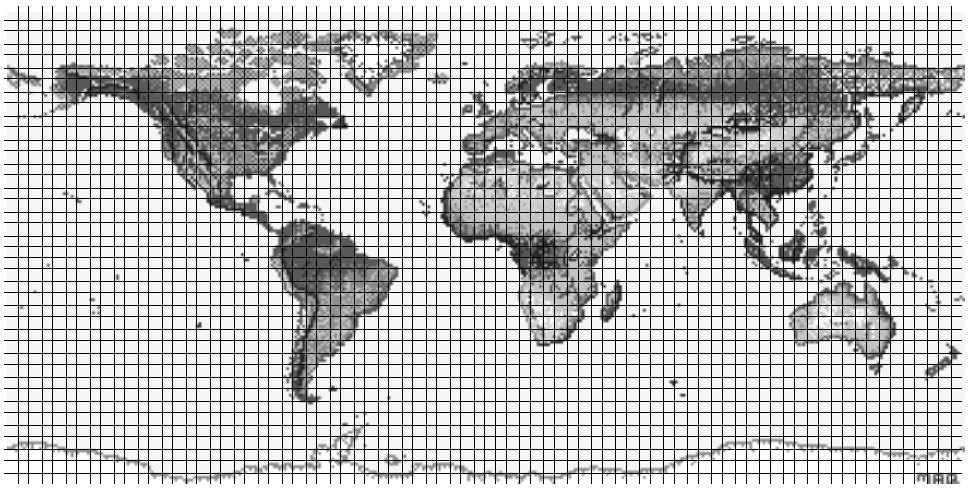


Figure 6-1. A Data Field in a Mercator-projected Grid

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.

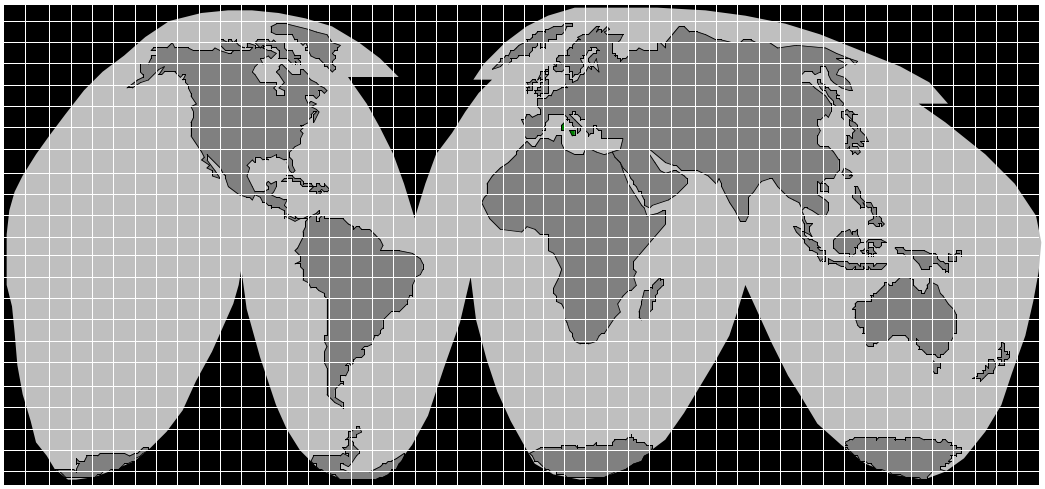


Figure 6-2. A Data Field in an Interrupted Goode's Homolosine-Projected Grid

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 for ECS Release B.

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 "GD" and the equivalent FORTRAN routine names are prefixed by "gd." The GD routines are classified into the following categories:

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

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 C	Routine Name FORTRAN	Description	Page Nos.
Access	GDopen	gdopen	creates a new file or opens an existing one	2-153
	GDcreate	gdcreate	creates a new grid in the file	2-114
	GDattach	gdattach	attaches to a grid	2-108
	GDdetach	gddetach	detaches from grid interface	2-134
	GDclose	gdclose	closes file	2-112
Definition	Gddeforigin	gddeforigin	defines origin of grid	2-123
	Gddefdim	gddefdim	defines dimensions for a grid	2-120
	GDdefproj	gddefproj	defines projection of grid	2-125
	GDdefpixreg	gddefpixreg	defines pixel registration within grid cell	2-124
	GDdeffield	gddeffld	defines data fields to be stored in a grid	2-121
	GDdefcomp	gddefcomp	defines a field compression scheme	2-118
	GDblkSOMoffset	none	This is a special function for SOM MISR data. Write block SOM offset values.	2-110
	GDsettilecomp	none	This routine was added as a fix to a bug in HDF-EOS. The current method of implementation didn't allow the user to have a field with fill values and use tiling and compression. This function allows the user to access all of these features.	2-165
	GDwritefieldmeta	gdwrmeta	writes metadata for field already existing in file	2-172
Basic I/O	GDwritefield	gdwrfld	writes data to a grid field.	2-170
	GDreadfield	gdrdfld	reads data from a grid field	2-158
	GDwriteattr	gdwrattr	writes/updates attribute in a grid.	2-168
	GDreadattr	gdrdattr	reads attribute from a grid	2-157
	GDsetfillvalue	gdsetfill	sets fill value for the specified field	2-163
	GDgetfillvalue	gdgetfill	retrieves fill value for the specified field	2-140
Inquiry	GDinqdims	gdinqdims	retrieves information about dimensions defined in grid	2-147
	GDinqfields	gdinqdflds	retrieves information about the data fields defined in grid	2-148
	GDinqattrs	gdinqattrs	retrieves number and names of attributes defined	2-146
	GDnentries	gdnentries	returns number of entries and descriptive string buffer size for a specified entity	2-152
	GDgridinfo	gdgridinfo	returns dimensions of grid and X-Y coordinates of corners	2-145
	GDprojinfo	gdprojinfo	returns all GCTP projection information	2-156
	GDdiminfo	gddiminfo	retrieves size of specified dimension.	2-135
	GDcompinfo	gdcompinfo	retrieve compression information about a field	2-113
	GDfieldinfo	gdfldinfo	retrieves information about a specific geolocation or data field in the grid	2-138
	GDinqgrid	gdinqgrid	retrieves number and names of grids in file	2-149
	GDattrinfo	gdattrinfo	returns information about grid attributes	2-109
	GDorigininfo	gdorginfo	return information about grid origin	2-154
	GDpixreginfo	gdpreginfo	return pixel registration information for given grid	2-155
	GDdefboxregion	gddefboxreg	define region of interest by latitude/longitude	2-117
	GDregioninfo	gdreginfo	returns information about a defined region	2-161

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

Category	Routine Name		Description	Page Nos.
	C	FORTRAN		
Subset	GDextractregion	gdextrreg	read a region of interest from a field	2-137
	GDdeftimeperiod	gddeftmeper	define a time period of interest	2-129
	GDdefvrtregion	gddefvrtreg	define a region of interest by vertical field	2-131
	GDgetpixels	gdgetpix	get row/columns for lon/lat pairs	2-141
	GDgetpixvalues	gdgetpixval	get field values for specified pixels	2-143
	GDinterpolate	gdinterpolate	perform bilinear interpolation on a grid field	2-150
	GDdupregion	gddupreg	duplicate a region or time period	2-136
Tiling	GDdeftile	gddeftle	define a tiling scheme	2-127
	GDtileinfo	gdtleinfo	returns information about tiling for a field	2-166
	GDsettilecache	gdsettleche	set tiling cache parameters	2-164
	GDreadtile	gdrdtle	read data from a single tile	2-160
	GDwritetile	gdwrtile	write data to a single tile	2-173

6.3.2 File Identifiers

As with all HDF-EOS interfaces, file identifiers in the GD interface are 32-bit values, 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 data set 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 data set by obtaining a grid id from a grid name.
3. Perform desired operations on the data set.
4. Close the grid data set by disposing of the grid id.
5. Terminate grid access to the file by disposing of the file id.

To access a single grid data set that already exists in an HDF-EOS file, the calling program must contain the following sequence of C calls:

```
file_id = GDopen(filename, access_mode);
gd_id = GDattach(file_id, grid_name);
<Optional operations>
status = GDdetach(gd_id);
status = GDclose(file_id);
```

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 data set, a calling program must obtain a separate grid id for each data set. For example, to open two data sets stored in two files, a program would execute the following series of C function calls:

```
file_id_1 = GDopen(filename_1, access_mode);
gd_id_1 = GDattach(file_id_1, grid_name_1);
file_id_2 = GDopen(filename_2, access_mode);
gd_id_2 = GDattach(file_id_2, grid_name_2);
<Optional operations>
status = GDdetach(gd_id_1);
status = GDclose(file_id_1);
status = GDdetach(gd_id_2);
status = GDclose(file_id_2);
```

Because each file and grid data set is assigned its own identifier, the order in which files and data sets 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_LAMCC        (4)   Lambert Conformal Conic
GCTP_PS           (6)   Polar Stereographic
GCTP_POLYC        (7)   Polyconic
GCTP_TM           (9)   Transverse Mercator
GCTP_LAMAZ        (11)  Lambert Azimuthal Equal Area
GCTP_HOM          (20)  Hotine Oblique Mercator
GCTP_SOM          (22)  Space Oblique Mercator
GCTP_GOOD         (24)  Interrupted Goode Homolosine
GCTP_ISINUS       (99)  Intergerized Sinusoidal Projection*
```

* 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.

Note that other projections supported by GCTP will be adapted for HDF-EOS Version 2.6 as new user requirements are surfaced. For further details on the GCTP projection package, please refer to Section 6.3.4 and Appendix G of the SDP Toolkit Users Guide for the ECS Project, March, 1998, (333-CD-100-001)

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)
Austrailian National (14)
Krassovsky (15)
Hough (16)
Mercury 1960 (17)
Modified Mercury 1968 (18)
Sphereof Radius 6370997m(19)

6.5.4 Projection Parameters

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						
4 Lambert Conformal C	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
6 Polar Stereographic	SMajor	SMinor			LongPol	TrueScale	FE	FN
7 Polyconic	SMajor	SMinor			CentMer	OriginLat	FE	FN
9 Transverse Mercator	SMajor	SMinor	Factor		CentMer	OriginLat	FE	FN
11 Lambert Azimuthal	Sphere				CentLon	CenterLat	FE	FN
20 Hotin Oblique Merc A	SMajor	SMinor	Factor			OriginLat	FE	FN
20 Hotin Oblique Merc B	SMajor	SMinor	Factor	AziAng	AzmthPt	OriginLat	FE	FN
22 Space Oblique Merc A	SMajor	SMinor		IncAng	AscLon g		FE	FN
22 Space Oblique Merc B	SMajor	SMinor	Satnum	Path			FE	FN
24 Interrupted Goode	Sphere							
99 Integerized Sinusoidal	Sphere				CentMer		FE	FN

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					
4 Lambert Conformal C					
6 Polar Stereographic					
7 Polyconic					
9 Transverse Mercator					
11 Lambert Azimuthal					
20 Hotin Oblique Merc A	Long1	Lat1	Long2	Lat2	zero
20 Hotin Oblique Merc B					one
22 Space Oblique Merc A	PSRev	SRat	PFlag		zero
22 Space Oblique Merc B					one
24 Interrupted Goode					
99 Integerized Sinusoidal	NZone		RFlag		

Where,

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

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

Smajor — Semi-major axis of ellipsoid. If zero, Clarke 1866 in meters is assumed.

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

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

STDPR1 — Latitude of the first standard parallel

STDPR2 — Latitude of the second standard parallel

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

This page intentionally left blank.

7. Examples of HDF-EOS Library UsageÄ

This Section contains code examples of usage of the HDF-EOS Library specified in Sections 5, 6 and 7 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 of the use of the Point 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 Point interface.

7.1.1 A C Example of Creating a Simple Point

```
/* In this example we will (1) open an HDF file, and (2) create three point
* interfaces within the file.
*/


#include "hdf.h"
/* SetupPoint */
main()
{
    intn            status;
    int32           ptfid, PTid1, PTid2, PTid3;
/*
* We first open the HDF point file, "PointFile.hdf".  Because this file
* does not already exist, we use the DFACC_CREATE access code in the
*  open statement.  The PTopen routine returns the point file id, ptfid,
* which is used to identify the file in subsequent routines in the
* library.
*/
    ptfid = PTopen("PointFile.hdf", DFACC_CREATE);
    PTid1 = PTcreate(ptfid, "Simple Point");
    PTid2 = PTcreate(ptfid, "FixedBuoy Point");
    PTid3 = PTcreate(ptfid, "FloatBuoy Point");
/*
* We now close the point interface with the PTdetach routine.  This step
* is necessary to properly store the point information within the file.
```

```
*/
PTdetach(PTid1);
    PTdetach(PTid2);
    PTdetach(PTid3);
/*
* Finally, we close the point file using the PTclose routine.  This will
* release the point file handles established by Ptopen.
*/
    PTclose(ptfid);
return;
}


#include "mfhdf.h"
/* DefineLevels */
/*
 * In this example we open the "PointFile" HDF file and define the levels
 * for each point object.
 */
main()
{
intn            status;
int32           ptfid, PTid, fieldtype[8], fieldorder[8];
char            fldlist[255];
/*
* We first open the HDF swath file, "PointFile.hdf".  Because this file
* already exist and we wish to write to it, we use the DFACC_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 = PTopen("PointFile.hdf", DFACC_RDWR);
if (ptfid != -1)
    {
      /* Simple Point */
      PTid = PTattach(ptfid, "Simple Point");
   /* Attach to point object defined above */
      strcpy(fldlist, "Time,Concentration,Species");
      fieldtype[0] = DFNT_FLOAT64;
      fieldtype[1] = DFNT_FLOAT32;
      fieldtype[2] = DFNT_CHAR8;
      fieldorder[0] = 1;
      fieldorder[1] = 4;
      fieldorder[2] = 4;
```

```
   /* The first time Ptdeflevel is called, it defaults to calling it
      *  level 0. The second time it is level 1 and so on.
      */

      status = PTdeflevel(PTid, "Sensor", fldlist, fieldtype, fieldorder);
      PTdetach(PTid);
      goto skip;
      /* Fixed Buoy Point */
      PTid = PTattach(ptfid, "FixedBuoy Point");
      /* Define Description/Location Level */
      strcpy(fldlist, "Label,Longitude,Latitude,DeployDate,ID");
      fieldtype[0] = DFNT_CHAR8;
      fieldtype[1] = DFNT_FLOAT64;
      fieldtype[2] = DFNT_FLOAT64;
      fieldtype[3] = DFNT_INT32;
      fieldtype[4] = DFNT_CHAR8;
      fieldorder[0] = 8;
      fieldorder[1] = 0;  /* Order 0 same as Order 1 for numeric scalars */
      fieldorder[2] = 0;
      fieldorder[3] = 0;
      fieldorder[4] = 1;
   /* Defining level 1 */
      status = PTdeflevel(PTid, "Desc-Loc", fldlist,
                      fieldtype, fieldorder);
      /* Define Data Level */
      strcpy(fldlist, "Time,Rainfall,Temperature,ID");
      fieldtype[0] = DFNT_FLOAT64;
      fieldtype[1] = DFNT_FLOAT32;
      fieldtype[2] = DFNT_FLOAT32;
return;
}



#include "mfhdf.h"
#include <math.h>
/* WriteLevels */
main()
/* After defining levels, we will now write data to the fields. */
{
    intn            status, i, j;
    int16           wgt;
    int32           ptfid, dum, n;
    int32           PTid, date, i32 = 9999;
    float32         rain, temp, conc[4], f32 = -7.5;
```

```
    float64         lon, lat, time;
    char           *pntr, buffer[10000], id[2], desc[16], spc[8];
    FILE           *fp;
    /*
     * Open the HDF swath file, "PointFile.hdf".
     */
    ptfid = PTopen("PointFile.hdf", DFACC_RDWR);
    /* Write to Simple Point */
       PTid = PTattach(ptfid, "Simple Point");
    fp = fopen("simple.txt", "r");
    n = 0;
    pntr = buffer;
    while(fscanf(fp, "%lf %f %f %f %f %s",
             &time, &conc[0], &conc[1], &conc[2], &conc[3], spc) != -1)
    {
      n++;
      spc[strlen(spc)-1] = 0;
      memcpy(pntr, &time, 8);
      pntr += 8;
      memcpy(pntr, conc, 4*4);
      pntr += 4*4;
      memcpy(pntr, spc + 1, 4);
      pntr += 4;
    }
    fclose(fp);
    status = PTwritelevel(PTid, 0, n, buffer);
    PTdetach(PTid);
    /* Write to Fixed Buoy Point */
    PTid = PTattach(ptfid, "FixedBuoy Point");
    /* Write First (0th) Level */
```

```
#include "hdf.h"
main()
{
    intn            i, status;
    int32           ptfid, PTid, regionID, periodID, size;
    int32           level;
    float64         cornerlon[2], cornerlat[2], starttime, stoptime;
    float64        *datbuf;
    /*
     * Open the HDF point file, "PointFile.hdf".
     */
     ptfid = PTopen("PointFile.hdf", DFACC_RDWR);
```

```
  if (ptfid != -1)
 {
   PTid = PTattach(ptfid, "FloatBuoy Point");

/* This section of code demonstrates the use of the subsetting
   *  functions, Ptdefboxregion, PT regioninfo, PTextractregion
   */
   if (PTid != -1)
   {
       cornerlon[0] = -145.;
       cornerlat[0] = -15.;
       cornerlon[1] = -135;
       cornerlat[1] = -8.;
       regionID = PTdefboxregion(PTid, cornerlon, cornerlat);
       level = 1;
       status = PTregioninfo(PTid, regionID, level,
                      "Longitude,Latitude", &size);
       datbuf = (float64 *) malloc(size);
       status = PTextractregion(PTid, regionID, level,
                      "Longitude,Latitude", (char *) datbuf);
       for (i=0; i<size/16; i++)
       {
         printf("%d %lf %lf\n", i, datbuf[2*i], datbuf[2*i+1]);
       }
/* This section of code demonstrates the time subsetting functions,
   *  Ptdeftimeperiod, Ptperiodinfo, Ptextractperiod
   */
       free(datbuf);
       starttime = 35208757.6;
       stoptime = 35984639.2;
       periodID = PTdeftimeperiod(PTid, starttime, stoptime);
       level = 1;
       status = PTperiodinfo(PTid, periodID, level,
                      "Time", &size);
       datbuf = (float64 *) malloc(size);
       status = PTextractperiod(PTid, periodID, level,
                      "Time", datbuf);
       for (i=0; i<size/8; i++)
       {
         printf("%d %lf\n", i, datbuf[i]);
       }
       free(datbuf);
   }
 }
 PTdetach(PTid);
```

```
    PTclose(ptfid);
    return;
}



#include "mfhdf.h"
#include <math.h>
/* ReadLevels */
main()
{
    intn            status, i, j;
    int32           ptfid, n, nt, sz;
    int32           PTid, date, i32;
    float32         rain, temp, conc[4];
    float64         lon, lat, time;
    int32           fldsz, fldlevels[32], nattr;
    int32           fldtype[32], fldorder[32], recs[128];
    char           *pntr, *buffer, id[2], desc[16], spc[8];
    char            fldlist[128], attrnames[16];
    /*

     * Open the HDF swath file, "PointFile.hdf".
     */
    ptfid = PTopen("PointFile.hdf", DFACC_READ);
/* Read Simple Point */

PTid = PTattach(ptfid, "Simple Point");
    status = PTlevelinfo(PTid, 0, fldlist, fldtype, fldorder);
    fldsz = PTsizeof(PTid, fldlist, fldlevels);
    n = PTnrecs(PTid, 0);
    buffer = (char *) calloc(n * fldsz, 1);
    for (i=0; i<n; i++)
    {
      recs[i] = i;
    }
    status = PTreadlevel(PTid, 0, fldlist, n, recs, buffer);
    pntr = buffer;
    for (i=0; i<n; i++)
    {
      memcpy(&time, pntr, 8);
      pntr += 8;
      memcpy(conc, pntr, 4*4);
      pntr += 4*4;
      memcpy(spc, pntr, 4);
```

```
      pntr += 4;
      printf("%12.1f %6.2f %6.2f %6.2f %6.2f %s\n",
             time,conc[0],conc[1],conc[2],conc[3],spc);
    }
    free(buffer);
    PTdetach(PTid);



#include "mfhdf.h"
/*   UpdateLevels */
main()
{
    intn            status, i, j;
    int32           ptfid;
    int32           PTid1, PTid2;
    int32           nrec, recs[32];
    int32           outNrec, outRecs[32];
    int32           inLevel, outLevel;
    char            datbuf[1024];
    float64         f64;
    /*
     * Open the HDF point file, "PointFile.hdf".
     */
    ptfid = PTopen("PointFile.hdf", DFACC_RDWR);
    if (ptfid != -1)
    {
{
      PTid1 = PTattach(ptfid, "Simple Point");
      PTid2 = PTattach(ptfid, "FixedBuoy Point");
      if (PTid1 != -1 && PTid2 != -1)
      {
   /* In this section of code, we show how to modify data */
          f64 = 43.2;
          memcpy(datbuf, &f64, 8);
          recs[0] = 1;
          status = PTupdatelevel(PTid2, 0, "Longitude", 1, recs, datbuf);
          datbuf[0] = 'F';
          nrec = 1;
          recs[0] = 0;
          status = PTupdatelevel(PTid2, 0, "ID", nrec, recs, datbuf);
          inLevel = 0;
          outLevel = 1;
          status = PTgetrecnums(PTid2, inLevel, outLevel, nrec, recs,
                         &outNrec, outRecs);
```

```
          for (i=0; i<outNrec; i++)
          {
            datbuf[i] = 'F';
          }
          status = PTupdatelevel(PTid2, outLevel, "ID",
                           outNrec, outRecs, datbuf);
      }
    }
    PTdetach(PTid1);
    PTdetach(PTid2);
    PTclose(ptfid);
    HEprint(stdout,0);
    return;
}
```

7.1.2 A FORTRAN Example of Creating a Simple Point

```
c
c  In this example we will (1) open an HDF file, and (2) create three point
c  interfaces within the file.
c
program setuppoint
      integer*4          ptfid, ptid1, ptid2, ptid3, ptdetach, ptclose
      integer*4          ptopen, ptcreate
      integer DFACC_CREATE
      parameter (DFACC_CREATE=4)
c
c      We first open the HDF point, "PointFile.hdf".  Because this
c      file does not already exist, we use the DFACC_CREATE access
c      code in the open statement.  The ehopen routine returns the point
c      file id, ptfid, which is used to identify the file in subsequent
c      routines in the library.
c
      ptfid = ptopen("PointFile.hdf", DFACC_CREATE)
      ptid1 = ptcreate(ptfid, "Simple Point")
c
c     We now close the point interface with the ptdetach routine.  This
c        step is necessary to properly store the point information within the
c     file.
c
      status = ptdetach(ptid1)
c
c     Finally, we close the point file using the ehclose routine.  This
c     will release the point file handles established by ptopen.
c
      status = ptclose(ptfid)
      stop
      end
c
C
C
C
c     In this example we will (1) open the "PointFile" HDF file,
c     (2) attach to the point structure, and (3) define
c     the fields within the point.
c
      program definelevels
      integer         status
      integer*4       ptfid, ptid
      integer*4       fieldtype(8), fieldorder(8)
```

```
      character*255   fldlist
      integer*4       ptopen, ptattach
      integer         ptdeflev, ptdeflink, ptdetach, ptclose
      integer  DFACC_RDWR
      parameter (DFACC_RDWR=3)
      integer  DFNT_CHAR8, DFNT_INT16, DFNT_INT32
      parameter (DFNT_CHAR8=4)
      parameter (DFNT_INT16=22)
      parameter (DFNT_INT32=24)
      integer  DFNT_FLOAT32, DFNT_FLOAT64
      parameter (DFNT_FLOAT32=5)
      parameter (DFNT_FLOAT64=6)
c
c     We first open the HDF point file, "PointFile.hdf".  Because this
c     file already exist and we wish to write to it, we use the
c     DFACC_RDWR access code in the open statement.  The ptopen
c     routine returns the point fileid, ptfid, which is used to
c     identify the file in subsequent routines.
      ptfid = ptopen("PointFile.hdf", DFACC_RDWR)
c
c     We next attach to the three point structures within the file
c  using the ptattach routine, identifying each point by its
c  name defined by the ptcreate routine previously.
c
      if (ptfid .ne. -1) then
c      Define "Simple" point
         ptid = ptattach(ptfid, "Simple Point")
         fldlist = "Time,Concentration,Species"
         fieldtype(1) = DFNT_FLOAT64
         fieldtype(2) = DFNT_FLOAT32
         fieldtype(3) = DFNT_CHAR8
         fieldorder(1) = 1
         fieldorder(2) = 4
         fieldorder(3) = 4
         status = ptdeflev(ptid, "Sensor", fldlist, fieldtype,
     1                     fieldorder)
         status = ptdetach(ptid)
c
c      Close HDF file
         status = ptclose(ptfid)
         write(*,*) 'status close ', status
      endif
      stop
      end
c
```

```
c
c
c  In this example we will (1) open the "PointFile" HDF file, (2) attach to
c  the points, and (3) write data to each level of the points.
c
      program writelevels
      implicit none
      integer            status, i, pntr
      integer*2          wgt
      integer*4          ptfid, ptid, n, date
      real*4             rain, temp, conc(4)
      real*8             lon, lat, time
      character          buffer*10000, id*2, desc*16, spc*8
      character          ctime*8, cconc*16, clon*8, clat*8
      character          cdate*4, crain*4, ctemp*4, cwgt*2
      equivalence        (time,ctime), (conc,cconc), (lon,clon)
      equivalence        (lat,clat), (rain,crain), (temp,ctemp)
      equivalence        (date,cdate), (wgt,cwgt)
      integer            ptwrlev, ptdetach, ptclose
      integer*4          ptopen, ptattach
      integer DFACC_RDWR
      parameter (DFACC_RDWR=3)
c     Open the HDF point file, "PointFile.hdf".
c
      ptfid = ptopen("PointFile.hdf", DFACC_RDWR)
c    Write to Simple Point
c
      ptid = ptattach(ptfid, "Simple Point")
      open(unit=1, file='simple.txt', status='OLD')
      n = 0
      pntr = 1
      do 10 i=1,1000
         read(1, *, end=100)  time, conc(1), conc(2), conc(3),
     1                            conc(4), spc
         n = n + 1
         buffer(pntr:pntr+8) = ctime
         pntr = pntr + 8
         buffer(pntr:pntr+4*4) = cconc
         pntr = pntr + 4*4
         buffer(pntr:pntr+4) = spc
         pntr = pntr + 4
10    continue
100   close(unit=1)
      status = ptwrlev(ptid, 0, n, buffer)
      status = ptdetach(ptid)
```

```
               status = plclose(ptfid)
c
c
c
c  In this example we will (1) open the "PointFile" HDF file, (2) attach to
c  the points, and (3) read data from each level of the points.
c
      program readlevels
      integer*4         ptfid, ptid, recs(32)
      integer*4         fldtype(32), fldorder(32)
      integer            status, i, pntr
      integer*2          wgt
      integer*4          n, date
      real*4             rain, temp, conc(4)
      real*8             lon, lat, time
      character         fldlist*256, buffer*100000, id*2
      character          spc*4, desc*8
      character          ctime*8, cconc*16, clon*8, clat*8
      character          cdate*4, crain*4, ctemp*4, cwgt*2
      equivalence        (time,ctime), (conc,cconc), (lon,clon)
      equivalence        (lat,clat), (rain,crain), (temp,ctemp)
      equivalence        (date,cdate), (wgt,cwgt)
      integer           ptrdlev
      integer           ptdetach, ptclose, ptlevinfo
      integer*4         ptopen, ptattach, ptnrecs
      integer DFACC_READ
      parameter (DFACC_READ=1)
c     Open the HDF point file, "PointFile.hdf".
c
      ptfid = ptopen("PointFile.hdf", DFACC_READ)
c    Read Simple Point
c
      ptid = ptattach(ptfid, "Simple Point")
      status = ptlevinfo(ptid, 0, fldlist, fldtype, fldorder)
      n = ptnrecs(ptid, 0)
      do 5 i=1,n
         recs(i) = i - 1
 5    continue
      status = ptrdlev(ptid, 0, fldlist, n, recs, buffer)
      pntr = 1
      do 10 i=1,n
         ctime = buffer(pntr:pntr+8)
         pntr = pntr + 8
         cconc = buffer(pntr:pntr+4*4)
         pntr = pntr + 4*4
```

```
         spc = buffer(pntr:pntr+4)
         pntr = pntr + 4
         write(*,*) time, conc(1), conc(2), conc(3), conc(4), spc
10    continue
      status = ptdetach(ptid)
      status = ptclose(ptfid)
      stop
      end


program updatelevels
      integer            ptuplev, ptdetach, ptclose, ptgetrecnums
      integer*4          ptfid, ptid1, ptid2
      integer*4          recs(32), inlevel, outlevel
      integer*4          ptopen, ptattach
      integer*4          outrecs(32), outnrec
      real*8             f64
      character          datbuf*256, c8*8
      equivalence        (f64,c8)
      integer DFACC_RDWR
      parameter (DFACC_RDWR=3)
c     Open the HDF point file, "PointFile.hdf".
c
      ptfid = ptopen("PointFile.hdf", DFACC_RDWR)
      ptid2 = ptattach(ptfid, "FixedBuoy Point")
      f64 = 43.2
      datbuf(1:8) = c8
      recs(1) = 1
      status = ptuplev(ptid2, 0, "Longitude", 1, recs, datbuf)
      datbuf(1:1) = 'F'
      recs(1) = 0
      status = ptuplev(ptid2, 0, "ID", 1, recs, datbuf)
      inlevel = 0
      outlevel = 1
      nrec = 1
      status = ptgetrecnums(ptid2, inlevel, outlevel, nrec, recs,
     1                      outnrec, outrecs)
      do 10 i=1,outnrec
            datbuf(i:i) = 'F'
 10   continue
      status = ptuplev(ptid2, outlevel, "ID", outnrec, outrecs, datbuf)
      status = ptdetach(ptid2)
      status = ptclose(ptfid)
      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. The section concludes with a real world example from the ECS “V0 Data Migration” effort, written in C.

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

The following C source code is part of a program written for the "ECS V0 Data Migration"effort. It creates, defines and writes a Swathdata set in HDF-EOS format using the Swath interface functions. The program converts ERBE radiometric data from native format to HDF-EOS.Only the portions of the program that deals with the Swath HDF-EOS interface has been reproduced here.

Note: This is a test example; it is presented for the purpose of providing a guide for users of HDF-EOS. The example has not yet been validated by the LaRC DAAC or users of ERBE data.

```
/* Driver for the set up of the swath HDF-EOS file
   and the transfer of the data to that file */
/*
   William E. Smith, Applied Research Corporation
   Peter Chiang, Lockheed-Martin
   September 9, 1996
*/
/* Include Ingest DMS Preprocessing common external science header */
#include "InPpCommonExtSciData.h"
/* Include PGS Toolkit headers */
#include "PGS_MET.h"
#include "PGS_PC.h"
#include "hdf.h"
/* External function declarations */
int setup( char* filename );
int write_data( FILE* logout, char* hdffile, char* infile );
int metawrite( FILE* logout,  char* hdf_filename, char* dumpfilename );
int sids8_( char* infilename, char* s8name, char* convertfilename,
             char* dumpfilename, char* logfilename, char* patrcfm );
/* Main convert function DMS Preprocessing calls */
#ifdef InDPpSharedObject
int Convert()
```

```
#else
int main( void )
#endif
{
    /* Conversion status declaration */
    int   convertStatus;
    /* Log file pointer */
    FILE* logout;
    /* Temporary file declarations */
    char  datFilename  [ PGSd_PC_FILE_PATH_MAX ];
    char  dumpFilename [ PGSd_PC_FILE_PATH_MAX ];
    /* Filename arrays */
    char  logFilename  [ PGSd_PC_FILE_PATH_MAX ];
    char  inputFilename[ PGSd_PC_FILE_PATH_MAX ];
    char  patrcfm [ PGSd_PC_FILE_PATH_MAX ];
    char  outputHDFFilename     [ PGSd_PC_FILE_PATH_MAX ];
    char* s8Name;
    /* PGS Toolkit -related variables */
    PGSt_integer    version;
    PGSt_SMF_status returnStatus;
    PGSt_SMF_code   code;
    char            mnemonic [PGS_SMF_MAX_MNEMONIC_SIZE];
    char            message  [PGS_SMF_MAX_MSGBUF_SIZE];
    /* Start off with the convert status at 0 (success) */
    convertStatus = 0;
    /* Associate logical reference to physical filename */
    version = 1;
    returnStatus = PGS_PC_GetReference( InDPpPCFLogIDReturnStatus, &version,
logFilename);
    if ( returnStatus != PGS_S_SUCCESS )
    {
        /* Get and report an error */
        PGS_SMF_GetMsg( &code, mnemonic, message );
        printf("Error, mig_driver(), \"Log File Status = %s\"\n", message );
        return ( 1 );
    }
    /* If filename missing or there is a problem, print message and exit */
    if ( ( logout = fopen( logFilename, "w" ) ) == NULL)
    {
        printf("Error, mig_driver(), \"Cannot open %s.\"\n",logFilename);
        return ( 1 );
    }
    /* Indicate that we are starting migration */
    fprintf(logout,"Informational, mig_driver(), \"Start migration.\"\n");
    /* Associate logical reference to the input filename */
```

```
    version =1;
    returnStatus = PGS_PC_GetReference( InDPpPCFLogIDStartInput, &version,
inputFilename);
    if ( returnStatus != PGS_S_SUCCESS )
    {
/* Get and report an error */
        PGS_SMF_GetMsg( &code, mnemonic, message );
        fprintf(logout, "Error, mig_driver(), \"Input File Status = %s\"\n",
message );
        convertStatus = 1;
    }
    /* Associate logical reference to the output filename */
    version =1;
    returnStatus = PGS_PC_GetReference( InDPpPCFLogIDStartOutput, &version,
outputHDFFilename);
    if ( returnStatus != PGS_S_SUCCESS )
    {
        /* Get and report an error */
        PGS_SMF_GetMsg( &code, mnemonic, message );
        fprintf(logout, "Error, mig_driver(), \"Output File Status = %s\"\n",
message );
        convertStatus = 1;
    }
    if ( getenv( "INGEST_CONVERSION_ALGORITHM_PATH" ) )
    {
        /* use the conversion algorithm path environment variable */
        strcpy( patrcfm, getenv( "INGEST_CONVERSION_ALGORITHM_PATH" ) );
    }
    else
    {
        /* Default to the current path */
        strcpy( patrcfm, "." );
    }
    /* construct the rest of the patrcfm file */
    strcat( patrcfm, "/patrcfm" );
    /* Close the log file while we go to the fortran program */
    fclose( logout );
    /* Check for any errors, if so, get out */
    if ( convertStatus )
        return ( convertStatus );
    /* Now construct the the dump filename and converted data filename */
    strcpy( dumpFilename, inputFilename );
    strcat( dumpFilename, ".dump" );
    strcpy( datFilename, inputFilename );
    strcat( datFilename, ".dat" );
```

```
    /* Read and unpack the raw data granule */
    if ( (s8Name = strrchr( inputFilename, '/' ) ) )
        s8Name++;
    else
        s8Name = inputFilename;
    /* Call routine to read and unpack the raw data */
    convertStatus = sids8_( inputFilename, s8Name, datFilename,
        dumpFilename, logFilename, patrcfm );
    /* Check for any conversion errors */
    if ( convertStatus )
    {
        /* If we have an error, return */
        return ( convertStatus );
    }
    /* Reopen the log file, appending to it */
    if ( (logout = fopen(logFilename, "a") ) == NULL )
    {
        printf("Error, mig_driver(), \"Cannot open %s.\"\n",logFilename);
        convertStatus = 1;
    }
    if ( !convertStatus )
    {
        /* Report status information to the operator */
        fprintf(logout, "Informational, mig_driver(), \"Read and unpack of raw
data granule complete.\"\n" );
        /* Set up HDF-EOS data file and check for errors */
        convertStatus = setup( outputHDFFilename );
        /* Report status information to the operator */
        if ( !convertStatus )
            fprintf(logout, "Informational, mig_driver(), \"HDF-EOS file set-
up complete.\"\n");
        else
            fprintf(logout, "Error, mig_driver(), \"Unable to setup HDF-EOS
file.\"\n");
    }
    if ( !convertStatus )
    {
        /* Write the data out from direct access data to HDF-EOS format */
        convertStatus = write_data(logout, outputHDFFilename, datFilename);
        /* Report status information to the operator */
        if ( !convertStatus )
            fprintf(logout,"Informational, mig_driver(), \"HDF-EOS data write
complete.\"\n");
        else
```

```
            fprintf(logout, "Error, mig_driver(), \"Error writing data out
from the direct access data file to HDF-EOS\"\n");
    }
    if ( !convertStatus )
    {
        /* Write the metadata and check for errors */
        convertStatus=metawrite( logout, outputHDFFilename, dumpFilename );
        /* Report status information to the operator */
        if ( !convertStatus )
            fprintf(logout, "Informational, mig_driver(), \"Metadata write
complete.\"\n");
        else
            fprintf(logout, "Error, mig_driver(), \"Metadata write
error.\"\n");
    }
    /* Cleanup by closing the log file and removing temporary files */
    fclose( logout );
    unlink( dumpFilename );
/*
    unlink( datFilename );
 */
    /* Return a success back */
    return( convertStatus );
}
#include "hdf.h"
#include "HdfEosDef.h"
#include <string.h>
/* William E. Smith
   Applied Research Corporation
   October 9, 1996 */
/* Function to create the HDF-EOS swath
   file, define the swath structure dimensions and
   define the swath data fields */
int setup(char *filename)
{
    intn            status;
    int32           swfid, SWid;
    /* Open the HDF file
*/
    swfid = SWopen(filename, DFACC_CREATE);
    /* Create the first swath
*/
    SWid = SWcreate(swfid, "ERBE_S8");
    /* Define the swath dimensions for the S-8 ERBE product fields.
*/
```

```
    /* Dimension for the Satellite Track (max 5400 records)
*/
    status = SWdefdim(SWid, "SatTrack",  5400);
    status = SWdefdim(SWid, "GeoTrack",  5400);
    /* Dimension for the cross track (248)
*/
    status = SWdefdim(SWid, "GeoXtrack", 248 );
    status = SWdefdim(SWid, "SatXtrack", 248 );
    /* Dimension for the scanner operations flag words */
    status = SWdefdim(SWid, "ScanOp", 2);
    /* Dimensions for the scanner radiation flag words */
    status = SWdefdim(SWid, "ScanRad", 18);
/* Setup mapping */
    status = SWdefdimmap(SWid, "GeoTrack", "SatTrack", 0, 1);
    status = SWdefdimmap(SWid, "GeoXtrack", "SatXtrack", 0, 1);
/* Set up satellite parameter fields */
    status = SWdefdatafield(SWid, "JulDay", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "Time", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "EarthSunDist", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCPOSX_Start", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCPOSX_End", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCPOSY_Start", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCPOSY_End", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCPOSZ_Start", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCPOSZ_End", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCVELX_Start", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCVELX_End", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCVELY_Start", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCVELY_End", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCVELZ_Start", "SatTrack",
         DFNT_FLOAT64, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "SCVELZ_End", "SatTrack",
```

```
          DFNT_FLOAT64, HDFE_NOMERGE);
     status = SWdefdatafield(SWid, "SC_Nadir_Colat_Start", "SatTrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     status = SWdefdatafield(SWid, "SC_Nadir_Colat_End", "SatTrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     status = SWdefdatafield(SWid, "SC_Nadir_Lon_Start", "SatTrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     status = SWdefdatafield(SWid, "SC_Nadir_Lon_End", "SatTrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     status = SWdefdatafield(SWid, "Sun_Pos_Colat", "SatTrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     status = SWdefdatafield(SWid, "Sun_Pos_Lon", "SatTrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     status = SWdefdatafield(SWid, "Orbit_Number", "SatTrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     /* Colatitude Field */
     status = SWdefgeofield(SWid, "Colatitude","GeoTrack,GeoXtrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     /* Longitude Field */
     status = SWdefgeofield(SWid, "Longitude","GeoTrack,GeoXtrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
/* Set up scanner fields */
     /* Total radiation Field */
     status = SWdefdatafield(SWid, "Total_Rad","SatTrack,SatXtrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     /* SW radiometric Field */
     status = SWdefdatafield(SWid, "SW_Rad","SatTrack,SatXtrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     /* LW radiometric Field */
     status = SWdefdatafield(SWid, "LW_Rad","SatTrack,SatXtrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     /* Field of View (FOV) Zenith Viewing angle field */
     status = SWdefdatafield(SWid, "FOV_Zenith_Viewing","SatTrack,SatXtrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     /* Field of View (FOV) Zenith Sun angle field */
     status = SWdefdatafield(SWid, "FOV_Zenith_Sun","SatTrack,SatXtrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     /* Field of View (FOV) Relative Azimuth angle field */
     status = SWdefdatafield(SWid, "FOV_Rel_Azimuth","SatTrack,SatXtrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     /* Unfiltered SW field */
     status = SWdefdatafield(SWid, "Unfiltered_SW","SatTrack,SatXtrack",
          DFNT_FLOAT32, HDFE_NOMERGE);
     /* Unfiltered LW field */
     status = SWdefdatafield(SWid, "Unfiltered_LW","SatTrack,SatXtrack",
```

```
         DFNT_FLOAT32, HDFE_NOMERGE);
    /* Top of atmosphere (TOA) SW Estimate field */
    status = SWdefdatafield(SWid, "TOA_EST_SW","SatTrack,SatXtrack",
         DFNT_FLOAT32, HDFE_NOMERGE);
    /* Top of atmosphere (TOA) LW Estimate field */
    status = SWdefdatafield(SWid, "TOA_EST_LW","SatTrack,SatXtrack",
         DFNT_FLOAT32, HDFE_NOMERGE);
    /* ERBE Scene Classification of the scanner FOV */
    status = SWdefdatafield(SWid, "Scanner_FOV_SceneID","SatTrack,SatXtrack",
         DFNT_FLOAT32, HDFE_NOMERGE);
/* Set up flag words */
    /* Scanner operations flags */
    status = SWdefdatafield(SWid, "Scan_Ops_Flags","SatTrack,ScanOp",
         DFNT_INT16, HDFE_NOMERGE);
    /* Scanner radiation flags */
    status = SWdefdatafield(SWid, "Scan_Rad_Flags_Total","SatTrack,ScanRad",
         DFNT_INT16, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "Scan_Rad_Flags_SW","SatTrack,ScanRad",
         DFNT_INT16, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "Scan_Rad_Flags_LW","SatTrack,ScanRad",
         DFNT_INT16, HDFE_NOMERGE);
    status = SWdefdatafield(SWid, "Scan_Rad_Flags_FOV","SatTrack,ScanRad",
         DFNT_INT16, HDFE_NOMERGE);
    SWdetach(SWid);
    SWclose(swfid);
    return(0);
}



#include "hdf.h"
/* William E. Smith
   Applied Research Corporation
   October 8, 1996 */
/* Function to write the data to the HDF-EOS
   swath file */
#define MAXREC 5400
#define MAXFLD 13
int write_data(FILE *logout, char *hdffile, char *infile)
{
    intn            status, i, j, k;
    intn            ifld;
    int32           swfid, SWid;
    int32           start[2],edge[2];
    int32           iflag[2],ibuff[72];
```

```
    int16           iflag16[2],jbuff16[18];
    float64         *trackbuf;
    float32         *trackbuflp,trackbufl;
    float64         track[22];
    float32         data_in[248];
    float32         buff[62];
    FILE            *in;
    struct entry
    {
        char name[20];
    };
    struct entry field[MAXFLD] =
    { "Colatitude", "Longitude",  "Total_Rad", "SW_Rad", "LW_Rad",
      "FOV_Zenith_Viewing", "FOV_Zenith_Sun", "FOV_Rel_Azimuth",
      "Unfiltered_SW", "Unfiltered_LW", "TOA_EST_SW", "TOA_EST_LW",
      "Scanner_FOV_SceneID"};
/* If filename missing or there is a problem, print message and exit */
    if((in = fopen(infile,"rb")) == NULL)
    {
        fprintf(logout,"Error, write_data(), \"Cannot open file %s for
reading\"\n",infile);
        return(1);
    }
    /* Open the HDF file                                                 */
    swfid = SWopen(hdffile, DFACC_RDWR);
    /* Attach the Swath within the file */
    SWid = SWattach(swfid, "ERBE_S8");
/* Read and write data fields */
    start[1]=0;
    edge[0]=1;
    for(j=0;j<MAXREC;++j)
    {
        if( j%100 == 0 )
            printf("Writing for record: %d\n", j);
    /* Read in the spacecraft time and position data */
        status=fread(track,sizeof(float64),22,in);
        if(feof(in) !=0)
        {
            for(i=0;i<22;++i)
                track[i]=32767;
        }
    /* Write out the spacecraft time and position array */
        start[0] = j;
        edge[1] = 1;
        trackbuf = &track[0];
```

```
        status=SWwritefield(SWid,"JulDay",start,NULL,edge,trackbuf);
        trackbuf = &track[1];
        status=SWwritefield(SWid,"Time",start,NULL,edge,trackbuf);
        trackbuf = &track[2];
        status=SWwritefield(SWid,"EarthSunDist",start,NULL,edge,trackbuf);
        trackbuf = &track[3];
        status=SWwritefield(SWid,"SCPOSX_Start",start,NULL,edge,trackbuf);
        trackbuf = &track[4];
        status=SWwritefield(SWid,"SCPOSX_End",start,NULL,edge,trackbuf);
        trackbuf = &track[5];
        status=SWwritefield(SWid,"SCPOSY_Start",start,NULL,edge,trackbuf);
        trackbuf = &track[6];
        status=SWwritefield(SWid,"SCPOSY_End",start,NULL,edge,trackbuf);
        trackbuf = &track[7];
        status=SWwritefield(SWid,"SCPOSZ_Start",start,NULL,edge,trackbuf);
        trackbuf = &track[8];
        status=SWwritefield(SWid,"SCPOSZ_End",start,NULL,edge,trackbuf);
        trackbuf = &track[9];
        status=SWwritefield(SWid,"SCVELX_Start",start,NULL,edge,trackbuf);
        trackbuf = &track[10];
        status=SWwritefield(SWid,"SCVELX_End",start,NULL,edge,trackbuf);
        trackbuf = &track[11];
        status=SWwritefield(SWid,"SCVELY_Start",start,NULL,edge,trackbuf);
        trackbuf = &track[12];
        status=SWwritefield(SWid,"SCVELY_End",start,NULL,edge,trackbuf);
        trackbuf = &track[13];
        status=SWwritefield(SWid,"SCVELZ_Start",start,NULL,edge,trackbuf);
        trackbuf = &track[14];
        status=SWwritefield(SWid,"SCVELZ_End",start,NULL,edge,trackbuf);
        trackbuflp=&trackbufl;
        trackbufl= track[15];
status=SWwritefield(SWid,"SC_Nadir_Colat_Start",start,NULL,edge,trackbuflp);
        trackbufl= track[16];
status=SWwritefield(SWid,"SC_Nadir_Colat_End",start,NULL,edge,trackbuflp);
        trackbufl= track[17];
status=SWwritefield(SWid,"SC_Nadir_Lon_Start",start,NULL,edge,trackbuflp);
        trackbufl= track[18];
status=SWwritefield(SWid,"SC_Nadir_Lon_End",start,NULL,edge,trackbuflp);
        trackbufl= track[19];
status=SWwritefield(SWid,"Sun_Pos_Colat",start,NULL,edge,trackbuflp);
        trackbufl= track[20];
        status=SWwritefield(SWid,"Sun_Pos_Lon",start,NULL,edge,trackbuflp);
        trackbufl= track[21];
status=SWwritefield(SWid,"Orbit_Number",start,NULL,edge,trackbuflp);
    /* Process the scanner geoposition and data fields */
        edge[1]=248;
```

```
        for(ifld=0;ifld<MAXFLD;++ifld)
        {
            status=fread(data_in,sizeof(float32),248,in);
            if(feof(in) !=0)
            {
                for(i=0;i<248;++i)
                    data_in[i]=32767.;
            }
            status=SWwritefield(SWid,field[ifld].name,start,
                NULL,edge,data_in);
        }
        /* Read the scanner operations flag words */
        status=fread(iflag,sizeof(int32),2,in);
        start[0]=j;
        if(feof(in) !=0)
        {
            for(i=0;i<2;++i)
                iflag16[i]=32767;
        }
        else
        {
            for(i=0;i<2;++i)
                iflag16[i]=iflag[i];
        }
        /* Write the scanner operations flag words */
        edge[1]=2;
        status=SWwritefield(SWid,"Scan_Ops_Flags",start,NULL,edge,iflag16);
        /* Read the scanner radiation flag words */
        status=fread(ibuff,sizeof(int32),72,in);
        if(feof(in) !=0)
        {
for(i=0;i<72;++i)
                ibuff[i]=32767;
        }
        /* Write the scanner radiation flag words */
        edge[1]=18;
        for(k=0;k<4;++k)
        {
            for(i=0;i<18;++i)
                jbuff16[i]=ibuff[i+k*18];
            if(k==0)
                status=SWwritefield(SWid,"Scan_Rad_Flags_Total",
                    start,NULL,edge,jbuff16);
            if(k==1)
                status=SWwritefield(SWid,"Scan_Rad_Flags_SW",
```

```
                    start,NULL,edge,jbuff16);
            if(k==2)
                status=SWwritefield(SWid,"Scan_Rad_Flags_LW",
                    start,NULL,edge,jbuff16);
            if(k==3)
                status=SWwritefield(SWid,"Scan_Rad_Flags_FOV",
                    start,NULL,edge,jbuff16);
        }
    }
    fclose(in);
    SWdetach(SWid);
    SWclose(swfid);
    return(0);
}



#include <time.h>
#include "PGS_MET.h"
#include "PGS_PC.h"
#include "hdf.h"
#include "HdfEosDef.h"
#include "InPpCommonExtSciData.h"
/*
   William E. Smith, Applied Research Corporation
   Peter Chiang, Lockheed-Martin
   September 23, 1996
*/
```

7.2.1.2 A FORTRAN Example of a Simple Swath Creation

```
c simpleswath.f
c In this program we create, define and write a simple swath hdfeos file
c using the swath interface
   program simpleswath
   integer index1, index2, swfid, swid, status
   integer start(2), stride(2), edge(2), attr(4)
   integer swdefdim, swdetach, swdefgfld, swdefdfld
   integer swdefmap, swwrfld, swwrattr, swclose
   integer*4 swopen, swcreate, swattach
   real ray1(40,40), ray2(40,40), lat(40,40), lon(40,40)
   real latcnt, loncnt, raycnt
   integer DFACC_CREATE
   parameter (DFACC_CREATE=4)
   integer DFNT_FLOAT32
   parameter (DFNT_FLOAT32=5)
   integer DFNT_INT32
   parameter (DFNT_INT32=24)
   integer HDFE_NOMERGE
   parameter (HDFE_NOMERGE=0)
   integer HDFE_AUTOMERGE
   parameter (HDFE_AUTOMERGE=1)
   raycnt=-799.0
   latcnt=38.0
   loncnt=75.0
   attr(1)=11
   attr(2)=33
   attr(3)=66
   attr(4)=99
   start(1)=0
   start(2)=0
   stride(1)=1
   stride(2)=1
   edge(1)=40
   edge(2)=40
c This section of the program just fills some arrays with data that will be
c used later in the program
   do 110 index1=1,40
     do 100 index2=1,40
      ray1(index1, index2)=raycnt
      ray2(index1, index2)=raycnt + 1.0
      lat(index1, index2)=latcnt
      lon(index1, index2)=loncnt
c      write(*,*)'Lat =',latcnt,' Lon =',loncnt,' Array =', raycnt
```

```
      raycnt = raycnt +1.
 100  continue
 110 continue
c First open the HDF file, "SimpleSwathf.hdf", the file doesn't already exist
c so we use the DFACC_CREATE access code in the open statement. The routine
c returns the swath file id, swfid, which is used throughout the program
c to identify the file in subsequent routines
   swfid=swopen("SimpleSwathf.hdf", DFACC_CREATE)
   write(*,*)'Value returned by swopen ', swfid
c The first of these, swcreate, creates the swath, "Simplef", within the
c file designated by the file id, swfid. It returns the swath id, swid,
c which identifies the swath in subsequent routines. We will show how
c to define, write and read field swaths in later programs.
   swid=swcreate(swfid, "Simplef")
   write(*,*)'Value returned by swcreate ', swid
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 position1 and
c position2 dimensions with size 40 and two dimensions corresponding to these.
   status=swdefdim(swid, "Voltage", 40)
   write(*,*)'Value returned by swdefdim ',status
   status=swdefdim(swid, "Bias", 40)
   write(*,*)'Value returned by swdefdim ',status
   status=swdefdim(swid, "Position1", 40)
   write(*,*)'Value returned by swdefdim ',status
   status=swdefdim(swid, "Position2", 40)
   write(*,*)'Value returned by swdefdim ',status
c In this section, we define geolocated and data fields using dimensions
c defined above. In this particular example, Postiton1 dimension is the
c track dimension for geolocation fileds and Voltage is the track
c dimension for data fields
   status=swdefgfld(swid, "Latitude", "Position1,Position2",
   1         DFNT_FLOAT32, HDFE_AUTOMERGE)
   write(*,*)'Value returned by swdefgfld ',status
   status=swdefgfld(swid, "Longitude", "Position1,Position2",
   1         DFNT_FLOAT32,HDFE_AUTOMERGE)
   write(*,*)'Value returned by swdefgfld ',status
   status=swdefdfld(swid, "Temperature", "Voltage,Bias",
   1         DFNT_FLOAT32, HDFE_NOMERGE)
   write(*,*)'Value returned by swdefdfld ',status
   status=swdefdfld(swid, "Conduction", "Voltage,Bias",
   1         DFNT_FLOAT32, HDFE_NOMERGE)
   write(*,*)'Value returned by swdefdfld ',status
```

```
c Here we complete the definition phase of the swath by defining the
c mapping relationships of the track and cross-track dimensions
c In this example, the relationship is trivial, a one-to-one, starting
c at the first element and increment to the next value
   status=swdefmap(swid, "Position1", "Voltage", 0, 1)
   write(*,*)'Value returned by swdefmap ',status
   status=swdefmap(swid, "Position2", "Bias", 0, 1)
   write(*,*)'Value returned by swdefmap ',status
c We now detach from the swath interface, this step is necessary to properly
c store the swath information within the file AND MUST BE DONE BEFORE
c WRITING OR READING DATA TO OR FROM THE SWATH
   status=swdetach(swid)
   write(*,*)'Value returned by swdetach ',status
   swid=swattach(swfid, "Simplef")
   write(*,*)'Value returned by swattach ',swid
c After attaching to the swath defined above, we write data to the fields
c using the arrays we filled earlier in the program. We write to the fields
c by giving the element to start with, the number of elements to stride(skip)
c and the total number of elements to write to, then with provide the variable
c containing the data
   status=swwrfld(swid, "Temperature", start, stride, edge, ray1)
   write(*,*)'Value returned by swwrfld ',status
   status=swwrfld(swid, "Latitude", start, stride, edge, lat)
   write(*,*)'Value returned by swwrfld ',status
   status=swwrfld(swid, "Longitude", start, stride, edge, lon)
   write(*,*)'Value returned by swwrfld ',status
c This line of code is just an example of using the function swwrattr
   status=swwrattr(swid, "Drift", DFNT_INT32, 4, attr)
   write(*,*)'Value returned by swwrattr ',status
   status=swdetach(swid)
   write(*,*)'Value returned by swdetach ',status
   status=swclose(swfid)
   write(*,*)'Value returned by swclose ',status
   end
```

7.2.2 A 'Real World' Example of a Swath Creation

The following C program is derived from a program written for the “ECS V0 Data Migration” effort. It creates, defines, and writes a Swath data set to an HDF-EOS file using the HDF-EOS Swath interface. The program converts an ERBE Cloud data set from a native format into HDF-EOS. Only the portion of the code dealing with writing the Swath through HDF-EOS has been reproduced here.

Note: This is a test example, it is presented for purposes of providing a guide for users of HDF-EOS. The example has yet not been validated by the LaRC DAAC or users of ERBE data.

```
#include <string.h>
#include "hdf.h"
/* Driver for the set up of the swath HDF-EOS file
  and the transfer of the data to that file */
main(int argc, char *argv[])
{
  int setup(char *filename);
  int write_data(char *filename);
  intn      status;
/* If filename neglected, print message and exit */
  if(argc != 2)
  {
    printf("Filename missing\n");
    exit(1);
  }
/* Setup the Swath HDFEOS file */
  status = setup(argv[1]);
/* Read in the native data and write it to the
  Swath HDFEOS file */
  status = write_data(argv[1]);
}
/* Function to create and set up and HDF-EOS swath
  file, define the swath structure dimensions and
  define the swath HDF-EOS data fields */
int setup(char *filename)
{
  intn      status;
  int32      swfid, SWid;
  char hdf[]={".hdf"};
  char hdfile[80];
  /* Add hdf extension to filename to create swath filename */
  strcpy(hdfile,filename);
  strcat(hdfile,hdf);
  /* Open the HDF file                            */
```

```
  swfid = SWopen(hdfile, DFACC_CREATE);
  /* Create the first swath                        */
  SWid = SWcreate(swfid, "Swath1");
  /* Define the swath dimensions for the S-8 ERBE product fields.     */
  /* Dimension for the Satellite Track (max 5400 records)         */
  status = SWdefinedim(SWid, "SatTrack", 5400);
  /* Dimension for the cross track (62 every 4 seconds )          */
  status = SWdefinedim(SWid, "SatXtrack", 62);
  /* Dimension for the cross track stack (4 cross tracks to cover 16 secs */
  status = SWdefinedim(SWid, "Satstack",4);
/* Set up Geolocation fields */
  /* Colatitude Field */
  status = SWdefgeofield(SWid, "Colatitude","SatTrack,SatXtrack,Satstack",
     DFNT_FLOAT32);
  /* Longitude Field */
  status = SWdefgeofield(SWid, "Longitude","SatTrack,SatXtrack,Satstack",
     DFNT_FLOAT32);
/* Set up scanner fields */
  /* Total radiation Field */
  status = SWdefdatafield(SWid, "Total_Rad","SatTrack,SatXtrack,Satstack",
     DFNT_FLOAT32);
/* Detach and close swath file */
  SWdetach(SWid);
  SWclose(swfid);
  return(SWid);
}
/* Function to write the data to the HDF-EOS
  swath file */
#define MAXREC 5400
int write_data(char *filename)
{
  intn      status, i, j, k;
  int32      swfid, SWid;
  int32      start[3],edge[3];
  float32     data_in[248];
  float32     buff[62];
  char dat[]={".dat"};
  char hdf[]={".hdf"};
  char hdfile[80];
  char infile[80];
  FILE      *in;
  char fldname[20]="Total_Rad";
/* Setup input and output filenames */
  strcpy(hdfile,filename);
  strcpy(infile,filename);
```

```
  strcat(hdfile,hdf);
  strcat(infile,dat);
/* If filename missing or there is a problem, print message and exit */
  if((in = fopen(infile,"rb")) == NULL)
  {
    printf("Cannot open file.\n");
    exit(2);
  }
  /* Open the HDF file  */
  swfid = SWopen(hdfile, DFACC_RDWR);
  /* Attach the Swath within the file */
  SWid = SWattach(swfid, "Swath1");
/* Read and write data fields */
/* Set HDFEOS file indices */
  start[1]=0;
  edge[0]=1;
  edge[1]=62;
  edge[2]=1;
  for(j=0;j<MAXREC;++j)
  {
    start[0] = j;
  /* Read in the data in native format */
    status=fread(data_in,sizeof(float32),248,in);
    for(k=0;k<4;++k)
    {
      start[2]=k;
      for(i=0;i<62;++i)
        buff[i]=data_in[i+k*62];
  /* Write out the data field in HDFEOS format */
      status=SWwritefield(SWid,fldname,start,
        NULL,edge,buff);
    }
  }
  fclose(in);
  SWdetach(SWid);
  SWclose(swfid);
  return(SWid);
}
```

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. The section concludes with a real world example taken from the ECS “V0 Data Migration” effort..

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. Both programs assume that the data array has been projected using GCTP’s Universal Transverse Mercator projection, zone 18.

7.3.1.1 A C Example of a Simple Grid Creation

```
/* SimpleGrid.c
** In this program we create, define and write a simple grid hdfeos file
** using the grid interface
*/
#include <stdio.h>
#include "mfhdf.h"
#include "hdf_eos_def.h"
main()
{
  int32 index1 = 0;
  int32 index2 = 0;
  int32 rank = 2;
  int32 dimsizes[2] = {40,40};
  float32 temp[40][40], array[40][40];
  float32 fillvalue = 1996.6991;
  float32 raycnt = -799.0;
  int cnt;
  intn status;
  int32 gdfid, GDid, dim_id, i, zonecode, attr[4] = {11,33,66,99};
  int32 projcode, spherecode, xdim, ydim;
  int32 start[2] = {0, 0};
  int32 stride[2] = {1, 1};
  int32 edge[2] = {40, 40};
  float64 projparm[16], uplft[2], lowrgt[2];
  float lat[40][40];
  float latcnt = 38.0;
  float lon[40][40];
  float loncnt = 75.0;
/*
** This section of the program just fills some arrays with data that we will
** use later in the program
*/
while(index1 < 40) {
   while(index2 < 40) {
```

```
     temp[index1][index2] = raycnt;
     array[index1][index2] = raycnt + 1.;
     lat[index1][index2] = latcnt;
     lon[index1][index2] = loncnt;
/*     printf("Lat %f Lon %f\n", latcnt, loncnt); */
     index2++;
     raycnt++;
   }
   latcnt = latcnt + 0.05;
   loncnt = loncnt + 0.1;
   index1++;
   index2 = 0;
  }
/*
** First open the HDF file, "SimpleGrid.hdf", the file doesn't already exist
** so we use the DFACC_CREATE access code in the open statement. The routine
** returns the grid file id, gdfid, which is used throughout the program
** to identify the file in subsequent routines
*/
  gdfid = GDopen("SimpleGrid.hdf", DFACC_CREATE);
  printf("Handle id returned from GDopen %d\n", gdfid);
/*
** Create UTM Grid
** The region is bounded by 75.0 E to 78.6 E longitude & 38.0 N to 39.8 N
** latitude, UTM Zone 18
** Use default spheroid (Clarke 1866) spherecode = 0
** grid into 40 bins along x-axis by 40 bins along y-axis
*/
  zonecode = 18;
  spherecode = 0;
  xdim = 40;
  ydim = 40;
/* Upper Left & Lower Right corners in meters
*/
  uplft[0] = 243893.62149;
  uplft[1] = 4431859.13218;
  lowrgt[0] = 763427.96361;
  lowrgt[1] = 4209857.46415;
  GDid = GDcreate(gdfid, "UTMGrid", xdim, ydim, uplft, lowrgt);
  printf("Handle returned from GDcreate %d\n", GDid);
  status = GDdefproj(GDid, GCTP_UTM, zonecode, spherecode, projparm);
  printf("Returned value from GDdefproj %d\n", status);
/*
** This section of code gives examples of how to use other routines
** in the grid interface
```

```
*/
/* This function is not implemented at this time April/96
**  status = GDdefpixreg(GDid, HDFE_UPLEFT);
*/  printf("Returned from GDdefpixreg %d\n", status);
  status = GDdefdim(GDid, "Burrito", 25);
  printf("Returned value from GDdefdim %d\n", status);
  status = GDdetach(GDid);
  HEprint(stdout,0);
  printf("Returned value from GDdetach %d\n", status);
  GDid = GDattach(gdfid, "UTMGrid");
  printf("Handle returned from GDattach %d\n", GDid);
  status = GDdeffield(GDid, "TexasTaco", "XDim,YDim",
             DFNT_FLOAT32, HDFE_AUTOMERGE);
  printf("Value returned from GDdeffield %d\n", status);
  status = GDdeffield(GDid, "4AlarmTaco", "XDim,YDim",
             DFNT_FLOAT32, HDFE_AUTOMERGE);
  printf("Value returned from GDdeffield %d\n", status);
  status = GDdeffield(GDid, "GreenOnions", "XDim,YDim", DFNT_FLOAT32,
           HDFE_AUTOMERGE);
  printf("Value returned from GDdeffield %d\n", status);
  status = 0;
  status = GDdetach(GDid);
  HEprint(stdout,0);
  printf("Returned value from GDdetach %d\n", status);
/*
** We now detach from the grid interface, this step is necessary
** to properly store the grid information within the file AND MUST BE DONE
**   BEFORE   WRITING   OR   READING   DATA   TO   OR   FROM   THE   GRID
================================================================
** In this example We attach to the grid to write, so this is a complete
** example of creating and writing the grid
** We attach to the grid we just created
*/
  GDid = GDattach(gdfid, "UTMGrid");
  printf("Handle returned from GDattach %d\n", GDid);
/*
** We write to fields we just defined with arrays we created early in the
** program
*/
  status = GDwritefield(GDid, "TexasTaco", start, stride, edge, temp);
  printf("Returned from GDwritefield %d\n", status);
  status = GDwritefield(GDid, "4AlarmTaco", start, stride, edge, array);
  printf("Returned from GDwritefield %d\n", status);
  status = GDsetfillvalue(GDid, "GreenOnions", &fillvalue);
  printf("Returned from GDsetfillvalue %d\n", status);
  status = GDwriteattr(GDid, "TacoBell", DFNT_INT32, 4, attr);
  printf("Returned from GDwriteattr %d\n", status);
  status = GDdetach(GDid);
  printf("Returned from GDdetach %d\n", status);
  status = GDclose(gdfid);
  printf("Returned from GDclose %d\n", status);
}
```

7.3.1.2 A FORTRAN Example of a Simple Grid Creation

```
c simplegrid.f
c In this program we create, define and write a simple grid hdfeos file
c using the grid interface
   program simplegrid
   integer index1, index2, gddetach, gdclose, gddefproj, gddeffld
   integer gdfid, gdid, gdopen, gdcreate, gdattach
   integer zonecode, spherecode, xdim, ydim, gddefdim, gdwrfld
   integer status, gdwrattr, gdsetfill
   integer start(2), stride(2), edge(2)
   real latcnt, loncnt
   real lat(40,40), lon(40,40), ray1(40,40), ray2(40,40)
   real attr(4)
   real raycnt
   real fillvalue
   double precision uplft(2), lowrgt(2)
   integer DFACC_CREATE
   parameter (DFACC_CREATE=4)
   integer DFACC_RDWR
   parameter (DFACC_RDWR=3)
   integer GCTP_UTM
   parameter (GCTP_UTM=1)
   integer DFNT_FLOAT32
   parameter (DFNT_FLOAT32=5)
   integer DFNT_INT32
   parameter (DFNT_INT32=24)
   integer HDFE_AUTOMERGE
   parameter (HDFE_AUTOMERGE=1)
   raycnt = -799.0
   latcnt = 38.0
   loncnt = 75.0
   fillvalue=1996.6991
   attr(1)=11
   attr(2)=33
   attr(3)=66
   attr(4)=99
   start(1)=0
   start(2)=0
   stride(1)=1
   stride(2)=1
   edge(1)=40
   edge(2)=40
c This section of the program just fills some arrays with data that we will
c use later in the program
```

```
   do 110 index1 = 1, 40
     do 100 index2 = 1, 40
      ray1(index1, index2) = raycnt
      ray2(index1, index2) = raycnt + 1.0
      lat(index1, index2) = latcnt
      lon(index1, index2) = loncnt
      write(*,*)'Lat =',latcnt, ', Lon =',loncnt
      raycnt = raycnt + 1
 100  continue
     latcnt = latcnt + 0.05
     loncnt = loncnt + 0.1
     index 2 =1
 110 continue
c First open the HDF file, "SimpleGrid.hdf", the file doesn't already exist
c so we use the DFACC_CREATE access code in the open statement. The routine
c returns the grid file id, gdfid, which is used throughout the program
c to identify the file in subsequent routines
   gdfid = gdopen("SimpleGridf.hdf", DFACC_CREATE)
   write(*,*)'Value returned by gdopen ',gdfid
c Create UTM Grid
c The region is bounded by 75.0 E to 78.6 E longitude & 38.0 N to 39.8 N
c latitude, UTM Zone 18
c Use default spheroid (Clarke 1866) spherecode = 0
c grid into 40 bins along x-axis by 40 bins along y-axis
   zonecode = 18
   spherecode = 0
   xdim = 40
   ydim = 40
c Upper Left & Lower Right corners in meters
c ==========================================
   uplft(1) = 243893.62149
   uplft(2) = 4431859.13218
   lowrgt(1) = 763427.96361
   lowrgt(2) = 4209857.46415
   gdid = gdcreate(gdfid, "UTMGridf", xdim, ydim, uplft, lowrgt)
   write(*,*)'Value returned by gdcreate ', gdid
   status = gddefproj(gdid, GCTP_UTM, zonecode, spherecode, projparm)
   write(*,*)'Value returned by gddefproj ',status
c This section of code gives examples of how to use other routines
c in the grid library
   status = gddefdim(gdid, "Burrito", 25)
   write(*,*)'Value returned by gddefdim ', status
   status = gddeffld(gdid, "TexasTaco", "XDim,YDim",
   1           DFNT_FLOAT32, HDFE_AUTOMERGE)
   write(*,*)'Value returned by gddeffld ',status
```

```
   status = gddeffld(gdid, "4AlarmTaco", "XDim,YDim",
   1           DFNT_FLOAT32, HDFE_AUTOMERGE)
   write(*,*)'Value returned by gddeffld ',status
   status = gddeffld(gdid, "GreenOnions", "XDim,YDim",
   1           DFNT_FLOAT32, HDFE_AUTOMERGE)
   write(*,*)'Value returned by gddeffld ',status
   status = gddetach(GDid)
   write(*,*)'Value returned by gddeatch ',status
   status = gdclose(gdfid)
   write(*,*)'Value returned by gdclose ', status
c We now detach from and close the grid interface, this step is necessary
c to properly store the grid information within the file AND MUST BE DONE
c BEFORE WRITING OR READING DATA TO OR FROM THE GRID
=================================================================
c In this example We open the file to write to the grid, so this is a omplete
c example of creating and writing the grid
c
c We reopen and attach to the grid we just created
   gdfid = gdopen("SimpleGridf.hdf", DFACC_RDWR)
   write(*,*)'Value returned by gdopen ',gdfid
   gdid = gdattach(gdfid, "UTMGridf")
   write(*,*)'Value returned by gdattach ',gdid
c We write to fields we just defined with arrays we created early in the
c program
   status = gdwrfld(gdid, "TexasTaco", start, stride, edge, ray1)
   write(*,*)'Value returned by gdwrfld ',status
   status = gdwrfld(gdid, "4AlarmTaco", start, stride, edge, ray2)
   write(*,*)'Value returned by gdwrfld ',status
   status = gdsetfill(gdid, "GreenOnions", fillvalue)
   write(*,*)'Value returned by gdsetfill ', status
   status = gdwrattr(gdid, "TacoBell", DFNT_INT32, 4, attr)
   write(*,*)'Value returned by gdwrattr ', status
   status = gddetach(GDid)
   write(*,*)'Value returned by gddeatch ',status
   status = gdclose(gdfid)
   write(*,*)'Value returned by gdclose ', status
   end
```

7.3.1.3 Performing Geographic and Vertical Subsetting on a 3 Dimensional Grid

The following C program creates, define, write a three dimensional Griddata set to an HDF-EOS file using the HDF-EOS Grid interface. Finally thisprogram performs the geographic and vertical subsetting on this grid.

```
#include "hdf.h"
#include "HdfEosDef.h"
#include <math.h>
main()
{
    intn            status, i, j, l;
    int32           gdfid, GDid, dummy, regionID, size, dims[8], ntype,
       rank;
    int32           xdim, ydim, zonecode, projcode, spherecode;
    int32           *datbuf32;
    float64         cornerlon[2], cornerlat[2], range[2];
    float64         projparm[16], uplft[2], lowrgt[2];
    int32           co2[100*100*5];
    float64         alt[5] = {2.0,3.0,4.0,5.0,6.0}
    /* This part of the code fills the array with data that will be
       written to the grid */
    for (i=0; i<100*100*5; i++)
        co2[i] = i;
    /*
     * We first open the HDF grid file, "GridFile.hdf".  Because this file
     * does not already exist, we use the DFACC_CREATE 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 = GDopen("GridFile.hdf", DFACC_CREATE);
    printf("Handle id returned from GDopen %d\n", gdfid);
    /*
     * Create polar stereographic grid
     * Northern Hemisphere  (True Scale at 40 N, 0 Longitude below pole)
     * Use International 1967 spheriod (spherecode = 0)
     * Grid into 100 bins along x-axis and y-axis
     */
    spherecode = 0;
    /* 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] = 000000000.00;
projparm[5] = 40000000.00;
xdim = 100;
ydim = 100;
/* Create the PolarGrid */
GDid = GDcreate(gdfid, "PolarGrid", xdim, ydim, NULL, NULL);
printf("Handle returned from GDcreate %d\n", GDid);
/* Define projection parameters */
status = GDdefproj(GDid, GCTP_PS, dummy, spherecode, projparm);
printf("Returned value from GDdefproj %d\n", status);
/* Define the origin of the grid */
status = GDdeforigin(GDid, HDFE_GD_UL);
printf("Returned value from GDdeforigin %d\n", status);
/* Define the vertical dimension */
status = GDdefdim(GDid, "VertDim", 5);
printf("Returned value from GDdefdim %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.
 */
GDdetach(GDid);
GDid = GDattach(gdfid, "PolarGrid");
/* Define a field within in the PolarGrid */
status = GDdeffield(GDid, "CO2_NOx", "VertDim,YDim,XDim",
                       DFNT_INT32, HDFE_AUTOMERGE);
printf("Value returned from GDdeffield %d\n", status);
status = GDdeffield(GDid, "Altitude", "VertDim",
                       DFNT_FLOAT64, HDFE_NOMERGE);
printf("Value returned from GDdeffield %d\n", status);
GDdetach(GDid);
GDid = GDattach(gdfid, "PolarGrid");
/* Write the data into the grid fields */
if (GDid != -1)
{
    status = GDwritefield(GDid, "Altitude",
                          NULL, NULL, NULL, alt);
    printf("Returned from GDwritefield %d\n", status);
    status = GDwritefield(GDid, "CO2_NOx",
                          NULL, NULL, NULL, co2);
    printf("Returned from GDwritefield %d\n", status);
}
GDdetach(GDid);
GDid = GDattach(gdfid, "PolarGrid");
/* Perform the geographic and vertical subsetting on the grid */
```

```
    if (GDid != -1)
    {
            /* Define the box for geographic subsetting region */
            cornerlon[0] = -90.;
          cornerlat[0] = 0.;
          cornerlon[1] = 180;
          cornerlat[1] = 90.;
          regionID = GDdefboxregion(GDid, cornerlon, cornerlat);
            /* Define the vertical subsetting region */
            range[0] = 5.5;
            range[1] = 14.5;
            regionID = GDdefvrtregion(GDid, regionID, "Altitude", range);
            printf("Returned value from GDdefvrtregion %d\n", regionID);
            /* Obtain the size required to store the subsetted data */
          status = GDregioninfo(GDid, regionID, "CO2_NOx", &ntype,
                        &rank, dims, &size, upleft, lowright);
            printf("Returned value from GDregioninfo %d\n", status);
            /* Allocate the buffer to store the subsetted data */
          datbuf32 = (int32 *) calloc(size, 1);
            /* Extract the data obtained from both geographic and vertical
               subsetting into the buffer */
          status = GDextractregion(GDid, regionID, "CO2_NOx", datbuf32);
            printf("Returned value from GDextractregion %d\n", status);
            /* Dump the subsetted data from the buffer */
            for(l=0; l<size/4; l++)
                  printf("value of %d = %d\n",l,*(datbuf32 + l));
          free(datbuf32);
    }
    GDdetach(GDid);
    GDclose(gdfid);
    return;
}
```

7.3.1.4 A 'Real World' Example of a Grid Creation

The following excerpt from a C++ program creates, defines, and writes a Grid data set to an HDF-EOS file using the HDF-EOS Grid interface. The excerpt has been taken from a program used to convert a data set in the Geographic projection from the National Meteorological Center from a native format into HDF-EOS. Only the portion of the code dealing with writing the Grid through HDF-EOS has been reproduced here.

The code presented here has been written within ECS for writing incoming NCEP GRID formatted data into HDF-EOS. Each incoming product contains a number of parameters at differing atmospheric levels or layers on different geographic grids.

Following unpacking of the data from the native format, a number of grids are created, one for each parameter on a specific grid. The grid name chosen for each is made up from the abbreviated parameter name and the grid identifier (as defined in the source document for the native format file) in the form parameter_gridID.

For each grid, data is then written to a number of two dimensional fields, one for each atmospheric level. This field is named to give an indication of the level or layer in the atmosphere appropriate for that field, for example "isobaric 1000 hPa level" or "tropopause" or "isobaric layer between 100hPa and 950hPa".

```
static const char* ownerShipStr = "ECS Software - Property of US Government";
#ifdef __cplusplus
extern “c”{
#endif
#include <hdf.h>
#include <mfhdf.h>
#include <HdfEosDef.h>
#ifdef __cplusplus
}
#endif
//
// WriteSDS
//
InPpGRIBHDF::WriteSDS()
{
int32  fid;    // HDF File identifier
int32  gctpGridId;  // Grid identifier from HDF-EOS toolkit
int32  gctpGridHandle; // Grid handle returned from HDF-EOS
int32  hdfNumberType = DFNT_FLOAT32; // Number type for of science data
EcTInt  errorStatus; // Error status
float64 ulCorner[2]; // Coordinates of upper left corner
float64 lrCorner[2]; // Coordinates of lower right corner
float64 projParameters[16]; // GCTP projection parameters
float64 hdfBoundary[4];   // Geographic boundary for HDF data set
```

```
// Uses HDF-EOS grid library calls
// Open the file
  if ((fid = GDopen(InPpGRIBScienceData::myOutputFile, DFACC_RDWR)) == -1)
  {
    fprintf (stderr, "Error opening file to write grids.\n");
    return (InCPpADPPFail);
  }
// Need to loop through for each SDS found
  for (EcTInt i=0; i< myNonSearchableMetadata.nSDS; i++)
  {
  // Find the grid projection
    if ( strcmp (myNonSearchableMetadata.nonSearchableSDSData[i].
        projectionName, "PLATE CARREE" ) == 0 )
    {
      gctpGridId = InCPpLatLon;
    }
    else if ( strcmp (myNonSearchableMetadata.nonSearchableSDSData[i].
           projectionName, "MERCATOR" ) == 0 )
    {
      gctpGridId = InCPpMercator;
    }
    else if ( strcmp (myNonSearchableMetadata.nonSearchableSDSData[i].
           projectionName, "LAMBERT" ) == 0 )
    {
      gctpGridId = InCPpLambert;
    }
    else if ( strcmp (myNonSearchableMetadata.nonSearchableSDSData[i].
           projectionName, "GNOMIC" ) == 0 )
    {
      gctpGridId = InCPpGnomic;
    }
    else if ( strcmp (myNonSearchableMetadata.nonSearchableSDSData[i].
           projectionName, "ORTHOGRAPHIC" ) == 0 )
    {
      gctpGridId = InCPpOrthographic;
    }
    else if ( strcmp (myNonSearchableMetadata.nonSearchableSDSData[i].
           projectionName, "POLAR STEREOGRAPHIC" ) == 0 )
    {
      gctpGridId = InCPpPStereographic;
    }
    else
    {
      fprintf (stderr, "Error identifying GCTP grid type.\n");
      return InCPpADPPFail;
```

```
    }
  // Get the bounding coordinates for the grid in gctp terms
    for (EcTInt j=0; j<InCPpNumberOfDefinedGrids; j++)
    {
if (myNonSearchableMetadata.nonSearchableSDSData[i].gridId ==
        InCPpGridCoordinates[j].gridId)
      {
        ulCorner[0] = InCPpGridCoordinates[j].upperLeftLon;
        ulCorner[1] = InCPpGridCoordinates[j].upperLeftLat;
        lrCorner[0] = InCPpGridCoordinates[j].lowerRightLon;
        lrCorner[1] = InCPpGridCoordinates[j].lowerRightLat;
      }
    }
  // Fill in the required GCTP parameters
    projParameters[0] = radiusOfEarth;
    projParameters[1] = 0;
    projParameters[6] = 0;
    projParameters[7] = 0;
    projParameters[8] = 0;
    projParameters[9] = 0;
    projParameters[10] = 0;
    projParameters[11] = 0;
    projParameters[12] = 0;
    for (j=0; j<InCPpNumberOfDefinedGrids; j++)
    {
      if ( InCPpGCTPparms[j].gridId ==
          myNonSearchableMetadata.nonSearchableSDSData[i].gridId)
      {
        projParameters[2] = InCPpGCTPparms[j].stdpar1;
        projParameters[3] = InCPpGCTPparms[j].stdpar2;
        projParameters[4] = InCPpGCTPparms[j].centLon;
        projParameters[5] = InCPpGCTPparms[j].originLat;
        break;
      }
    }
  // Fill in the boundary values for the grid
    hdfBoundary[0] = myNonSearchableMetadata.nonSearchableSDSData[i].
              northmostLat;
    hdfBoundary[1] = myNonSearchableMetadata.nonSearchableSDSData[i].
              eastmostLon;
    hdfBoundary[2] = myNonSearchableMetadata.nonSearchableSDSData[i].
              southmostLat;
    hdfBoundary[3] = myNonSearchableMetadata.nonSearchableSDSData[i].
              westmostLon;
  // Create the required grid
```

```
    if ( ( gctpGridHandle = GDcreate(fid,
         &(myNonSearchableMetadata.nonSearchableSDSData[i].sdsLabel[0]),
         mystructuralMetadata.sdsData[i].gridInfo.iXDimensionSize,
         mystructuralMetadata.sdsData[i].gridInfo.iYDimensionSize,
         ulCorner, lrCorner) ) == -1)
    {
fprintf (stderr, "Error creating grid %s\n",
        myNonSearchableMetadata.nonSearchableSDSData[i].sdsLabel);
      return (InCPpADPPFail);
    }
  // Define the projection for this grid
    if ( ( errorStatus = GDdefproj (gctpGridHandle, gctpGridId, 0,
         0, projParameters) ) == -1)
    {
      fprintf (stderr, "Error defining projection %s\n",
        myNonSearchableMetadata.nonSearchableSDSData[i].sdsLabel);
      return (InCPpADPPFail);
    }
  // For each level, define the field
    for (j=0; j<mystructuralMetadata.sdsData[i].zSize; j++)
    {
      if ( ( errorStatus = GDdeffield(gctpGridHandle,
           myNonSearchableMetadata.
              nonSearchableSDSData[i].verticalExtents[j],
           "YDim,XDim",
           hdfNumberType, HDFE_AUTOMERGE)
         ) == -1 )
      {
        fprintf (stderr, "Error defining grid field.\n");
        return (InCPpADPPFail);
      }
    }
  // Detach from the grid and re-attach for writing the data
    if ( (errorStatus = GDdetach(gctpGridHandle) ) == -1)
    {
      fprintf (stderr, "Error detaching from grid.\n");
        HEprint(stdout, 0);
      return (InCPpADPPFail);
    }
    if ( (gctpGridHandle = GDattach(fid,
         myNonSearchableMetadata.
         nonSearchableSDSData[i].sdsLabel) ) == -1)
    {
      fprintf (stderr, "Error attaching to grid.\n");
      return (InCPpADPPFail);
```

```
    }
  // Now we can write the data for the grid
    for (j=0; j<mystructuralMetadata.sdsData[i].zSize; j++)
    {
      if ( ( errorStatus = GDwritefield( gctpGridHandle,
           myNonSearchableMetadata.
              nonSearchableSDSData[i].verticalExtents[j],
           (int32*)NULL, (int32*)NULL, (int32*)NULL,
           (VOIDP)mySDSArray[myNonSearchableMetadata.
              nonSearchableSDSData[i].recordsIncluded[j]] )
         ) == -1 )
      {
        fprintf (stderr, "Error defining grid field.\n");
        return (InCPpADPPFail);
      }
    }
  // Finally detach from the current grid.
    if ( ( errorStatus = GDdetach(gctpGridHandle) ) == -1 )
    {
      fprintf (stderr, "Error detatching from grid.\n");
      return InCPpADPPFail;
    }
  }
  if (GDclose(fid) == -1)
  {
    fprintf (stderr, "Error closing HDF File.\n");
    return InCPpADPPFail;
  }
  return (InCPpADPPSuccess);
}
```

7.4 Combining HDF and HDF-EOS Objects

The HDF-EOS structures, swath, point, and grid, are built out of the standard HDF objects such as the Vgroup, Vdata, and SDS and thus an HDF-EOS file can contain both HDF and HDF-EOS entities.

7.4.1 Adding HDF-EOS Structures to an Existing HDF File

In this example we open an existing HDF file, *NativeHDF.hdf*, and add a swath structure. Because the swath is an HDF-EOS entity we use the HDF-EOS API routines to open, create, detach and close, ie, *SWopen*, *SWcreate*, *SWdetach*, and *SWclose.*

```
      /* Open HDF file for read-write access */
      fileID = SWopen("NativeHDF.hdf", DFACC_RDWR);
      /* Create Swath Structure */
swathID = SWcreate(fileID, "SwathStructure");
/* Detach Swath Structure */
status = SWdetach(swathID);
/* Close File */
status = SWclose(fileID);
```

7.4.2 Adding an SDS to an Existing HDF File

In this example, we open an existing HDF-EOS file, *HDFEOS.hdf*, and add an SDS. Thus we use the native HDF SDS interface routines, *SDstart*, *SDcreate*, *SDendaccess*, and *SDend*. To add a Vdata or Vgroup we would simply use the relevant HDF routines.

```
/* Open HDF file for read-write access to the SD interface */
      sdID = SDstart("HDFEOS.hdf", DFACC_RDWR);
      /* Create SDS */
sdsID = SDcreate(sdID, "SDS", DFNT_FLOAT32, rank, dims);
/* Detach SDS */
status = SDendaccess(sdsID);
/* Close File */
status = SDend(sdID);
```

7.4.3 Writing HDF-EOS Structures and Native HDF Objects Simultaneously

At times, the user might wish to manipulate both HDF and HDF-EOS objects simultaneously. The *EHidinfo* routine (*ehidinfo* for FORTRAN) returns the HDF file id and SDS interface id given the HDF-EOS file id. These ids can then be used in the standard HDF calls. Note that the file must be opened and closed using the HDF-EOS routines.

```
/* Open HDF file for read-write access */
/* Note: DFACC_CREATE access can be used to create a new file */
      hdfeosID = SWopen("HDFEOS_HDF.hdf", DFACC_RDWR);
      /* Create Swath Structure */
swathID = SWcreate(hdfeosID, "SwathStructure");
```

```
/* Get SDS interface ID HDF from HDF-EOS file id */
status = EHidinfo(hdfeosID, &hdfID, &sdsInterfaceID);
sdsID = SDcreate(sdsInterfaceID, “SDS”, DFNT_FLOAT32, rank, dims);
/* Detach SDS */
status = SDendaccess(sdsID);
/* Detach Swath Structure */
status = SWdetach(swathID);
/* Close File */
status = SWclose(hdfeosID);
```

This page intentionally left blank.

8. Examples of SDP Toolkit Usage

In Section 8, examples of usage of HDF-EOS were presented, which did not use the SDP Toolkit. In this section , we give simple examples of usage of HDF-EOS in conjunction with the Toolkit..

In the ECS production System, users will have access to disk files through the Toolkit. That is, physical file handles required by open statements, will be accessed by toolkit calls. Error messages will be written to Status Message Files for processing by the production system. The user will not write a log file directly to disk, except through the Toolkit interface.

We assume that the user has installed the SDP Toolkit.

The reader is directed to The SDP Toolkit Users Guide for the ECS Project, Sections 5 and 6 and Appendices C and B, for further detailed references.

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 undergone thorough inspections and checks for ECS software standard compliance.

8.1 Opening a File

The following code fragments are simple examples of how the science software might use the SDP Toolkit logical–to–physical filename translation function in conjunction with an HDF-EOS open function.

The examples assume the following exists in the Process Control File (PCF):

? PRODUCT OUTPUT FILES

399|test10.hdf|/fire2/toma/data||||3

399|test9.hdf|/fire2/toma/data||||2

399|test8.hdf|/fire2/toma/data||||1

8.1.1 A C Example of a File Open Using the SDP Toolkit

```
#include <PGS_PC.h>
#include <PGS_SMF.h>
#include <hdf.h>
#include <dfi.h>
#define HDF_FILE 399
PGSt_integer version;
char physical_filename[PGSd_PC_FILE_PATH_MAX];
PGSt_SMF_status returnStatus;
int32 status;
/*
```

```
Begin example
*/
version = 1;
returnStatus = PGS_PC_GetReference
      ( HDF_FILE, &version, physical_filename );
/*
Variable physical_filename now contains the string
"/fire2/toma/data/test10.hdf"
Variable version now contains the value 2, i.e., the number
of versions left in order, below this version in the PC file
*/
/*
Open the HDF file
*/
status = SWopen(physical_filename, DFACC_CREATE);
```

8.1.2 A FORTRAN Example of a File Open Using the SDP Toolkit

```
implicit none
INCLUDE           'PGS_SMF.f'
INCLUDE           'PGS_PC.f'
INCLUDE           'PGS_PC_9.f'
INCLUDE           'hdf.f'
INCLUDE           'dfi.f'
INTEGER           HDF_INFILE
PARAMETER         (HDF_INFILE=399)
CHARACTER*(*)     physicalfilename
INTEGER           pgs_pc_getreference
INTEGER           version
INTEGER           returnstatus
INTEGER           status
INTEGER           ndds
C
C Begin example
C
  version = 1
       returnstatus = pgs_pc_getreference
.     ( HDF_INFILE, version, physicalfilename )
C
C Variable physicalfilename now contains the string
C "/fire2/toma/data/test10.hdf"
C Variable version now contains the value 2, i.e., the number
C of versions left in order below this version in the PC file
C
C Open the HDF file
C
      status = swopen(physicalfilename,DFACC_CREATE)
```

Notes:

In order for this tool to work properly in the SCF environment, a Process Control File (PCF) must first be created by the science software developer. This file is part of the mechanism that maps the logical file identifiers in the science code to physical filenames. (This mapping will be performed by the scheduling subsystem in the DAAC environment.) See Section 4.2.2, SDP Toolkit Users Guide for the ECS Project "File Management," for further discussion. UNIX environment variable $PGS_PC_INFO_FILE must point to the this file. In general, the PCF created by the user must follow the format given in Appendix C.

8.2 Status Message Example

The following example shows how a user of HDF-EOS can use SDP Toolkit status message calls to write HDF status messages into status message files. This files will be processed by the ECS production system. For more detailed information, please refer to The SDP Toolkit Users Guide for the ECS Project, Section 6.2.2 and Appendix B.

A. The examples in this Section assume the following exists in the Process Control File (PCF):

```
#  Process Control File: PCF.mypcf
#  Remember to reset the environment variable PGS_PC_INFO_FILE
#  to point to the instance of your PCF file
?  SYSTEM RUNTIME PARAMETERS
# ----------------------------------------------------------------------
# Production Run ID - unique production instance identifier
1
# Software ID - unique software configuration identifier
# ----------------------------------------------------------------------
1
?  PRODUCT INPUT FILES
# 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
?  PRODUCT OUTPUT FILES
# Next line is the default location for PRODUCT OUTPUT FILES
! ~/runtime
# Output files for metadata test program.
# ---------------------------------------------------------------------
201|test.hdf|||||1
?  SUPPORT INPUT FILES
# Next line is the default location for SUPPORT INPUT FILES
! ~/runtime
?  SUPPORT OUTPUT FILES
# 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.
# ----------------------------------------------------------------------
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.
# ----------------------------------------------------------------------
10111|ShmMem|||||1
? USER DEFINED RUNTIME PARAMTERS
? INTERMEDIATE INPUT
# Next line is default location for INTERMEDIATE INPUT FILES
! ~/runtime
? INTERMEDIATE OUTPUT
# Next line is default location for INTERMEDIATE OUTPUT FILES
!
~/runtime
? TEMPORARY I/O
# Next line is default location for TEMPORARY I/O FILES
! ~/runtime
?  END
```

B. The examples in this Section assume the following exists in the Status Message File (SMF):

```
ERBE,SCANNER,99
815104,SCANNER_F_OPEN_FILE,NULL,FATAL_ERROR...opening file
814081,SCANNER_W_WRITE_FAILURE,NULL,WARNING: Problem writing data attribute
811522,SCANNER_S_SUCCESS,NULL,SUCCESS: No errors
```

8.2.1 A C Example of SDP Toolkit Status Message Handling

```
#include "PGS_SMF.h"
#include "PGS_PC.h"
#include "hdf.h"
#include "dfi.h"
#include "PGS_SCANNER_99.h"
#define HDF_FILE 201
main()
{
 PGSt_integer version;
 PGSt_SMF_status returnStatus;
 int32   errId,swfid,SWid;
 int32 array[11]={1,2,3,4,5,6,7,8,9,10,11};
 char message[PGS_SMF_MAX_MSGBUF_SIZE];
 char physical_filename[PGSd_PC_FILE_PATH_MAX];
 char error_filename[PGSd_PC_FILE_PATH_MAX];
/* Associate logical with physical filename */
 version =1;
 returnStatus = PGS_PC_GetReference(HDF_FILE, &version, physical_filename);
/* Create the HDF-EOS file $PGSHOME/runtime/test.hdf */
 swfid = SWopen(physical_filename, DFACC_CREATE);
/* If error, send message to .LogStatus file */
 if(swfid != PGS_S_SUCCESS)
 {
  PGS_SMF_GetMsgByCode(SCANNER_F_OPEN_FILE,message);
  PGS_SMF_SetStaticMsg( SCANNER_F_OPEN_FILE,
        "SWopen" );
 }
/* Create the first swath  */
 SWid = SWcreate(swfid, "Swath1");
 returnStatus = SWwriteattr(swfid, "Orbit_params", DFNT_INT32, 11, array);
/* If error, send message to .LogStatus file */
 if(returnStatus != PGS_S_SUCCESS)
 {
  PGS_SMF_GetMsgByCode(SCANNER_W_WRITE_FAILURE,message);
  PGS_SMF_SetStaticMsg( SCANNER_W_WRITE_FAILURE,
        "SWwriteattr" );
 }
/* Detach and close swath file */
 SWdetach(SWid);
 SWclose(swfid);
}
```

8.2.2 A FORTRAN Example of SDP Toolkit Status Message Handling

```
 implicit none
 integer HDF_FILE
 include "./include/PGS_SMF.f"
 include "./include/PGS_PC.f"
 include "./include/PGS_PC_9.f"
 include "./include/PGS_MET.f"
 include "./include/PGS_MET_13.f"
 include "./include/PGS_SCANNER_99.f"
 integer version
 integer returnStatus
 integer pgs_pc_getreference
 integer pgs_smf_getmsgbycode
 integer pgs_smf_setstaticmsg
 integer swopen,DFACC_CREATE,swcreate,DFNT_INT32
 integer swwrattr,swdetach,swclose
 integer   swfid,swid
 integer array(11)/1,2,3,4,5,6,7,8,9,10,11/
 character*40 physical_filename
 character*40 message
 data HDF_FILE/201/
 data DFACC_CREATE/4/
 data DFNT_INT32/5/
C* Associate logical with physical filename
 version=1
 returnStatus = pgs_pc_getreference
 .    (HDF_FILE,version,physical_filename)
C* Create the HDF-EOS file
 swfid = swopen(physical_filename, DFACC_CREATE)
C* If error, send message to .LogStatus file
 if(swfid .ne. PGS_S_SUCCESS)then
  returnStatus=
 .    pgs_smf_getmsgbycode(SCANNER_F_OPEN_FILE,message)
  returnStatus=
 .    pgs_smf_setstaticmsg(SCANNER_F_OPEN_FILE,"swopen")
 end if
C* Create swath
 swid = swcreate(swfid,"Swath1")
 returnStatus = swwrattr
 . (swfid,"Orbit_params", DFNT_INT32,11,array)
C* If error, send message to .LogStatus file
 if(returnStatus .ne. PGS_S_SUCCESS)then
  returnStatus=
 .    pgs_smf_getmsgbycode(SCANNER_W_WRITE_FAILURE,message)
  returnStatus=
 .    pgs_smf_setstaticmsg
 .    (SCANNER_W_WRITE_FAILURE,"swwrattr")
 end if
C* Detach and close swath file
 returnStatus = swdetach(swid)
 returnStatus = swclose(swfid)
 stop
 end
```

8.3 Writing ODL Metadata into HDF-EOS

8.3.1. A C Example of Metadata Write

The following C code fragment is an example of how a user can write granule metadata (or inventory metadata) into their HDF-EOS file. The Metadata Configuration File (MCF), which the code accesses is given in Section 8.3.1.2. The output ODL file which results from running the code in 8.3.1.1. is given in Section 8.3.1.3. Details on Metadata Configuration Files and Metadata toolkit in general can be found in *The SDP Toolkit Users Guide for the ECS Project, Section 6.2.1 and Appendix J.*

8.3.1.1 C Code

```
#include "PGS_MET.h"
#include "PGS_PC.h"
#include "hdf.h"
/* Program to collect and append the meta-data to
 the swath HDF-EOS file */
#define INVENTORY 1
#define ARCHIVE 2
#define AUX 3
#define  MODIS_FILE 10200
#define  MCF_FILE 10250
#define  HDF_FILE 301
main()
{
  int32    sdId, hdf_status;
  intn i,j,k,l,m;
  char physical_filename[PGSd_PC_FILE_PATH_MAX];
  char hdfeos_filename[PGSd_PC_FILE_PATH_MAX];
  char name[25];
  char *NAval = "N/A";
  char *spacecraft = "ERBS";
  char *start_time = "1986-001T00:00:00.000";
  char *end_time =     "1986-001T23:59:59.000";
  char message[PGS_SMF_MAX_MSGBUF_SIZE];
  PGSt_MET_all_handles mdHandles; /* These represent group names present
      in the MCF*/
  PGSt_SMF_status    retVal = PGS_S_SUCCESS;
  PGSt_integer attrValI,version;
  PGSt_double attrValD;
  PGSt_SMF_status returnStatus;
  /* Associate logical reference to physical filename */
  version =1;
```

```
   returnStatus =
       PGS_PC_GetReference(HDF_FILE,&version,physical_filename);
  PGS_SMF_GetMsgByCode(returnStatus,message);
  PGS_SMF_SetStaticMsg(returnStatus,"Metadata Output
      PGS_PC_GetReference");
  retVal = PGS_MET_Init(MCF_FILE, mdHandles);
  PGS_SMF_GetMsgByCode(retVal,message);
  PGS_SMF_SetStaticMsg(retVal,"PGS_MET_Init");
  if(retVal == PGS_S_SUCCESS)
  {
    retVal = PGS_MET_SetAttr(mdHandles[INVENTORY],
      "RANGEBEGINNINGDATETIME",
      &start_time);
    PGS_SMF_GetMsgByCode(retVal,message);
    PGS_SMF_SetStaticMsg(retVal,"RANGEBEGINNINGDATETIME");
    retVal = PGS_MET_SetAttr(mdHandles[INVENTORY],
      "RANGEENDINGDATETIME",
      &end_time);
    PGS_SMF_GetMsgByCode(retVal,message);
    PGS_SMF_SetStaticMsg(retVal,"RANGEENDINGDATETIME");
    version = 3.0;
    attrValI =      version;
    retVal = PGS_MET_SetAttr(mdHandles[AUX],
      "VERSIONID",
      &attrValI);
    PGS_SMF_GetMsgByCode(retVal,message);
    PGS_SMF_SetStaticMsg(retVal,"VERSION_ID");
    attrValI = 0;
    retVal = PGS_MET_SetAttr(mdHandles[ARCHIVE],
      "BROWSESIZE",
      &attrValI);
    PGS_SMF_GetMsgByCode(retVal,message);
    PGS_SMF_SetStaticMsg(retVal,"BROWSESIZE");
  }
  else
  {
    exit(1);
  }
/* Write the meta data to the HDF-EOS file */
  sdId = SDstart( physical_filename, DFACC_RDWR);
  retVal = PGS_MET_Write(mdHandles[INVENTORY],
    "coremetadata.0", sdId);
  if(retVal != PGS_S_SUCCESS)
  PGS_SMF_GetMsgByCode(retVal,message);
  PGS_SMF_SetStaticMsg(retVal,"Inventory WRITE");
```

```
   retVal = PGS_MET_Write(mdHandles[ARCHIVE],
    "coremetadata.1",sdId);
  if(retVal != PGS_S_SUCCESS)
  PGS_SMF_GetMsgByCode(retVal,message);
  PGS_SMF_SetStaticMsg(retVal,"Archive WRITE");
  retVal = PGS_MET_Write(mdHandles[AUX],
    "productspecific.0", sdId);
  if(retVal != PGS_S_SUCCESS)
  PGS_SMF_GetMsgByCode(retVal,message);
  PGS_SMF_SetStaticMsg(retVal,"Product Specific WRITE");
  SDend(sdId);
  PGS_MET_Remove();
}


#include "PGS_MET.h"
#include "PGS_PC.h"
#include "hdf.h"
#include "HdfEosDef.h"
#include "InPpCommonExtSciData.h"

/*
 William E. Smith
 Applied Research Corporation

 April 22, 1997
*/

/* Program to collect and append the meta-data to
 the swath HDF-EOS file */

#define INVENTORY 1
#define ARCHIVE 2

PGSt_MET_all_handles mdHandles; /* MCF group names */

int metawrite( FILE *logout, char *hdf_filename, char *dumpfilename)
{
  void start_end(char dumpline[],char start_time[],char end_time[]);
  void parse_param(intn *ip, intn *kp, intn *lp, intn *flagp, FILE *logout,
    char *fieldlist, int32 flds, char one[], char ten[]);
  void sys_time(char *create_time);

  long GetFilesize (char *filename);

  int32    sdId, hdf_status;
  int32    strbufsize, swfid, SWid, nflds;
  int32    geoflds, datflds;
  int32    *rank, *ntype;
```

```
   intn j,m;
  intn i,k,l,flag=0;
  intn *ip,*kp,*lp,*flagp;

  char *inputp;
  char spacecraft[80];
  char start_time[25]="19";
  char end_time[25]="19";
  char *fieldlist;
  char *parname;
  char dumpline[80];
  char ten[]={"0"};
  char one[]={"0"};
  char message[PGS_SMF_MAX_MSGBUF_SIZE];
  char mnemonic [PGS_SMF_MAX_MNEMONIC_SIZE];
  char file_des[2000];

  PGSt_SMF_status    retVal = PGS_S_SUCCESS;
  PGSt_SMF_code      code;
  PGSt_integer attrValI;
  PGSt_double attrValD;
  PGSt_SMF_status returnStatus;

  FILE *in;

/* Assign pointer */
  ip = &i;
  kp = &k;
  lp = &l;
  flagp = &flag;

/* Report informational to the log file */
  fprintf(logout,"Informational, metawrite(), \"Generating
      metadata.\"\n");

/* If filename missing or there is a problem, print message and exit */
  if((in = fopen(dumpfilename,"r")) == NULL)
  {
    fprintf(logout,"Error, metawrite(), \"Cannot open %s.\"\n",
      dumpfilename);
    return(1);
  }

  /* Read file header file to extract spacecraft name */
  /* and the start and end time of the granule */
  for(m=0;m<13;++m)
  {
    fscanf(in,"%s",dumpline);
    if(feof(in) !=0)
      fprintf(logout,"EOF for the dump file\n");

    /* Extract start and end time from file header */
```

```
    if(m == 4)
      start_end(dumpline,start_time,end_time);

    /* Extract spacecraft name */
    if(m == 10)
      strcpy(spacecraft,dumpline);

  }

  fclose(in);

  retVal = PGS_MET_Init(InDPpPCFLogIDMCF, mdHandles);
  if ( retVal != PGS_S_SUCCESS )
  {
    PGS_SMF_GetMsg( &code, mnemonic, message );
    fprintf(logout,"Error, metawrite(), \"Initialization Status =
      %s\"\n",message);
  }

  if(retVal == PGS_S_SUCCESS)
  {
    attrValD = 360.;
    retVal = PGS_MET_SetAttr(mdHandles[INVENTORY],
      "EASTBOUNDINGCOORDINATE",
      &attrValD);
    if ( retVal != PGS_S_SUCCESS )
    {
      PGS_SMF_GetMsg( &code, mnemonic, message );
      fprintf(logout,"Error, metawrite(), \"Set Attr Status for
       EASTBOUNDINGCOORDINATE = %s\"\n", message);
    }

    attrValD = 0.;
    retVal = PGS_MET_SetAttr(mdHandles[INVENTORY],
      "WESTBOUNDINGCOORDINATE",
      &attrValD);
    if ( retVal != PGS_S_SUCCESS )
    {
      PGS_SMF_GetMsg( &code, mnemonic, message );
      fprintf(logout,"Error, metawrite(), \"Set Attr Status for
       WESTBOUNDINGCOORDINATE = %s\"\n", message);
    }

    attrValD = -90.;
    if(spacecraft[0] != 'N')
      attrValD = -57.;

    retVal = PGS_MET_SetAttr(mdHandles[INVENTORY],
      "SOUTHBOUNDINGCOORDINATE",
      &attrValD);
    if ( retVal != PGS_S_SUCCESS )
    {
```

```
    PGS_SMF_GetMsg( &code, mnemonic, message );
   fprintf(logout,"Error, metawrite(), \"Set Attr Status for
    SOUTHBOUNDINGCOORDINATE = %s\"\n", message);
  }

  attrValD = 90.;
  if(spacecraft[0] != 'N')
   attrValD = 57.;

  retVal = PGS_MET_SetAttr(mdHandles[INVENTORY],
   "NORTHBOUNDINGCOORDINATE",
   &attrValD);
  if ( retVal != PGS_S_SUCCESS )
  {
   PGS_SMF_GetMsg( &code, mnemonic, message );
   fprintf(logout,"Error, metawrite(), \"Set Attr Status for
    NORTHBOUNDINGCOORDINATE = %s\"\n", message);
  }

  inputp = &start_time[0];
  retVal = PGS_MET_SetAttr(mdHandles[INVENTORY],
    "RANGEBEGINNINGDATETIME",
    &inputp);
  if ( retVal != PGS_S_SUCCESS )
  {
   PGS_SMF_GetMsg( &code, mnemonic, message );
   fprintf(logout,"Error, metawrite(), \"Set Attr Status for
    RANGEBEGINNINGDATETIME = %s\"\n", message);
  }

  inputp = &end_time[0];
  retVal = PGS_MET_SetAttr(mdHandles[INVENTORY],
    "RANGEENDINGDATETIME",
    &inputp);
  if ( retVal != PGS_S_SUCCESS )
  {
   PGS_SMF_GetMsg( &code, mnemonic, message );
   fprintf(logout,"Error, metawrite(), \"Set Attr Status for
    RANGEENDINGDATETIME = %s\"\n", message);
  }

  inputp = &spacecraft[0];
  retVal = PGS_MET_SetAttr(mdHandles[ARCHIVE],
    "SPACECRAFT",
    &inputp);
  if ( retVal != PGS_S_SUCCESS )
  {
   PGS_SMF_GetMsg( &code, mnemonic, message );
   fprintf(logout,"Error, metawrite(), \"Set Attr Status for
    SPACECRAFT = %s\"\n", message);
  }
```

```
    inputp = &hdf_filename[0];
    retVal = PGS_MET_SetAttr(mdHandles[INVENTORY],
      "GRANULEPOINTER",
      &inputp );
    if ( retVal != PGS_S_SUCCESS )
    {
      PGS_SMF_GetMsg( &code, mnemonic, message );
      fprintf(logout,"Error, metawrite(), \"Set Attr Status for
       GRANULEPOINTER = %s\"\n", message);
    }

/* This block of code extracts the field parameter names from the
 structural metadata using the API function calls.   The parameter
 names are then written to the collection level metadata. */

  /*
   * Open the Swath File for read only access
   */

    swfid = SWopen(hdf_filename, DFACC_READ);

    if (swfid != -1)
    {

    /* Attach the swath */

      SWid = SWattach(swfid,"ERBE_S8" );

      if (SWid != -1 )
      {

/* Obtain the geolocation field list. */
        nflds = SWnentries(SWid, HDFE_NENTGFLD, &strbufsize);
        geoflds = nflds;
        rank = (int32 *) calloc(nflds, 4);
        ntype = (int32 *) calloc(nflds, 4);
        fieldlist = (char *) calloc(strbufsize + 1, 1);
        nflds = SWinqgeofields(SWid, fieldlist, rank, ntype);

/* Parse the geolocation field list names */
        l=0;
        k=0;
        i=0;

        parse_param(ip, kp, lp, flagp, logout, fieldlist, geoflds,
one, ten);

        free(fieldlist);
        free(rank);
        free(ntype);

/* Obtain the data field list. */
```

```
        nflds = SWnentries(SWid, HDFE_NENTDFLD, &strbufsize);
        datflds = nflds;
        rank = (int32 *) calloc(nflds, 4);
        ntype = (int32 *) calloc(nflds, 4);
        fieldlist = (char *) calloc(strbufsize + 1, 1);
        nflds = SWinqdatafields(SWid, fieldlist, rank, ntype);

/* Parse the data field list names */
        k=0;
        i=0;

        parse_param(ip, kp, lp, flagp, logout, fieldlist, datflds,
             one, ten);

        free(fieldlist);
        free(rank);
        free(ntype);

      }
      else
      {
        fprintf(logout,"Unable to open SWATH\n");
        return(1);
      }
    }
    hdf_status = SWdetach(SWid);
    hdf_status = SWclose(swfid);
/* End of parameter name processing */

  }
  else
  {
    return(1);
  }

/* Write the meta data to the HDF-EOS file */
  sdId = SDstart( hdf_filename, DFACC_RDWR);
  if(sdId < 0)
  {
    fprintf(logout,"Error, metawrite(), \"Filename %s Not
      Found.\"\n",hdf_filename);
    return(1);
  }

  retVal = PGS_MET_Write(mdHandles[INVENTORY],
    "coremetadata.0", sdId);
  if(retVal != PGS_S_SUCCESS)
  {
    PGS_SMF_GetMsg( &code, mnemonic, message );
    fprintf(logout,"Error, metawrite(), \"Inventory write status =
      %s\"\n", message);
    SDend(sdId);
```

```
    PGS_MET_Remove();
    return(1);
  }

  retVal = PGS_MET_Write(mdHandles[ARCHIVE],
    "archivedata.0",sdId);
  if(retVal != PGS_S_SUCCESS)
  {
    PGS_SMF_GetMsg( &code, mnemonic, message );
    fprintf(logout,"Error, metawrite(), \"Archive write status =
      %s\"\n", message);
    SDend(sdId);
    PGS_MET_Remove();
    return(1);
  }

  SDend(sdId);

  PGS_MET_Remove();

  fprintf(logout,"Informational, metawrite(), \"Meta data write
      complete.\"\n");

  fclose(logout);

/* Add file desciption to HDFEOS file */

/* If filename missing or there is a problem, print message and exit */
  if((in = fopen(dumpfilename,"r")) == NULL)
  {
    fprintf(logout,"Error, metawrite(), \"Cannot open %s.\"\n",
      dumpfilename);
    return(1);
  }

  strcpy(file_des,"\0");
  for(m=0;m<28;++m)
  {
    fgets(dumpline,80,in);
    strcat(file_des,dumpline);
/*      strcat(file_des,"\n"); */
  }
  strcat(file_des,"\0");

  sdId = Hopen(hdf_filename, DFACC_RDWR, 0);

  DFANaddfds(sdId, file_des, 2000);

  Hclose(sdId);

  fclose(in);
```

```
   return(0);
}

/* Function to determine Julian day number */
      int julian(int mon, int day, int year)
{
  int monref[12]={0,31,28,31,30,31,30,31,31,30,31,30};
  int jul,i;

  jul =0;

  if(year%4 == 0)
    monref[2] = 29;
    if(year%100 == 0)
      if(year%400 != 0)
        monref[2]=28;

  for(i=0;i<mon;++i)
   jul += monref[i];

  jul += day;

  return(jul);
}


      void start_end(char dumpline[],char start_time[],char end_time[])
{
/* Extract start and end time from file header */
  int julian(int mon, int day, int year);

  char time[25];
  char julday[4],mon[3],day[3],year[5];
  intn j1,j2,j3,jul,moni,dayi,yeari;
  char times[15]="T00:00:00.000";
  char timee[15]="T23:59:00.000";

  strcpy(time,dumpline);
  strncat(start_time,time,2);
  strcat(start_time,"-");
  year[0] = '1';
  year[1] = '9';
  year[2] = time[0];
  year[3] = time[1];
  year[4] = '\0';
  mon[0]=time[3];
  mon[1]=time[4];
  mon[2]='\0';
  day[0]=time[6];
  day[1]=time[7];
  day[2]='\0';
  moni = atoi(mon);
```

```
   dayi = atoi(day);
  yeari = atoi(year);

/* Function to determine Julian day number */
  jul = julian(moni, dayi, yeari);

  j1 = jul/100;
  jul = jul-j1*100;
  j2 = jul/10;
  jul = jul-j2*10;
  j3 = jul;
  julday[0]='0'+j1;
  julday[1]='0'+j2;
  julday[2]='0'+j3;
  julday[3]='\0';
  strcat(start_time,julday);
  strcpy(end_time,start_time);
  strcat(start_time,times);
  strcat(end_time,timee);
}


      void parse_param(intn *ip, intn *kp, intn *lp, intn *flagp, FILE
      *logout, char *fieldlist, int32 flds, char one[], char ten[])
{
  intn i,j,k,l,flag;
  char name[25];
  char *inputp;
  char *parname;
  char parameternamе[80];
  char paramt[]={"PARAMETERNAME"};

  char message[PGS_SMF_MAX_MSGBUF_SIZE];
  char mnemonic [PGS_SMF_MAX_MNEMONIC_SIZE];

  PGSt_SMF_status    retVal = PGS_S_SUCCESS;
  PGSt_SMF_code      code;

  i=*ip;
  k=*kp;
  l=*lp;
  flag=*flagp;

  for(j=0;j<flds;++j)
  {
    while(fieldlist[k] != ',' )
    {
      name[i] = fieldlist[k];
      ++i;
      ++k;
      if(fieldlist[k] == NULL)
        break;
```

```
    }

    name[i] = NULL;
    inputp = &name[0];
    strcpy(parametername,paramt);
    strcat(parametername,".");
    one[0]=one[0]+1;
    ++l;
    if(l>9)
    {
      l=0;
      one[0]='0';
      ten[0]=ten[0]+1;
      flag=1;
    }

    if(flag == 1)
      strcat(parametername,ten);
    strcat(parametername,one);

/* Set the field name paramter attribute */
    parname = &parametername[0];

    retVal = PGS_MET_SetAttr(mdHandles[ARCHIVE],
      parname, &inputp);

    if ( retVal != PGS_S_SUCCESS )
    {
      PGS_SMF_GetMsg( &code, mnemonic, message );
      fprintf(logout,"Error, metawrite(), \"Set Attr Status for %s =
       %s\"\n", parname, message);
    }

    ++k;
    if(fieldlist[k] == NULL)
      break;
    i=0;

  }
  *ip=i;
  *kp=k;
  *lp=l;
  *flagp=flag;
}
```

8.3.1.2 The Metadata Configuration File (MCF) for Code in Section 8.3.1.1.

```
GROUP = INVENTORYMETADATA

GROUPTYPE = MASTERGROUP

/* IMPORTANT - any text comment in the MCF MUST be enclosed */
/* per line with /* and */.  Unlike C where a block of text may */
/* be enclosed */


OBJECT  = ShortName
      Data_Location  = "MCF"
      Value = "ER8PAT"
      TYPE = "STRING"
      NUM_VAL = 1
      Mandatory  = "TRUE"
END_OBJECT = ShortName

OBJECT = SizeMBECSDataGranule
      Data_Location  = "DSS"
      TYPE = "DOUBLE"
      NUM_VAL = 1
      Mandatory  = "TRUE"
END_OBJECT = SizeMBECSDataGranule

OBJECT = ProductionDateTime
    Data_Location       = "TK"
    Mandatory           = "TRUE"
    Type = "STRING"
    Num_val = 1
END_OBJECT = ProductionDateTime

GROUP = BoundingRectangle

      OBJECT = EastBoundingCoordinate
            Data_Location = "PGE"
            TYPE = "DOUBLE"
            NUM_VAL = 1
            Mandatory = "TRUE"
      END_OBJECT = EastBoundingCoordinate

      OBJECT = WestBoundingCoordinate
            Data_Location     = "PGE"
            TYPE = "DOUBLE"
            NUM_VAL =  1
            Mandatory = "TRUE"
      END_OBJECT = WestBoundingCoordinate

      OBJECT = NorthBoundingCoordinate
            Data_Location = "PGE"
            TYPE = "DOUBLE"
```

```
            NUM_VAL = 1
            Mandatory = "TRUE"
      END_OBJECT = NorthBoundingCoordinate

      OBJECT = SouthBoundingCoordinate
            Data_Location = "PGE"
            TYPE = "DOUBLE"
            NUM_VAL = 1
            Mandatory  = "TRUE"
      END_OBJECT = SouthBoundingCoordinate

END_GROUP = BoundingRectangle

/* range time */

GROUP = RangeDateTime

      OBJECT = RangeBeginningDateTime
            Data_Location = "PGE"
            TYPE = "DATETIME"
            NUM_VAL = 1
            Mandatory = FALSE
      END_OBJECT = RangeBeginningDateTime

      OBJECT = RangeEndingDateTime
            Data_Location = "PGE"
            TYPE = "DATETIME"
            NUM_VAL = 1
            Mandatory = FALSE
      END_OBJECT = RangeEndingDateTime

END_GROUP = RangeDateTime

OBJECT = GranulePointer
      Data_Location = "PGE"
    TYPE = "STRING"
    NUM_VAL = 1
    Mandatory = "TRUE"
END_OBJECT = GranulePointer

OBJECT = ReprocessingActual
      Data_Location  = "MCF"
      Value = "Reprocessed Once"
      TYPE = "STRING"
      NUM_VAL = 1
      Mandatory = "TRUE"
END_OBJECT = ReprocessingActual

OBJECT = ReprocessingPlanned
      Data_Location  = "MCF"
      Value = "No Further Update Anticipated"
      TYPE = "STRING"
```

```
      NUM_VAL = 1
      Mandatory = "TRUE"
END_OBJECT = ReprocessingPlanned

/* these sensor characteristics change relatively rapidly and */
/* need to be store in a database.  The values vary according */
/* to data in the telemetry stream and are processed */
/* automatically.  They are set up in the same way as */
/* Parameter values.   SensorCharacteristicValue has */
/* type SensorCharacteristictype. This group/object construct */
/* allows for 2-dimensional  values sets; i.e. where each */
/* SensorCharacteristicValue has many values per granule.  */


GROUP = SensorCharacteristic

OBJECT = SensorCharacteristicContainer

      CLASS = "1"
      Mandatory = "TRUE"
      Data_Location = "NONE"

      OBJECT = SensorCharacteristicName
            Data_Location = "MCF"
            CLASS = "1"
            TYPE = "STRING"
            NUM_VAL = 1
            Mandatory = "TRUE"
       Value = "Earth Radiation Budget Experiment"
      END_OBJECT = SensorCharacteristicName

      OBJECT = SensorShortName
            Data_Location = "MCF"
            CLASS = "1"
            TYPE = "STRING"
            NUM_VAL = 1
            Mandatory = "TRUE"
       Value = "ERBE-Scanner"
      END_OBJECT = SensorShortName

      OBJECT = InstrumentShortName
            Data_Location = "MCF"
            CLASS = "1"
            TYPE = "STRING"
            NUM_VAL = 1
            Mandatory = "TRUE"
       Value = "ERBE"
      END_OBJECT = InstrumentShortName

/* Instrument and Sensor names needed to allow the */
/* database to locate the correct charactreristic name */
```

```
END_OBJECT = SensorCharacteristicContainer

END_GROUP = SensorCharacteristic


/* end of master group */

END_GROUP = INVENTORYMETADATA


GROUP = ARCHIVEDMETADATA

GROUPTYPE = MASTERGROUP

/* this master group may contain any core attributes group */
/* plus product specific attributes.  Both may be single */
/* attributes or repeating.  In the latter case, arrays plus the */
/* Group, Object and CLASS = "1" construct shown above should*/
/*  be used. */

/* This group should be written using MET_WRITE. */

/* These attributes are NOT parsed into the inventory and */
/* are therefore NOT SEARCHABLE */

/* the following are core attributes not required to be searched */
/*  , but which may be mandatory or just useful - according to */
/* choices made in appendix B. */

/* these attribute needed in browse granules */

/* Shortname +                                  same as in granule */
/* BrowseDescription + */
/* BrowseSize + */
/* SpatialDomainContainer +                        same as in granule */

/* [RangeDateTime | SingleDateTime] +       same as in granule */

OBJECT  = ShortName
    Data_Location      = "MCF"
    Value = "ERBE_S8_PAT"
    TYPE = "STRING"
    NUM_VAL = 1
    Mandatory      = "TRUE"
END_OBJECT = ShortName

GROUP = Browse

OBJECT =  BrowseSize
      Data_Location  = "MCF"
      TYPE = "DOUBLE"
      NUM_VAL = 1
```

```
      Mandatory = FALSE
    Value = 0.
END_OBJECT =  BrowseSize

OBJECT =  BrowseDescription
      Data_Location  = "MCF"
      Value = "Data distribution map"
      NUM_VAL = 1
      TYPE = "STRING"
      Mandatory = FALSE
END_OBJECT =  BrowseDescription

END_GROUP = Browse

GROUP = SensorCharacteristic

OBJECT = SensorCharacteristicContainer

      CLASS = "1"
      Data_Location = "NONE"
      Mandatory = "FALSE"

      OBJECT = SensorCharacteristicName
            Data_Location = "MCF"
            CLASS = "1"
            TYPE = "STRING"
            NUM_VAL = 1
            Mandatory = "FALSE"
       Value = "Earth Radiation Budget Experiment"
      END_OBJECT = SensorCharacteristicName

      OBJECT = SensorShortName
            Data_Location = "MCF"
            CLASS = "1"
            TYPE = "STRING"
            NUM_VAL = 1
            Mandatory =  "FALSE"
       Value = "ERBE-Scanner"
      END_OBJECT = SensorShortName

      OBJECT = InstrumentShortName
            Data_Location = "MCF"
            CLASS = "1"
            TYPE = "STRING"
            NUM_VAL = 1
            Mandatory =  "FALSE"
       Value = "ERBE"
      END_OBJECT = InstrumentShortName

END_OBJECT = SensorCharacteristicContainer

END_GROUP = SensorCharacteristic
```

```
OBJECT = SensorCharacteristicUnit
        Data_Location = "MCF"
        TYPE = "STRING"
        NUM_VAL = 1
        Mandatory =           "FALSE"
        Value = "Watts/m**2/sr"
END_OBJECT = SensorCharacteristicUnit

OBJECT = LongName
      Data_Location  = "MCF"
      Mandatory  = "TRUE"
    Type = "STRING"
    Value = "ERBE S-8 PROCESSED ARCHIVAL TAPE"
    Num_val = 1
END_OBJECT = LongName

OBJECT = VersionID
    Data_Location       = "MCF"
    Mandatory           = "TRUE"
    Type = "INTEGER"
    Num_val = 1
    Value = 1
END_OBJECT = VersionID

OBJECT = ProducerAgency
     Data_Location        = "MCF"
     Mandatory            = "TRUE"
     Value                = "NASA"
    Type = "STRING"
    Num_val = 1
END_OBJECT = ProducerAgency

OBJECT = ProducerInstitution
     Data_Location        = "MCF"
     Mandatory            = "TRUE"
     Value                = "LaRC"
    Type = "STRING"
    Num_val = 1
END_OBJECT = ProducerInstitution

OBJECT = ProjectID
     Data_Location        = "MCF"
     Mandatory            = "TRUE"
     Value                = "ERBE"
    Type = "STRING"
    Num_val = 1
END_OBJECT = ProjectID

OBJECT = DataFormatType
     Data_Location        = "MCF"
     Mandatory            = "TRUE"
```

```
     Value                = "HDF-EOS"
    Type = "STRING"
    Num_val = 1
END_OBJECT = DataFormatType

OBJECT = Spacecraft
    Data_Location        = "PGE"
    Mandatory            = "TRUE"
    Type = "STRING"
    Num_val = 1
END_OBJECT = Spacecraft

OBJECT = DataType
    Data_Location        = "MCF"
    Value = "Level 2"
    Mandatory            = "TRUE"
    Type = "STRING"
    Num_val = 1
END_OBJECT = DataType

OBJECT = CollectionDescription
    Data_Location        = "MCF"
    Value = "Scanner radiometric and geolocation data."
    Mandatory            = "TRUE"
    Type = "STRING"
    Num_val = 1
END_OBJECT = CollectionDescription

OBJECT = TemporalKeyword
    Data_Location        = "MCF"
    Value = "Daily"
    Mandatory            = "TRUE"
    Type = "STRING"
    Num_val = 1
END_OBJECT = TemporalKeyword

OBJECT = TemporalRangeType
    Data_Location        = "MCF"
    Value = "CONTINUOUS RANGE"
    Mandatory            = "TRUE"
    Type = "STRING"
    Num_val = 1
END_OBJECT = TemporalRangeType

OBJECT = ArchiveCenter
    Data_Location        = "MCF"
    Value = "LaRC"
    Mandatory            = "TRUE"
    Type = "STRING"
    Num_val = 1
END_OBJECT = ArchiveCenter
```

```
OBJECT = ContactFirstName
    Data_Location        = "MCF"
    Value = "James"
    Mandatory            = "TRUE"
   Type = "STRING"
   Num_val = 1
END_OBJECT = ContactFirstName

OBJECT = ContactLastName
    Data_Location        = "MCF"
    Value = "Kibler"
    Mandatory            = "TRUE"
   Type = "STRING"
   Num_val = 1
END_OBJECT = ContactLastName

OBJECT = ContactJobPosition
    Data_Location        = "MCF"
    Value = "Project Manager"
    Mandatory            = "TRUE"
   Type = "STRING"
   Num_val = 1
END_OBJECT = ContactJobPosition

OBJECT = ContactInstructions
    Data_Location        = "MCF"
    Value = "Email: j.f.kibler@larc.nasa.gov       Phone: (804) 864-5386"
    Mandatory            = "TRUE"
   Type = "STRING"
   Num_val = 1
END_OBJECT = ContactInstructions

OBJECT = MaintenanceandUpdateFrequency
    Data_Location        = "MCF"
    Value = "None Planned"
    Mandatory            = "TRUE"
   Type = "STRING"
   Num_val = 1
END_OBJECT = MaintenanceandUpdateFrequency

OBJECT = DisciplineKeyword
    Data_Location        = "MCF"
    Value = "Earth Science"
    Mandatory            = "TRUE"
   Type = "STRING"
   Num_val = 1
END_OBJECT = DisciplineKeyword

OBJECT = TopicKeyword
    Data_Location        = "MCF"
    Value = "ATMOSPHERIC SCIENCE"
    Mandatory            = "TRUE"
```

```
    Type = "STRING"
   Num_val = 1
END_OBJECT = TopicKeyword

OBJECT = Document
    Data_Location        = "MCF"
    Value = "http://eosdis.larc.nasa.gov/HPDOCS/proj_addl_info.html"
    Mandatory            = "TRUE"
   Type = "STRING"
   Num_val = 1
END_OBJECT = Document


GROUP = PARAMETER

OBJECT = PARAMETERCONTAINER

  DATA_LOCATION = "NONE"
  CLASS = "M"
  Mandatory = FALSE

  OBJECT = PARAMETERNAME
    Data_Location= "PGE"
    CLASS = "M"
    TYPE = "STRING"
    NUM_VAL = 1
    Mandatory = FALSE
  END_OBJECT = PARAMETERNAME

END_OBJECT = PARAMETERCONTAINER

END_GROUP = PARAMETER

END_GROUP = ARCHIVEDMETADATA

END
```

8.3.1.3 The ODL Output File Which Results from Running Code in Section 8.3.1.1

```
GROUP                   = INVENTORYMETADATA
 GROUPTYPE              = MASTERGROUP

 /* IMPORTANT - any text comment in the MCF MUST be enclosed */
 /* per line with /* and */.   Unlike C where a block of text may */
 /* be enclosed */

 OBJECT                   = SHORTNAME
  VALUE                   = "ER8PAT"
  NUM_VAL                 = 1
 END_OBJECT               = SHORTNAME

 OBJECT                   = PRODUCTIONDATETIME
  NUM_VAL                 = 1
  VALUE                   = "1997-04-23T19:29:27.000Z"
 END_OBJECT               = PRODUCTIONDATETIME

 GROUP                    = BOUNDINGRECTANGLE

  OBJECT                    = EASTBOUNDINGCOORDINATE
   NUM_VAL                  = 1
   VALUE                    = 360.000000
  END_OBJECT                = EASTBOUNDINGCOORDINATE

  OBJECT                    = WESTBOUNDINGCOORDINATE
   NUM_VAL                  = 1
   VALUE                    = 0.000000
  END_OBJECT                = WESTBOUNDINGCOORDINATE

  OBJECT                    = NORTHBOUNDINGCOORDINATE
   NUM_VAL                  = 1
   VALUE                    = 57.000000
  END_OBJECT                = NORTHBOUNDINGCOORDINATE

  OBJECT                    = SOUTHBOUNDINGCOORDINATE
   NUM_VAL                  = 1
   VALUE                    = -57.000000
  END_OBJECT                = SOUTHBOUNDINGCOORDINATE

 END_GROUP                = BOUNDINGRECTANGLE


 /* range time */

 GROUP                    = RANGEDATETIME

  OBJECT                    = RANGEBEGINNINGDATETIME
   NUM_VAL                  = 1
   VALUE                    = "1986-001T00:00:00.000"
  END_OBJECT                = RANGEBEGINNINGDATETIME
```

```
  OBJECT                    = RANGEBEGINNINGDATE
   NUM_VAL                  = 1
   VALUE                    = "1986-01-01"
  END_OBJECT                = RANGEBEGINNINGDATE

  OBJECT                    = RANGEBEGINNINGTIME
   NUM_VAL                  = 1
   VALUE                    = "00:00:00.000000"
  END_OBJECT                = RANGEBEGINNINGTIME

  OBJECT                    = RANGEENDINGDATETIME
   NUM_VAL                  = 1
   VALUE                    = "1986-001T23:59:00.000"
  END_OBJECT                = RANGEENDINGDATETIME

  OBJECT                    = RANGEENDINGDATE
   NUM_VAL                  = 1
   VALUE                    = "1986-01-01"
  END_OBJECT                = RANGEENDINGDATE

  OBJECT                    = RANGEENDINGTIME
   NUM_VAL                  = 1
   VALUE                    = "23:59:00.000000"
  END_OBJECT                = RANGEENDINGTIME

 END_GROUP                = RANGEDATETIME

 OBJECT                   = GRANULEPOINTER
  NUM_VAL                 = 1
  VALUE                   = "./s8_860101_2.hdfeos"
 END_OBJECT               = GRANULEPOINTER

 OBJECT                   = REPROCESSINGACTUAL
  VALUE                   = "Reprocessed Once"
  NUM_VAL                 = 1
 END_OBJECT               = REPROCESSINGACTUAL

 OBJECT                   = REPROCESSINGPLANNED
  VALUE                   = "No Further Update Anticipated"
  NUM_VAL                 = 1
 END_OBJECT               = REPROCESSINGPLANNED


 /* these sensor characteristics change relatively rapidly and */
 /* need to be store in a database.   The values vary according */
 /* to data in the telemetry stream and are processed */
 /* automatically.   They are set up in the same way as */
 /* Parameter values.    SensorCharacteristicValue has */
 /* type SensorCharacteristictype. This group/object construct */
 /* allows for 2-dimensional   values sets; i.e. where each */
 /* SensorCharacteristicValue has many values per granule. */
```

```
  GROUP                    = SENSORCHARACTERISTIC

   OBJECT                    = SENSORCHARACTERISTICCONTAINER
    CLASS                    = "1"

    OBJECT                     = SENSORCHARACTERISTICNAME
     CLASS                     = "1"
     NUM_VAL                   = 1
     VALUE                     = "Earth Radiation Budget Experiment"
    END_OBJECT                 = SENSORCHARACTERISTICNAME

    OBJECT                     = SENSORSHORTNAME
     CLASS                     = "1"
     NUM_VAL                   = 1
     VALUE                     = "ERBE-Scanner"
    END_OBJECT                 = SENSORSHORTNAME

    OBJECT                     = INSTRUMENTSHORTNAME
     CLASS                     = "1"
     NUM_VAL                   = 1
     VALUE                     = "ERBE"
    END_OBJECT                 = INSTRUMENTSHORTNAME

   END_OBJECT                  = SENSORCHARACTERISTICCONTAINER

  END_GROUP                  = SENSORCHARACTERISTIC

END_GROUP                = INVENTORYMETADATA

END

GROUP                    = ARCHIVEDMETADATA
 GROUPTYPE               = MASTERGROUP

 /* this master group may contain any core attributes group */
 /* plus product specific attributes.   Both may be single */
 /* attributes or repeating.   In the latter case, arrays plus the */
 /* Group, Object and CLASS = "1" construct shown above should */
 /*   be used. */
 /* This group should be written using MET_WRITE. */
 /* These attributes are NOT parsed into the inventory and */
 /* are therefore NOT SEARCHABLE */
 /* the following are core attributes not required to be searched */
 /*   , but which may be mandatory or just useful - according to */
 /* choices made in appendix B. */
 /* these attribute needed in browse granules */
 /* Shortname + same as in granule */
 /* BrowseDescription + */
 /* BrowseSize + */
 /* SpatialDomainContainer + same as in granule */
 /* [RangeDateTime | SingleDateTime] + same as in granule */
```

```
 OBJECT                    = SHORTNAME
  VALUE                    = "ERBE_S8_PAT"
  NUM_VAL                  = 1
 END_OBJECT                = SHORTNAME

 GROUP                     = BROWSE

  OBJECT                     = BROWSESIZE
   NUM_VAL                   = 1
   VALUE                     = 0.0
  END_OBJECT                 = BROWSESIZE

  OBJECT                     = BROWSEDESCRIPTION
   VALUE                     = "Data distribution map"
   NUM_VAL                   = 1
  END_OBJECT                 = BROWSEDESCRIPTION

 END_GROUP                 = BROWSE

 GROUP                     = SENSORCHARACTERISTIC

  OBJECT                     = SENSORCHARACTERISTICCONTAINER
   CLASS                     = "1"

   OBJECT                      = SENSORCHARACTERISTICNAME
    CLASS                      = "1"
    NUM_VAL                    = 1
    VALUE                      = "Earth Radiation Budget Experiment"
   END_OBJECT                  = SENSORCHARACTERISTICNAME

   OBJECT                      = SENSORSHORTNAME
    CLASS                      = "1"
    NUM_VAL                    = 1
    VALUE                      = "ERBE-Scanner"
   END_OBJECT                  = SENSORSHORTNAME

   OBJECT                      = INSTRUMENTSHORTNAME
    CLASS                      = "1"
    NUM_VAL                    = 1
    VALUE                      = "ERBE"
   END_OBJECT                  = INSTRUMENTSHORTNAME

  END_OBJECT                 = SENSORCHARACTERISTICCONTAINER

 END_GROUP                 = SENSORCHARACTERISTIC

 OBJECT                    = SENSORCHARACTERISTICUNIT
  NUM_VAL                  = 1
  VALUE                    = "Watts/m**2/sr"
 END_OBJECT                = SENSORCHARACTERISTICUNIT
```

```
 OBJECT                  = LONGNAME
 VALUE                   = "ERBE S-8 PROCESSED ARCHIVAL TAPE"
 NUM_VAL                 = 1
END_OBJECT               = LONGNAME

OBJECT                   = VERSIONID
 NUM_VAL                 = 1
 VALUE                   = 1
END_OBJECT               = VERSIONID

OBJECT                   = PRODUCERAGENCY
 VALUE                   = "NASA"
 NUM_VAL                 = 1
END_OBJECT               = PRODUCERAGENCY

OBJECT                   = PRODUCERINSTITUTION
 VALUE                   = "LaRC"
 NUM_VAL                 = 1
END_OBJECT               = PRODUCERINSTITUTION

OBJECT                   = PROJECTID
 VALUE                   = "ERBE"
 NUM_VAL                 = 1
END_OBJECT               = PROJECTID

OBJECT                   = DATAFORMATTYPE
 VALUE                   = "HDF-EOS"
 NUM_VAL                 = 1
END_OBJECT               = DATAFORMATTYPE

OBJECT                   = SPACECRAFT
 NUM_VAL                 = 1
 VALUE                   = "ERBS"
END_OBJECT               = SPACECRAFT

OBJECT                   = DATATYPE
 VALUE                   = "Level 2"
 NUM_VAL                 = 1
END_OBJECT               = DATATYPE

OBJECT                   = COLLECTIONDESCRIPTION
 VALUE                   = "Scanner radiometric and geolocation data."
 NUM_VAL                 = 1
END_OBJECT               = COLLECTIONDESCRIPTION

OBJECT                   = TEMPORALKEYWORD
 VALUE                   = "Daily"
 NUM_VAL                 = 1
END_OBJECT               = TEMPORALKEYWORD

OBJECT                   = TEMPORALRANGETYPE
 VALUE                   = "CONTINUOUS RANGE"
```

```
   NUM_VAL                = 1
 END_OBJECT               = TEMPORALRANGETYPE

 OBJECT                   = ARCHIVECENTER
  VALUE                   = "LaRC"
  NUM_VAL                 = 1
 END_OBJECT               = ARCHIVECENTER

 OBJECT                   = CONTACTFIRSTNAME
  VALUE                   = "James"
  NUM_VAL                 = 1
 END_OBJECT               = CONTACTFIRSTNAME

 OBJECT                   = CONTACTLASTNAME
  VALUE                   = "Kibler"
  NUM_VAL                 = 1
 END_OBJECT               = CONTACTLASTNAME

 OBJECT                   = CONTACTJOBPOSITION
  VALUE                   = "Project Manager"
  NUM_VAL                 = 1
 END_OBJECT               = CONTACTJOBPOSITION

 OBJECT                   = CONTACTINSTRUCTIONS
  VALUE                   = "Email: j.f.kibler@larc.nasa.gov  Phone: (804) 864-
5386"
  NUM_VAL                 = 1
 END_OBJECT               = CONTACTINSTRUCTIONS

 OBJECT                   = MAINTENANCEANDUPDATEFREQUENCY
  VALUE                   = "None Planned"
  NUM_VAL                 = 1
 END_OBJECT               = MAINTENANCEANDUPDATEFREQUENCY

 OBJECT                   = DISCIPLINEKEYWORD
  VALUE                   = "Earth Science"
  NUM_VAL                 = 1
 END_OBJECT               = DISCIPLINEKEYWORD

 OBJECT                   = TOPICKEYWORD
  VALUE                   = "ATMOSPHERIC SCIENCE"
  NUM_VAL                 = 1
 END_OBJECT               = TOPICKEYWORD

 OBJECT                   = DOCUMENT
  VALUE                   =
"http://eosdis.larc.nasa.gov/HPDOCS/proj_addl_info.html"
  NUM_VAL                 = 1
 END_OBJECT               = DOCUMENT

 GROUP                    = PARAMETER
```

```
 OBJECT                    = PARAMETERCONTAINER
 CLASS                     = "1"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "1"
  NUM_VAL                    = 1
  VALUE                      = "Colatitude"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "2"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "2"
  NUM_VAL                    = 1
  VALUE                      = "Longitude"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "3"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "3"
  NUM_VAL                    = 1
  VALUE                      = "JulDay"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "4"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "4"
  NUM_VAL                    = 1
  VALUE                      = "Time"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "5"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "5"
  NUM_VAL                    = 1
  VALUE                      = "EarthSunDist"
 END_OBJECT                  = PARAMETERNAME
```

```
 END_OBJECT                = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "6"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "6"
  NUM_VAL                    = 1
  VALUE                      = "SCPOSX_Start"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "7"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "7"
  NUM_VAL                    = 1
  VALUE                      = "SCPOSX_End"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "8"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "8"
  NUM_VAL                    = 1
  VALUE                      = "SCPOSY_Start"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "9"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "9"
  NUM_VAL                    = 1
  VALUE                      = "SCPOSY_End"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "10"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "10"
```

```
   NUM_VAL                    = 1
   VALUE                      = "SCPOSZ_Start"
  END_OBJECT                  = PARAMETERNAME

 END_OBJECT                 = PARAMETERCONTAINER

 OBJECT                     = PARAMETERCONTAINER
  CLASS                     = "11"

  OBJECT                     = PARAMETERNAME
   CLASS                     = "11"
   NUM_VAL                   = 1
   VALUE                     = "SCPOSZ_End"
  END_OBJECT                 = PARAMETERNAME

 END_OBJECT                 = PARAMETERCONTAINER

 OBJECT                     = PARAMETERCONTAINER
  CLASS                     = "12"

  OBJECT                     = PARAMETERNAME
   CLASS                     = "12"
   NUM_VAL                   = 1
   VALUE                     = "SCVELX_Start"
  END_OBJECT                 = PARAMETERNAME

 END_OBJECT                 = PARAMETERCONTAINER

 OBJECT                     = PARAMETERCONTAINER
  CLASS                     = "13"

  OBJECT                     = PARAMETERNAME
   CLASS                     = "13"
   NUM_VAL                   = 1
   VALUE                     = "SCVELX_End"
  END_OBJECT                 = PARAMETERNAME

 END_OBJECT                 = PARAMETERCONTAINER

 OBJECT                     = PARAMETERCONTAINER
  CLASS                     = "14"

  OBJECT                     = PARAMETERNAME
   CLASS                     = "14"
   NUM_VAL                   = 1
   VALUE                     = "SCVELY_Start"
  END_OBJECT                 = PARAMETERNAME

 END_OBJECT                 = PARAMETERCONTAINER

 OBJECT                     = PARAMETERCONTAINER
  CLASS                     = "15"
```

```
  OBJECT                      = PARAMETERNAME
   CLASS                      = "15"
   NUM_VAL                    = 1
   VALUE                      = "SCVELY_End"
  END_OBJECT                  = PARAMETERNAME

 END_OBJECT                 = PARAMETERCONTAINER

 OBJECT                     = PARAMETERCONTAINER
  CLASS                     = "16"

  OBJECT                      = PARAMETERNAME
   CLASS                      = "16"
   NUM_VAL                    = 1
   VALUE                      = "SCVELZ_Start"
  END_OBJECT                  = PARAMETERNAME

 END_OBJECT                 = PARAMETERCONTAINER

 OBJECT                     = PARAMETERCONTAINER
  CLASS                     = "17"

  OBJECT                      = PARAMETERNAME
   CLASS                      = "17"
   NUM_VAL                    = 1
   VALUE                      = "SCVELZ_End"
  END_OBJECT                  = PARAMETERNAME

 END_OBJECT                 = PARAMETERCONTAINER

 OBJECT                     = PARAMETERCONTAINER
  CLASS                     = "18"

  OBJECT                      = PARAMETERNAME
   CLASS                      = "18"
   NUM_VAL                    = 1
   VALUE                      = "SC_Nadir_Colat_Start"
  END_OBJECT                  = PARAMETERNAME

 END_OBJECT                 = PARAMETERCONTAINER

 OBJECT                     = PARAMETERCONTAINER
  CLASS                     = "19"

  OBJECT                      = PARAMETERNAME
   CLASS                      = "19"
   NUM_VAL                    = 1
   VALUE                      = "SC_Nadir_Colat_End"
  END_OBJECT                  = PARAMETERNAME

 END_OBJECT                 = PARAMETERCONTAINER
```

```
 OBJECT                    = PARAMETERCONTAINER
 CLASS                     = "20"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "20"
  NUM_VAL                    = 1
  VALUE                      = "SC_Nadir_Lon_Start"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "21"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "21"
  NUM_VAL                    = 1
  VALUE                      = "SC_Nadir_Lon_End"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "22"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "22"
  NUM_VAL                    = 1
  VALUE                      = "Sun_Pos_Colat"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "23"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "23"
  NUM_VAL                    = 1
  VALUE                      = "Sun_Pos_Lon"
 END_OBJECT                  = PARAMETERNAME

END_OBJECT                 = PARAMETERCONTAINER

OBJECT                     = PARAMETERCONTAINER
 CLASS                     = "24"

 OBJECT                      = PARAMETERNAME
  CLASS                      = "24"
  NUM_VAL                    = 1
  VALUE                      = "Orbit_Number"
```

```
  END_OBJECT                  = PARAMETERNAME

END_OBJECT                  = PARAMETERCONTAINER

OBJECT                      = PARAMETERCONTAINER
 CLASS                      = "25"

 OBJECT                       = PARAMETERNAME
  CLASS                       = "25"
  NUM_VAL                     = 1
  VALUE                       = "Total_Rad"
 END_OBJECT                    = PARAMETERNAME

END_OBJECT                  = PARAMETERCONTAINER

OBJECT                      = PARAMETERCONTAINER
 CLASS                      = "26"

 OBJECT                       = PARAMETERNAME
  CLASS                       = "26"
  NUM_VAL                     = 1
  VALUE                       = "SW_Rad"
 END_OBJECT                    = PARAMETERNAME

END_OBJECT                  = PARAMETERCONTAINER

OBJECT                      = PARAMETERCONTAINER
 CLASS                      = "27"

 OBJECT                       = PARAMETERNAME
  CLASS                       = "27"
  NUM_VAL                     = 1
  VALUE                       = "LW_Rad"
 END_OBJECT                    = PARAMETERNAME

END_OBJECT                  = PARAMETERCONTAINER

OBJECT                      = PARAMETERCONTAINER
 CLASS                      = "28"

 OBJECT                       = PARAMETERNAME
  CLASS                       = "28"
  NUM_VAL                     = 1
  VALUE                       = "FOV_Zenith_Viewing"
 END_OBJECT                    = PARAMETERNAME

END_OBJECT                  = PARAMETERCONTAINER

OBJECT                      = PARAMETERCONTAINER
 CLASS                      = "29"

 OBJECT                       = PARAMETERNAME
```

```
   CLASS                      = "29"
  NUM_VAL                     = 1
  VALUE                       = "FOV_Zenith_Sun"
 END_OBJECT                   = PARAMETERNAME

END_OBJECT                  = PARAMETERCONTAINER

OBJECT                      = PARAMETERCONTAINER
 CLASS                      = "30"

 OBJECT                       = PARAMETERNAME
  CLASS                       = "30"
  NUM_VAL                     = 1
  VALUE                       = "FOV_Rel_Azimuth"
 END_OBJECT                   = PARAMETERNAME

END_OBJECT                  = PARAMETERCONTAINER

OBJECT                      = PARAMETERCONTAINER
 CLASS                      = "31"

 OBJECT                       = PARAMETERNAME
  CLASS                       = "31"
  NUM_VAL                     = 1
  VALUE                       = "Unfiltered_SW"
 END_OBJECT                   = PARAMETERNAME

END_OBJECT                  = PARAMETERCONTAINER

OBJECT                      = PARAMETERCONTAINER
 CLASS                      = "32"

 OBJECT                       = PARAMETERNAME
  CLASS                       = "32"
  NUM_VAL                     = 1
  VALUE                       = "Unfiltered_LW"
 END_OBJECT                   = PARAMETERNAME

END_OBJECT                  = PARAMETERCONTAINER

OBJECT                      = PARAMETERCONTAINER
 CLASS                      = "33"

 OBJECT                       = PARAMETERNAME
  CLASS                       = "33"
  NUM_VAL                     = 1
  VALUE                       = "TOA_EST_SW"
 END_OBJECT                   = PARAMETERNAME

END_OBJECT                  = PARAMETERCONTAINER

OBJECT                      = PARAMETERCONTAINER
```

```
  CLASS                    = "34"

  OBJECT                     = PARAMETERNAME
   CLASS                     = "34"
   NUM_VAL                   = 1
   VALUE                     = "TOA_EST_LW"
  END_OBJECT                 = PARAMETERNAME

 END_OBJECT                = PARAMETERCONTAINER

 OBJECT                    = PARAMETERCONTAINER
  CLASS                    = "35"

  OBJECT                     = PARAMETERNAME
   CLASS                     = "35"
   NUM_VAL                   = 1
   VALUE                     = "Scanner_FOV_SceneID"
  END_OBJECT                 = PARAMETERNAME

 END_OBJECT                = PARAMETERCONTAINER

 OBJECT                    = PARAMETERCONTAINER
  CLASS                    = "36"

  OBJECT                     = PARAMETERNAME
   CLASS                     = "36"
   NUM_VAL                   = 1
   VALUE                     = "Scan_Ops_Flags"
  END_OBJECT                 = PARAMETERNAME

 END_OBJECT                = PARAMETERCONTAINER

 OBJECT                    = PARAMETERCONTAINER
  CLASS                    = "37"

  OBJECT                     = PARAMETERNAME
   CLASS                     = "37"
   NUM_VAL                   = 1
   VALUE                     = "Scan_Rad_Flags_Total"
  END_OBJECT                 = PARAMETERNAME

 END_OBJECT                = PARAMETERCONTAINER

 OBJECT                    = PARAMETERCONTAINER
  CLASS                    = "38"

  OBJECT                     = PARAMETERNAME
   CLASS                     = "38"
   NUM_VAL                   = 1
   VALUE                     = "Scan_Rad_Flags_SW"
  END_OBJECT                 = PARAMETERNAME
```

```
   END_OBJECT                = PARAMETERCONTAINER

  OBJECT                     = PARAMETERCONTAINER
   CLASS                     = "39"

   OBJECT                      = PARAMETERNAME
    CLASS                      = "39"
    NUM_VAL                    = 1
    VALUE                      = "Scan_Rad_Flags_LW"
   END_OBJECT                  = PARAMETERNAME

  END_OBJECT                 = PARAMETERCONTAINER

  OBJECT                     = PARAMETERCONTAINER
   CLASS                     = "40"

   OBJECT                      = PARAMETERNAME
    CLASS                      = "40"
    NUM_VAL                    = 1
    VALUE                      = "Scan_Rad_Flags_FOV"
   END_OBJECT                  = PARAMETERNAME

  END_OBJECT                 = PARAMETERCONTAINER

 END_GROUP                 = PARAMETER

END_GROUP                = ARCHIVEDMETADATA

END
```

8.3.2 A FORTRAN Example of Metadata Write

The following FORTRAN code fragment is an example of how a user can write granule metadata (or inventory metadata) into their HDF-EOS file. The Metadata Configuration File (MCF), which the code accesses is given in Section 8.3.2.2. The output ODL file which results from running the code in 8.3.2.1. is given in Section 8.3.2.3. Details on Metadata Configuration Files and Metadata toolkit in general can be found in *The SDP Toolkit Users Guide for the ECS Project, Section 6.2.1 and Appendix J.*

8.3.2.1 FORTRAN Code

```
            include "PGS_MET_13.f"
            include "PGS_MET.f"
C     the file id must also be defined in the PCF as follows

C 10253|hdftestfile|/home/asiyyid/pgstest/fortran/|||hdftestfile|1

            integer MODIS_FILE
            parameter(MODIS_FILE = 10253)
            integer INVENTORYMETADATA
            parameter(INVENTORYMETADATA = 2)

C the group have to be defined as 49 characters long. The C interface
 is 50.

C The cfortran.h mallocs an extra 1 byte for the null character '\0/',
 therefor makeing the actual length of a string pass of 50.

            character*PGS_MET_GROUP_NAME_L groups(PGS_MET_NUM_OF_GROUP)
            integer pgs_met_init
            integer pgs_met_setattr_d
            integer pgs_met_getsetattr_d
            integer pgs_met_write
            integer pgs_met_getpcattr_d
            integer pgs_met_getconfigdata_d
            integer pgs_met_remove
            integer hdfReturn
            integer result
            integer sdid
            integer access
            integer sfstart
            integer sfend
            character*30 datetime
            double precision dval

            fileName = "/home/asiyyid/pgstest/fortran/hdftestfile"
            result = pgs_met_init(PGSd_MET_MCF_FILE,groups)

            if (result.NE.PGS_S_SUCCESS) then
      print *, "Initialization error. See Logstatus for details"
            endif
```

```
      dval = 111.11

      result = pgs_met_setattr_d(groups[INVENTORYMETADATA],
      "WestBoundingCoordinate",dval)

      dval = 0

      result = PGS_MET_GetSetAttr_d(groups[INVENTORYMETADATA],
1     "WestBoundingCoordinate",dval)

      print *, sval

      dval = 222.22

      result = pgs_met_setattr_d(groups[INVENTORYMETADATA],
1     "NorthBoundingCoordinate",dval)

      dval = 0

      result = PGS_MET_GetSetAttr_d(groups[INVENTORYMETADATA],
1     "NorthBoundingCoordinate",dval)

      print *, dval

      dval = 333.33

      result = pgs_met_setattr_d(groups[INVENTORYMETADATA],
1     "EastBoundingCoordinate",dval)

      dval = 0

      result = PGS_MET_GetSetAttr_d(groups[INVENTORYMETADATA],
1     "EastBoundingCoordinate",dval)

      print *, dval


      dval = 444.44
      resultu = pgs_met_setattr_d(groups[INVENTORYMETADATA],
1     "SouthBoundingCoordinate",dval)

      dval = 0

      result = PGS_MET_GetSetAttr_d(groups[INVENTORYMETADATA],
1     "SouthBoundingCoordinate",dval)

      print *, dval

      sdid = sfstart(fileName, access)

      if (sdis.EQ.-1) then
```

```
 print *, "Failed to open the hdf file"
      endif

      result = pgs_met_write(group(INVENTORYMETADATA),
1     "coremetadata",sdid)

      if (result.NE.PGS_S_SUCCESS. and.
      result.NE.PGSMET_W_METADATA_NOT_SET) then
if (result.EQ.PGSMET_W_METADATA_NOT_SET) then
       print *, "Some of the mandatory parameters were not set"
 else
            print *, "ASCII write failed"
endif
      endif

      hdfReturn = sfend(sdis)

      dval = 11.11
      result = pgs_met_getpcattr_d(MODIS_FILE, 1, "coremetadata",
1     "WestBoundingCoordinate", dvals)

      printf *, dval

      dval = 22.22
      result = pgs_met_getpcattr_d(MODIS_FILE, 1, "coremetadata",
1     "NorthBoundingCoordinate", dvals)

      printf *, dval

      dval = 33.33
      result = pgs_met_getpcattr_d(MODIS_FILE, 1, "coremetadata",
1     "EastBoundingCoordinate", dvals)

      printf *, dval

      dval = 44.44
      result = pgs_met_getpcattr_d(MODIS_FILE, 1, "coremetadata",
1     "SouthBoundingCoordinate", dvals)

      datetime = ""

      result = pgs_met_getconfigdata_s("LONGNAME", datetime)

      dval = 0

      result = pgs_met_getconfigdata_d("CENTRELATITUDE", dval)

      if(result.NE.PGS_S_SUCCESS) then
       print *, "getconfigdata failed. See Logstatus for details"
      endif

      print *, dval, datetime
```

```
            result = pgs_met_remove()

            print *, "SUCCESS"

            end
```

8.3.2.2 The Metadata Configuration File (MCF) for Code in Section 8.3.2.1.

```
C  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
```

8.3.2.3 The ODL Output File Which Results from Running Code in Section 8.3.2.1

```
C  BoundingRectangle

   GROUP                       = BOUNDINGRECTANGLE

    OBJECT                       = WESTBOUNDINGCOORDINATE
     NUM_VAL                     = 1
     VALUE                       = 111.110000
    END_OBJECT                   = WESTBOUNDINGCOORDINATE

    OBJECT                       = NORTHBOUNDINGCOORDINATE
     NUM_VAL                     = 1
     VALUE                       = 222.220000
    END_OBJECT                   = NORTHBOUNDINGCOORDINATE

    OBJECT                       = EASTBOUNDINGCOORDINATE
     NUM_VAL                     = 1
     VALUE                       = 333.330000
    END_OBJECT                   = EASTBOUNDINGCOORDINATE

    OBJECT                       = SOUTHBOUNDINGCOORDINATE
     NUM_VAL                     = 1
     VALUE                       = 444.440000
    END_OBJECT                   = SOUTHBOUNDINGCOORDINATE

   END_GROUP                   = BOUNDINGRECTANGLE
```

Appendix A. Installation and Maintenance

A.1 Installation Procedures

A.1.1 Preliminary Step

Before installing HDFEOS, you must already have installed NCSA HDF, Version 4.1r1, on your host. The installation script will prompt for the paths to the HDF include and library directories. Please see the SDP Toolkit Users Guide for the ECS Project, 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 25 Mb of free space.

2) Copy the file HDF-EOSv2.6.tar.Z to the target directory by

typing the command:
cp HDF-EOSv2.6.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-EOSv2.6.tar.Z | tar xvf -
```

This will create a subdirectory of the current directory called 'hdfeos'. This is the top-level HDFEOS directory, which contains the full HDFEOS directory structure.

A.1.3 Starting the Installation Procedure

1) Set your default directory to the top-level HDFEOS directory by typing the command:
cd hdfeos

2) Select installation options.
Currently, the only options are those specific to the SGI Power Challenge platform.

On the SGI Challenge, the *default* is to build HDFEOS in 64-bit mode, which is the same as the Toolkit. The following table gives the option to specify the appropriate architecture to be built:

binary format	architecture	<install-options>
old 32-bit	sgi	-sgi
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 HDF4.1r1 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.1.4 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 HDFEOS

In order to compile and link programs with the HDFEOS you must access the HDFEOS include and library files. To do this be sure that your makefiles include something like the following:

```
INCLUDE = -I. -I$(HDFEOS_INC) -I$(HDFINC)
LIBRARY = -L. -L$(HDFEOS_LIB) -L$(HDFLIB)
LDFLAGS = -lhdfeos -lGctp -lmfhdf -ldf -ljpeg -lnsl -lz -lm (Sun platform)
LDFLAGS = -lhdfeos -lGctp -lmfhdf -ldf -ljpeg -lz -lm   (others)
```

The environment variables HDFEOS_INC, HDFEOS_LIB, HDFINC and HDFLIB are set up by the HDFEOS environment scripts (see User Setup, above). The refer to the include and library directories for HDFEOS and HDF, respectively.

The INCLUDE macro should be included in all compilation statements. The LIBRARY an LDFLAGS macros should be included in all link statements.

A.2 Notes

1) Approved Platforms

HDFEOS 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 HDFEOS User's Guide and is also listed below.

Platform	OS	Version	C Compiler	FORTRAN77
Sun Sparc	Solaris	2.5.1	Sun C 4.0	Sun FORTRAN 4.0
HP 9000/735	HP–UX	A.10.01	HP C 10.11	HP FORTRAN 10.00
DEC Alpha	Digital UNIX	4.0	DEC C 4.0	DEC FORTRAN 4.1
IBM RS–6000	AIX	4.1	IBM C 3.1.4	IBM FORTRAN 3.2.5
SGI Power Challenge	IRIX	6.2	SGI C 7.2.1	SGI FORTRAN 7.2.1

Note: The compilers are supplied by the vendor. The SGI Power Challenge (64-bit mode) had the native SGI FORTRAN 90 7.0.

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:

$BRAND	Architecture
dec	DEC Alpha
hp	HP 9000
ibm	IBM RS-6000
sgi	SGI Power Challenge (old-style 32-bit mode)
sgi64	SGI Power Challenge (64-bit mode)
sun5	Sun: SunOS 5.x

3) Directory and File Environment Variables

In order to use the HDFEOS library and utilities, a number of environment variables MUST be set up to point to HDFEOS 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
HDFEOS_HOME	<install-path>/hdfeos	top-level directory
(where <install-path> is the absolute directory path above hdfeos)		
HDFEOS_BIN	$HDFEOS_HOME/bin/$BRAND	executable files
HDFEOS_INC	$HDFEOS_HOME/include	header files
HDFEOS_LIB	HDFEOS_HOME/lib/$BRAND	library files
HDFEOS_OBJ	$HDFEOS_HOME/obj/$BRAND	object files
HDFEOS_SRC	$HDFEOS_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 HDFEOS, 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-EOS 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-EOS2.6v1.00_ 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 ECS Landover facility have been set up for user response:

- pgstlkit@eos.hitc.com and
- hdfeos@eos.hitc.com

2) Users will e–mail problem reports and comments to the above account. A receipt will be returned to the sender. A workoff plan for the discrepancy will be developed and status report issued once a month. 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: 301–925–0781. Calls will be returned as soon as possible. We note, however, that email is our preferred method of responding to users.

This page intentionally left blank.

Abbreviations and Acronyms

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

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

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

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