

170-TP-007-001

# Writing HDF-EOS Grid Products for Optimum Subsetting Services

Technical Paper

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

December 1996

Prepared Under Contract NAS5-60000

## RESPONSIBLE ENGINEER

<u>Shaun de Witt /s/</u>	12/13/96
Shaun de Witt, Senior Software Engineer EOSDIS Core System Project	Date

## SUBMITTED BY

<u>Steve Marley /s/</u>	12/13/96
Steve Marley, Senior Software Architect, EOSDIS Core System Project	Date

Hughes Information Technology Systems  
Upper Marlboro, Maryland

This page intentionally left blank.

# Abstract

---

This document provides a guideline for writing gridded data into HDF-EOS files to allow for optimum subsetting performance. HDF-EOS builds on the standard HDF libraries produced by NCSA for storing data, and extends it to support commonly used data structures used for earth science data. It allows for the construction of grids (in standard projections used by GCTP), swaths and point data sets. HDF and HDF-EOS are self-describing data formats and allow a great deal of flexibility in the internal organization of the data. Requirements exist within ECS to allow for subsetting and subsampling services to be performed on these data sets; specifically subsetting by geographic co-ordinate, altitude and time. Since there are a large number of methods for organizing the data, much specific code would need to be written for each product to allow these services to take place. This document is intended as a guide to writing gridded data in a form suitable to allow subsetting as described above, minimizing the amount of specialist code which would be needed for these services to take place.

**Keywords:** Grid, HDF, HDF-EOS, format, subsetting, services.

This page intentionally left blank.

# Contents

---

## Abstract

### 1. Introduction

1.1 Purpose.....	1-1
1.2 Organization.....	1-1
1.3 Reference Documents .....	1-1
1.4 Applicable Documents.....	1-1

### 2. Services Provided on Gridded Data by ECS1

2.1 Choosing an HDF-EOS Structure.....	2-1
2.2 Subsampling Service.....	2-2
2.3 Subsetting.....	2-3
2.3.1 Subsetting by Region .....	2-3
2.3.2 Subsetting by Altitude.....	2-3
2.3.3 Subsetting by Time .....	2-4
2.3.4 Subsetting by Parameter.....	2-4
2.4 Output File Organization .....	2-4
2.5 Guide to Using this Document.....	2-4

### 3. Writing a File Containing Parameters with no Altitude or Temporal Information

3.1 Sample Code for Writing a Product.....	3-2
--	-----

## **4. Writing Data Sets with One or More Parameters With One Additional Dimension**

4.1 Understanding HDF-EOS Three Dimensional Grids.....	4-1
4.2 Steps to Creating a Grid within a File.....	4-1
4.2.1 Restrictions to Units.....	4-3
4.3 Sample Code .....	4-4
4.4 Writing Grid Data with String Type Altitude Values .....	4-7

## **5. Writing Data Sets with One or More Parameters With Two Additional Dimensions**

5.1 Writing Data as Four Dimensional Fields.....	5-1
5.1.1 Sample Code .....	5-2
5.2 Writing Data as a Series of Three Dimensional Fields.....	5-4
5.2.1 Sample Code .....	5-5

## **Figures**

2-1. Decision Tree for Choosing an Appropriate HDF-EOS Structure .....	2-2
2-2. Subsampling of a Grid.....	2-3
2-3. Roadmap for Navigating this Document .....	2-5
3-1. Schematic Representation of a Grid File Containing Several Parameters with no Altitudinal or Temporal Information .....	3-2
4-1. Combining Data with both Numerical and Textual Altitude Levels.....	4-8

## **Abbreviations and Acronyms**

# 1. Introduction

---

## 1.1 Purpose

Given the self-describing nature of HDF-EOS, there are a vast number of ways in which data could be organized within a file. Having many different data organization approaches across the EOS data sets will lead to inefficiencies in developing common data type services, like subsetting, that are sensitive to data organization. In order to facilitate the development of subsetting services, this document provides a set of guidelines for writing HDF-EOS formatted grid files which will require these services.

The author wishes to thank the HDF-EOS development team, especially Doug Ilg and Joel Gales, for their input to this document.

## 1.2 Organization

This paper is organized as follows:

Section 2 gives a brief overview of the subsetting which will be performed by ECS on request. Later sections deal specifically with the data organization for a number of cases. These take the user step by step through creating a grid file, together with simple code examples.

## 1.3 Reference Documents

The following documents were used in the preparation of this paper.

HDF-EOS User's Guide, 170-TP-005-001, 6/96

Science Data Processing Segment (SDPS) Database Design and Database Schema Specifications for the ECS Project, 311-CD-002-004, 12/95

Release-B SDPS Database Design and Database Schema Specifications, 311-CD-008-001, 5/96

## 1.4 Applicable Documents

The following material is related to this document and may be useful further reading.

Release A SCF Toolkit Users Guide, 333-CD-003-004

SDP Toolkit Primer for the ECS Project, 194-815-SI4-001, 4/95

HDF User's Guide, Version 4.0r2, NCSA, University of Illinois at Urbana-Champaign, 7/96

The Astronomical Almanac, US Naval Observatory, 1996

Writing HDF-EOS Swath Products for Optimum Subsetting Services, 170-TP-009-001, 12/95

Writing HDF-EOS Point Products for Optimum Subsetting Services, 170-TP-008-001, 12/95

Questions regarding technical information contained within this Paper should be addressed to the following ECS contact:

- ECS Contacts
  - Doug Ilg, Senior Software Engineer, (301) 925 0780, dilg@eos.hitc.com (HDF-EOS related questions)
  - Joel Gales, Senior Software Engineer, (301) 925 0782, jgales@eos.hitc.com (HDF-EOS related questions)
  - Kate Senehi, Senior Designer, (301) 925 4035, ksenehi@eos.hitc.com (Service related questions)
  - Alward Siyyid, Senior Software Engineer, (301) 925 0579, asiyyid@eos.hitc.com (General questions)

Questions concerning distribution or control of this document should be addressed to:

Data Management Office  
The ECS Project Office  
Hughes Information Technology Systems  
1616 McCormick Drive  
Upper Marlboro, Maryland 20774-5372

## 2. Services Provided on Gridded Data by ECS

---

In the Release B timeframe ECS will provide services for specific data sets as requested by Instrument Teams in agreements between themselves and ECS. All products requiring some archiving, either permanently or with a limited lifetime, will have the standard set of services; insert, update and acquire. In addition, higher level services will be available for specific ESDT's. Amongst these services are subset and subsample.

It should be noted that subsetting and subsampling will only be performed on HDF-EOS format files during the release B timeframe, since it is intended to utilize functionality within HDF-EOS to perform most of the services. These high level services will only be provided if Instrument Teams have indicated a need to perform them on particular ESDT's. Also, both subsetting and subsampling can be performed on at most a four dimensional grid (for example, latitude, longitude, altitude and time). ECS will not support these higher level services on higher dimensioned data.

The information contained in this and subsequent chapters is only relevant for data in HDF-EOS Grid format. Swath and Point data structures are dealt with separately in the technical papers 170-TP-008-001 and 170-TP-009-001. While this document does only provide guidelines to writing Grid files requiring subsetting, it is strongly recommended that data providers do follow the instructions contained for all files. This would allow users unfamiliar with a product to quickly and easily understand the contents. In addition, if a product requiring subsetting does not conform to these guidelines, the Data Provider must provide ECS with very detailed format information regarding the file organization, grid, field and dimension naming conventions, etc.

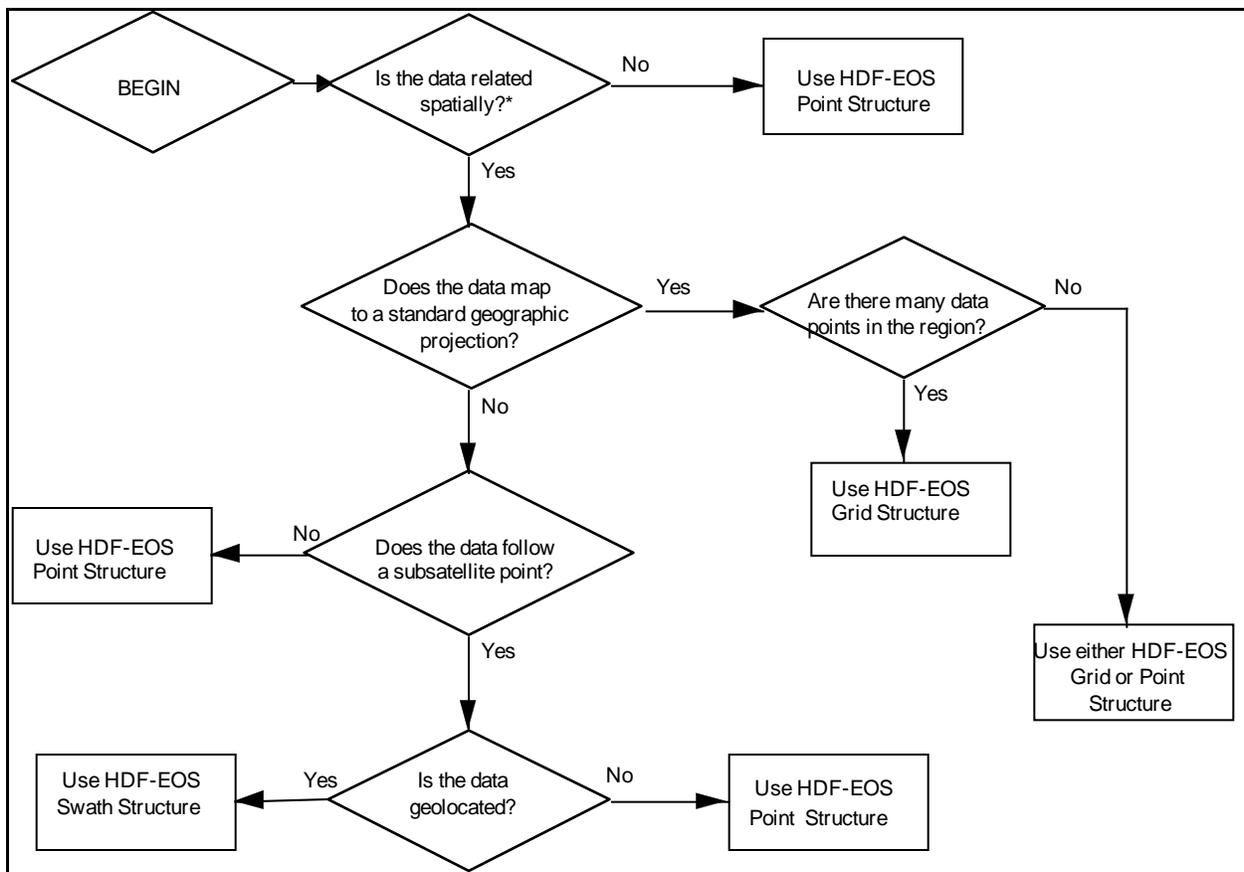
### 2.1 Choosing an HDF-EOS Structure

HDF-EOS has been developed as a part of the EOSDIS Core System, with the intent that it provide data producers with a standard format for earth science data archived within ECS. It is built on the widely used HDF libraries produced by NCSA, and is fully compatible with HDF. What it does provide is support for commonly used structures in earth science and remote sensing applications, namely point, swath and grid structures. It is important for a data provider to decide which, if any, of these structures should be used for a particular implementation. The decision tree in Figure 2.1 may help in this choice, but is intended for guidance only. The Instrument Team/Data Provider must individually decide which format best meets their needs on a product by product basis.

Note for the box reading "Use either HDF-EOS point or grid structure" the choice is up to the user. Using a grid will make the file larger, and many fill values may exist but the code is simpler. In addition HDF compression utilities may help reduce the file size. Use of the point structure may require more in terms of coding, but will result in a smaller file.

The first decision box requires the reader to understand what is meant by spatially related. This could be interpreted in a number of ways, but it is meant to indicate that the data is spatially

contiguous. As an example, an image of an area would be spatially related (all of the "pixels" are connected directly to others), while a set of meteorological reporting stations in different cities would not be related.



**Figure 2-1. Decision Tree for Choosing an Appropriate HDF-EOS Structure**

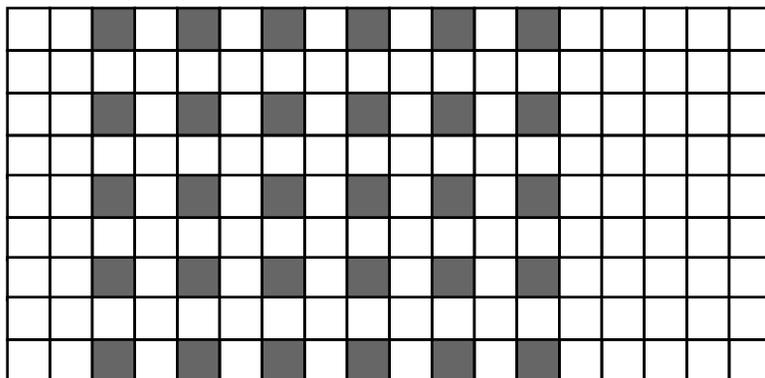
The remainder of this document only deals with the Grid Structure.

## 2.2 Subsampling Service

Subsampling will be implemented as a very coarse method of reducing the resolution of the data. It simply involves extracting every n'th pixel starting and stopping at a known point. This is done by providing the start, stride and edge required for the subsampling. More details regarding this can be found in the HDF Users Guide. Briefly, start is an array specifying where a user wishes to begin their subsetting, stride is an array specifying how many points to skip before reading the next point, and edge is an array specifying the number of data elements to read. Each of these arrays must be an integer, based on the row and column of the grid (rather than, for example, latitude and longitude).

Figure 2.2 shows an example of subsampling. In this case the data contained in the file is two dimensional of size 18x9. The filled squares indicate the values which would be extracted using a start of {2, 0} (note that the value of start is zero based), stride of {2,2} and edge of {6,5}.

Subsampling is useful for obtaining reduced resolution data sets for browsing or non-scientific use. The subsetted data will be returned again as a grid. It is possible to combine subsetting and subsampling requests, to further reduce the volume of data returned.



**Figure 2-2. Subsampling of a Grid**

## 2.3 Subsetting

Subsetting of a file is a means of extracting a portion of a data set relevant to a user's application. It is more sophisticated than subsampling, and retains the original resolution of the data set. Subsetting can be used if a user is only interested in a particular region, parameter or altitude for example. There are several types of subsetting which are available for HDF-EOS grid files, and these are dealt with in the following subsections.

### 2.3.1 Subsetting by Region

Subsetting by region allows you to specify a geographic region of interest. For example, the data set contains information over the whole world, but the user is only interested in data from the Amazonian Basin, then the user could select only that geographic region by passing in information regarding the bounding coordinates. These will represent two opposite corners of the region to be subsetted in decimal degrees. Note that only a bounding rectangle of data may be subsetted; subsetting on more complex shapes will not be implemented in the ECS Release B timeframe.

### 2.3.2 Subsetting by Altitude

Subsetting by altitude can allow a user to specify the range of altitudes in which they are interested. For example, if a data set contained a vertical profile of wind speed between 0 and

3000 m above sea level, but a user is only interested in windspeed between 500 and 1000m, these may be passed in as arguments to the vertical subsetting routines. Note that the units are important here, since the vertical profile may be given in one of a number of units (for example, meters, hPa, isentropic levels, etc.). Since often there is no simple means of providing conversion between units this will not be performed by the subsetting routines. It is the responsibility of the user to ensure that the correct units are supplied when requesting subsetting. Subsetting named altitude levels will also be supported, provided the name matches a valid domain value in one of the collection level attributes DepthDatumName or Note that throughout this paper depth and altitude are used interchangeably.

### **2.3.3 Subsetting by Time**

Along the same lines as subsetting by altitude, subsetting by time can allow a user to select only data from the temporal range. While it is anticipated that only a few HDF-EOS grid products will contain time series data, this functionality will be available for such data sets. The method of subsetting is similar to that for altitude, where the user must supply a start and stop time to extract the required data. For temporal subsetting some unit conversion is possible between the time format requested and the time format in which the data is archived (see later chapters).

### **2.3.4 Subsetting by Parameter**

In addition to the above, it will also be possible to subset by parameter. This parameter may be either geophysical (e.g. temperature) or product specific (e.g. band). Subsetting on more than one parameter will be permitted by allowing users to specify an array of parameters. The only restriction is that the parameter names requested must match those given in either of the collection level metadata attributes ParameterName or ECSVariableKeyword.

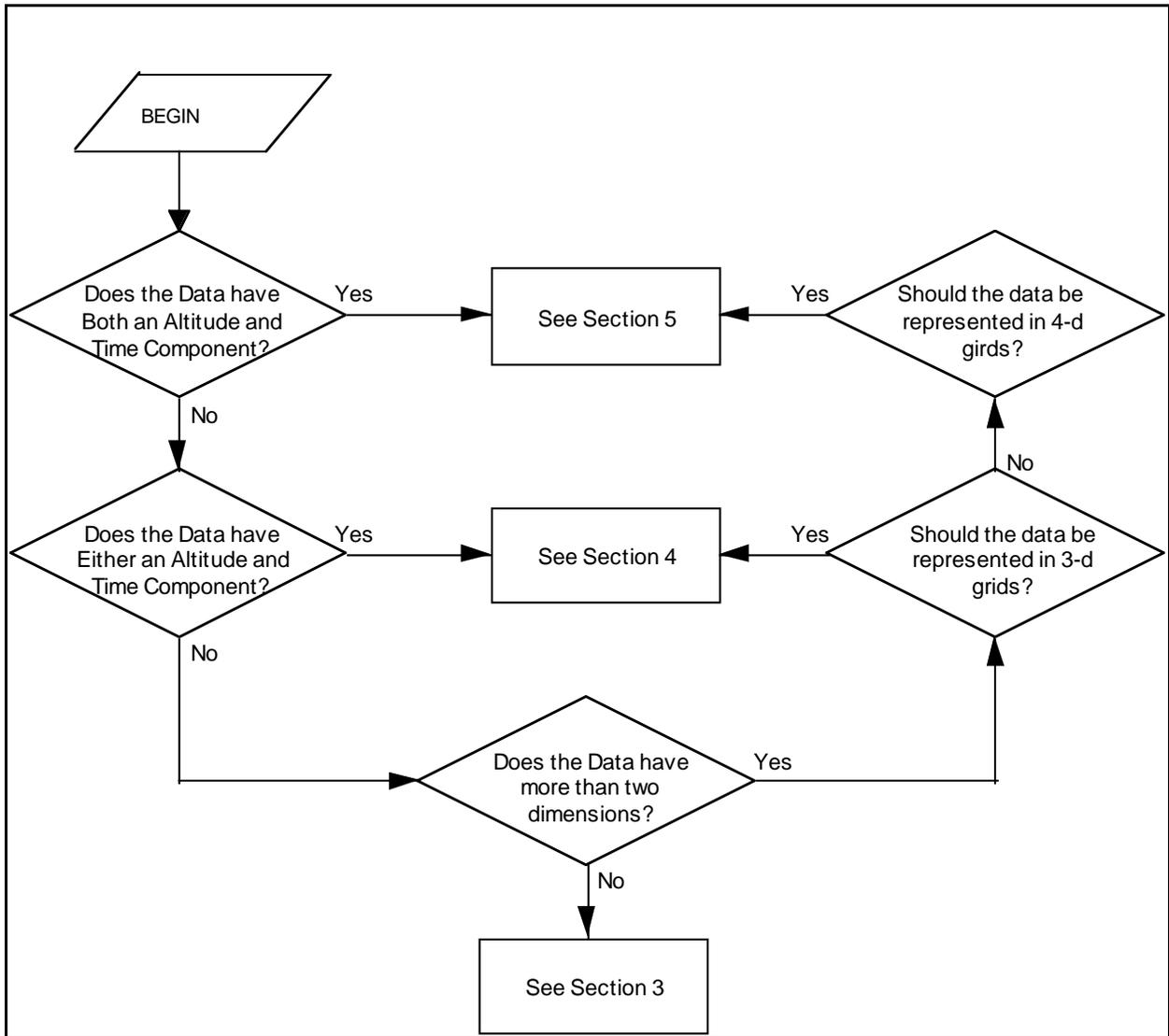
## **2.4 Output File Organization**

Whenever a request for subsetting or subsampling has been successfully completed, a new HDF-EOS file will be generated. Whenever possible, the organization of the data will be similar to that in the original file, with the same grid names and field names. The inventory and archive metadata will be copied from the original file to the subsetting/subsampled file, and the core attributes relating to bounding coordinates and time will be updated if necessary. HDF-EOS will generate new structural metadata. However, product specific and other core attributes will not be altered. Where a grid contains no relevant data, that grid will be completely omitted from the subsetting/subsampled file. Similarly, if a grid field contains no relevant data, it will be completely omitted. No "place holders" will be left showing where original data was located.

## **2.5 Guide to Using this Document**

This section provides a "road map" to the rest of this document. Since grid data can come in several different varieties, this section will lead the reader to the section most applicable for their needs. This guide takes the form of a simple decision tree, referencing a later section of this document. If the reader is still unsure which section to look at after looking at this roadmap, it is

suggested that each section should be read carefully, and refer back to HDF-EOS documentation for further assistance.



**Figure 2-3. Roadmap for Navigating this Document**

This page intentionally left blank.

### 3. Writing a File Containing Parameters with no Altitude or Temporal Information

---

The simplest case of a grid structure is one with no altitude or time dependencies. In this case, the only forms of subsetting which are applicable are subsetting by parameter and region.

The recommended steps to go through when writing the data for this format are as follows:

Step 0: Open the grid file using GDopen.

Step 1: Create a grid using GDcreate. The grid name does not have any special significance.

Step 2: Use GDdefprof and GDdeforigin to set the projection and origin for the grid.

Step 3: For each parameter to be written to the grid, define a field using GDdeffield. Only two dimensions should be specified, YDim and XDim. The field name be identical to one of the collection level attributes ECSVariableKeyword (derived from the GCMD keyword list) or ECSParameName (a user supplied parameter identifier). If this is not the case, then the data provider needs to supply ECS with a method of mapping between the field names and subsettable parameters.

Step 4: Detach from the grid using GDdetach in order to fix the size and information relating to the grid.

Step 5: Reattach to the grid using GDattach

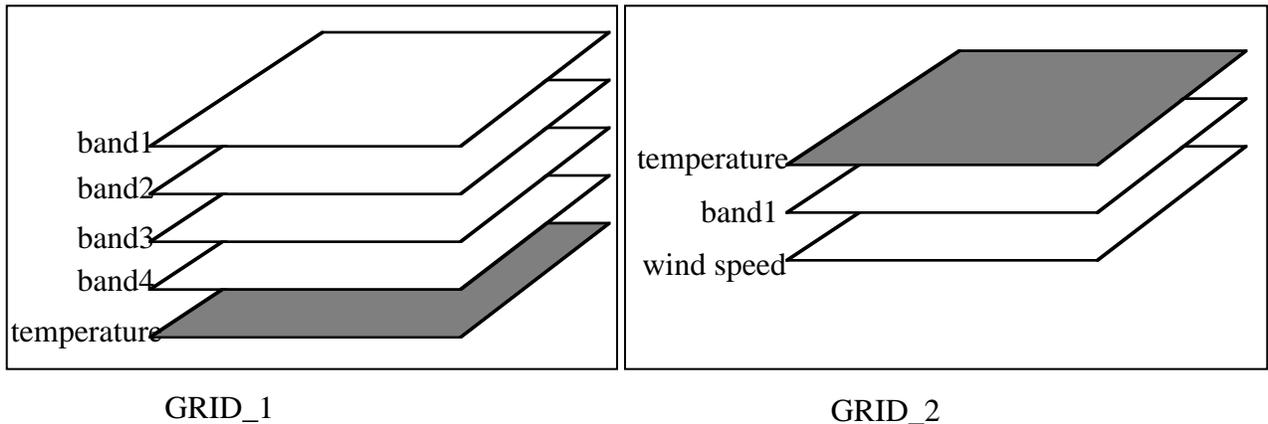
Step 6: Write the correct data into the correct field using GDwritefield.

Step 7: Detach from the grid using GDdetach.

Step 8: Repeat steps 1 to 7 for each additional grid required(i.e. different geographic regions and/or projections).

Step 9: Close the file using GDclose.

The data written to the file will be organized as shown in Figure 3-1. Each grid will contain a number of fields, each of which is uniquely identified. The same parameter name may be used in more than one grid. This diagram shows the case when both geophysical and non-geophysical parameters are contained within the grid (with field names "band1", "band2", "band3", "band4", "temperature" and "windspeed"). Two separate grids are shown, possibly indicating different geographic coordinates and/or projections, named "GRID\_1" and "GRID\_2". Each parameter is effectively a "layer" in the grid. Figure 3-1 also shows the effect of subsetting by parameter. The effect of requesting a subset by "temperature" would be to create a new file still containing two grids with the same grid name as the original file, but each grid would contain only a single field named "temperature" ( the gray layers indicated in the figure).



**Figure 3-1. Schematic Representation of a Grid File Containing Several Parameters with no Altitudinal or Temporal Information**

### 3.1 Sample Code for Writing a Product

The sample code shown below could be used to write the product illustrated in Figure 3-1. Where additional steps may be necessary, this is indicated by comments in the code. This sample has been written using the "C" language. The same calling sequence could be used using FORTRAN, since this language is supported by HDF-EOS. For simplicity, error checking is omitted.

```
#include <stdio.h>
#include <stdlib.h>
#include <hdf.h>
#include <mfhdf.h>
#include <HdfEosDef.h>

int main()
{
int32 gridId;
int32 fileId;
intn hdfReturn;
```

```

int band1DataA[65][65];
int band2Data[65][65];
int band3Data[65][65];
int band4Data[65][65];
float temperature1[65][65];
float windspeed[128][64];
int band1DataB[128][64];
float temperature2[128][64];
float64 ulCornerGrid1[2] = {12741994.0, 12741994.0};
float64 lrCornerGrid1[2] = {-12741994.0, -12741994.0};
float64 ulCornerGrid2[2] = {-180000000.0, 90000000.0};
float64 lrCornerGrid2[2] = {180000000.0, -90000000.0};
float64 projParameters[13] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                             0.0, 0.0, 0.0, 0.0, 0.0};

/*
The user must write code prior to this point to fill the data arrays described
above. The following assumes that the arrays have been filled and the user is
ready to write the data into an HDF-EOS product.
*/

/* First open the grid file */
fileId = GDopen("productFile.dat", DFACC_CREATE);

/* Now create the first grid in the file */
gridId = GDcreate(fileId, "GRID_1", 65, 65, ulCornerGrid1, lrCornerGrid1);

/*
Now we define the origin and projection for this grid. For this case we deal
with a polar stereographic grid, with a longitude below the pole of 0 and a
true scale at 90
*/
projParameters[5] = 90000000.0;
hdfReturn = GDdefproj(gridId, GCTP_PS, (int32)NULL, 19, projParameters);
hdfReturn = GDdeforigin(gridId, HDFE_GD_UL);

/*
Now define each field for the data to be inserted into the first grid

```

```

*/
hdfReturn = GDdeffield(gridId, "band1", "YDim,XDim", DFNT_INT16,
HDFE_NOMERGE);
hdfReturn = GDdeffield(gridId, "band2", "YDim,XDim", DFNT_INT16,
HDFE_NOMERGE);
hdfReturn = GDdeffield(gridId, "band3", "YDim,XDim", DFNT_INT16,
HDFE_NOMERGE);
hdfReturn = GDdeffield(gridId, "band4", "YDim,XDim", DFNT_INT16,
HDFE_NOMERGE);
hdfReturn = GDdeffield(gridId, "temperature", "YDim,XDim", DFNT_FLOAT32,
HDFE_NOMERGE);

/* Detach and reattach to the grid */
hdfReturn = GDdetach(gridId);
gridId = GDattach(fileId, "GRID_1");

/*
Having defined these fields we can now write the appropriate data to them.
*/
hdfReturn = GDwritefield(gridId, "band1", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)band1DataA);
hdfReturn = GDwritefield(gridId, "band2", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)band2Data);
hdfReturn = GDwritefield(gridId, "band3", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)band3Data);
hdfReturn = GDwritefield(gridId, "band4", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)band4Data);
hdfReturn = GDwritefield(gridId, "temperature", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)temperature1);

/* We can now detach from this grid */
hdfReturn = GDdetach(gridId);

/* We can now deal with the second grid in a similar manner to the first. In
this case it is assumed that we are writing a geographic grid using the
default GCTP spherical earth model, with the central meridian at 0*/
gridId = GDcreate(fileId, "GRID_2", 128, 64, ulCornerGrid2, lrCornerGrid2);
projParameters[5] = 0.0;
hdfReturn = GDdefproj(gridId, GCTP_GEO, (int32)NULL, 19, projParameters);
hdfReturn = GDdeforigin(gridId, HDFE_GD_LL);

```

```

/* Now define the fields for the second grid */
hdfReturn = GDdefield(gridId, "windspeed", "YDim, XDim", DFNT_FLOAT32,
HDFE_NOMERGE);

hdfReturn = GDdefield(gridId, "band1", "YDim, XDim", DFNT_INT16,
HDFE_NOMERGE);

hdfReturn = GDdefield(gridId, "temperature", "YDim, XDim", DFNT_FLOAT32,
HDFE_NOMERGE);

/* Detach and reattach to the grid */
hdfReturn = GDdetach(gridId);
gridId = GDattach(fileId, "GRID_2");

/* write the data to these fields */
hdfReturn = GDwritefield(gridId, "windspeed", NULL, NULL, NULL,
(VOIDP)windspeed);

hdfReturn = GDwritefield(gridId, "band1", NULL, NULL, NULL,
(VOIDP)band1DataB);

hdfReturn = GDwritefield(gridId, "temperature", NULL, NULL, NULL,
(VOIDP)temperature2);

/* Detach from the Grid */
hdfReturn = GDdetach(gridId);

/* Close the file */
hdfReturn = GDclose(fileId);
exit(0);
}

```

This page intentionally left blank.

## 4. Writing Data Sets with One or More Parameters With One Additional Dimension

---

It is anticipated that a number of products which require higher level services could contain data not only in terms of latitude and longitude, but also altitude profiles or time sequences. HDF-EOS provides a simple means of storing this "three dimensional" data, and extracting subsets by location and along the third dimension.

ECS can provide subsetting along any third dimension, although the data provider would need to provide the appropriate code. ECS will provide generic routines to subset by either altitude, depth or time. It is possible to have other dimension types (for example, wavelength, band, camera number), but specific code extensions will need to be written to provide for these, and the grid construction should follow the guidelines in this chapter.

When the altitudinal (or any data) dimension is not numerical, it is still possible to write this data (see Section 4.4). For example, this would cover the case where science values are given at SuraeOfEarth, Tropopause, and TopOfAtmosphere.

### 4.1 Understanding HDF-EOS Three Dimensional Grids

HDF-EOS provides generic means for storing three dimensional data. In understanding the organization of this data, it is useful to become familiar with HDF-EOS terms. Data is not stored directly in grids, but as fields in a 2-d grid. The grid itself can be considered an (x,y) plane and the data is stored as one or more boxes, or fields, with the same x and y dimensions as the grid. Each field may have a unique third dimension, or several fields may share a dimension. The field must be given a name and this name is the basis for subsetting by Parameter.

Other single dimensional fields may exist on the grid and are used for storing information about the dimensions of the data fields. In this chapter, we deal with three dimensional data sets (although, as will be seen later, this requires the creation of one dimensional fields).

### 4.2 Steps to Creating a Grid within a File

The steps to create the preferred format for three dimensional fields within a file are as follows. Additional Requirements on the data organization are explicitly stated at the end of this section.

Step 0: Open the grid file using GDopen.

Step 1: Create a grid using GDcreate. The grid name has no special significance.

Step 2: Use GDdefproj and GDdeforigin to set the projection and origin for the grid.

Step 3: Each subsettable parameter will be a field in the grid. For each field, define a third dimension using GDdefdim. **Note** : if two fields have the same size and occur at the same values along that dimension (e.g. the user has relative humidity and temperature

data at 1000hPa, 500hPa and 250hPa), the dimension need only be defined once. ECS will provide default routines for the case where this dimension is temporal or altitudinal. In the former case, the dimension should be named *time\_n\_units*, while for the latter case it should be *altitude\_n\_units*. In both cases *n* is an integer from 1 and is used to distinguish cases where fields have the same units but have data at different values along this dimension. **Note:** There are some restrictions to the units, as indicated later in this section.

Step 4: Use `GDdefield` to define the field for each parameter. The field name should represent the parameter contained in that field, and should map to one of the collection level attributes `ECSVariableKeyword` (derived from the `GCMD` keyword list) or `ECSParameterName` (a user supplied parameter identifier). The dimensions for the field should be the name of the appropriate dimension created in step 3, "YDim" and "XDim".

Step 5: For each dimension defined in step 3, create a field with the same name as the dimension, with same dimension. For example, `GDdefield(gridId, "altitude_1_hPa", "altitude_1_hPa", DFNT_INT32, HDFE_NOMERGE)`.

Step 6: Detach from the grid using `GDdetach` in order to fix the size and information relating to the grid.

Step 7: Reattach to the grid using `GDattach`

Step 8: For each parameter, write the data to the appropriate field using `GDwritefield`.

Step 9: For each field defined in step 5, write the altitude values to the field using `GDwritefield`.

Step 10: Detach from the grid using `GDdetach`.

Step 11: Repeat steps 1 thru 10 for each required grid (i.e. different geographic regions and/or projections).

Step 12: Close the grid file.

As stated earlier, default services will be provided for the case where the third dimension is time or altitude. Exactly the same method can be used for any third dimension, such as band or camera. Generally, the dimension name should be of the form *DimensionMeaning\_n\_units*, where *n* is again an integer starting at 1, the *DimensionMeaning* would be what is required for subsetting and units (which is optional in the general case) is a string indicating an abbreviation for the units of the dimension.

There is an important point to make when creating three dimensional grids. To allow subsetting, when writing science data to the fields, the "layers" of the data must be written in a monotonically increasing or decreasing way. For example, if data is available at 10, 50 and 100 meters, either the data must be written such that the values at 10m are written before those at 50m which are before those at 100m, or the value at 100m is written before those at 50m which are before those at 10m. In this simple case, the data at 50m must not be written first or last. This could create a problem when the altitudinal distance does not map directly to a measurable

quantity, such as when data is present at the tropopause, the cloud base level or the adiabatic condensation level. In this case a slightly different approach is necessary, which is explained in Section 4.4. Note that the method of writing the file described above and the method described in Section 4.4 can be combined in a single file.

#### 4.2.1 Restrictions to Units

There are some restrictions to the units supplied with the dimension. These restrictions are necessary to allow subsetting to take place. One of the restrictions is that the field containing dimension values must contain either integer or real values; no string values are allowed. This is a limitation imposed by HDF-EOS.

In the case of subsetting by altitude, the units may be one of the following:

Pa, hPa, kPa, atm, meters, km, millibars, theta, sigma, Kelvin, Celsius, fathoms.

Some unit conversion will be performed during a subsetting request. Conversion will be performed between the following units:

meters and km

Pa, hPa, kPa, millibars and atm

Kelvin and Celsius

In the release B timeframe no other conversions will be performed due to the often complex relationship between them (for instance Pa will not be converted to meters), although later sophisticated client interfaces may allow more conversion to occur. These units were derived from the valids list for the ECS Collection level attributes AltitudeDistanceUnits and DepthDistanceUnits.

Similarly, if the third dimension in a field is time, this must be given in one of the following units:

century, year, month, day, hour, minute, second, millisecond, microsecond, sbt (spacecraft binary time)

Note care must be taken when using sbt, since there may be clock wrap-arounds, where the counter is reset to zero. In this case, to provide subsetting, the data should be placed in two grids, one prior to the clock wrap-around and one following (including the wrap around point).

In addition, the unit could be a date. However, to fit with the requirement that the dimension be monotonically increasing, only certain time and date formats are allowed. In this case the units should be replaced with one of the following abbreviations:

TAI93 : The number of continuous seconds since 12:00:00 on 1. Jan. 93

JD : Julian Date

MJD : Modified Julian Date

UTC : Date and time as YYYYMMDDHHMMSS.ssssss...

TAI : International atomic time

The use of the dates is convenient for the cases when, for example, the data is produced hourly, but spans more than one day. Note that numerical versions of time must be given in double precision.

### 4.3 Sample Code

Sample code is shown below. For simplicity, this example will construct only a single grid with three fields. Comments in the code should serve to clarify the structure and file organization. This example is written in C and for clarity, error checking is omitted.

```
#include <stdio.h>
#include <stdlib.h>
#include <hdf.h>
#include <mfhdf.h>
#include <HdfEosDef.h>

#define XGRIDSIZE 128
#define YGRIDSIZE 64

int main()
{
    /* Declare variables */
    int32 gridId;      /* HDF-EOS Grid Identifier */
    int32 fileId;      /* HDF-EOS File Identifier */
    int32 rtn;         /* General return from HDF-EOS functions */
    float64 ulCorner[2] = {-180000000.0, 90000000.0};    /* Upper left corner */
                                                         /* of grid in meters */
    float64 lrCorner[2] = {180000000.0, -90000000.0};    /* Lower right corner */
                                                         /* of grid in meters */
    float64 projParameters[13] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                                   0.0, 0.0, 0.0, 0.0, 0.0};
    short altValues1 = {100, 250, 500}; /* Altitude in hPa for one of the */
                                         /* dimensions */
    short altValues2 = {300, 500}; /* Altitude in hPa for one of the */

```

```

                                /* dimensions */
        /* Array for data values on a 128x64 grid with three */
        /* vertical values */
float temperatureData[XGRIDSIZE][ YGRIDSIZE][3];
        /* Array for data values on a 128x64 grid with two */
        /* vertical values */
int windspeedData[XGRIDSIZE][ YGRIDSIZE][2];
        /* Array for data values on a 128x64 grid with three */
        /* vertical values */
float percipitationData[XGRIDSIZE][ YGRIDSIZE][3];

/*
The sections following assume that the data arrays have already been filled by
the code.
*/
/* Open the grid file */
fileId = GDopen("someGridFile.dat", DFACC_CREATE);

/* Create the grid within the file */
gridId = GDcreate(fileId, "SampleGrid", XDIMSIZE, YDIMSIZE, ulCorner,
lrCorner);

/* Define the projection and origin for the data on the grid. In this case we
will assume a simple geographic projection centered at 0 degrees, with an
origin at the upper left corner and the GCTP default spherical earth model */
rtn = GDdefproj(gridId, GCTP_GEO, NULL, 19, projParameters);
rtn = GDdeforigin (gridId, HDFE_GD_UL);

/* Now define two dimensions (assume the precipitation and temperature data
share a dimension. */
rtn = GDdefdim(gridId, "altitude_1_hPa", 3);
rtn = GDdefdim(gridId, "altitude_2_hPa", 2);

/* Define a field for each of the data sets to be written */
rtn = GDdeffield (gridId, "percipitation", "altitude_1_hPa, YDim, XDim",
DFNT_FLOAT64, HDFE_NOMERGE);
rtn = GDdeffield (gridId, "temperature", "altitude_1_hPa, YDim, XDim",
DFNT_FLOAT64, HDFE_NOMERGE);

```

```

rtn = GDdefield (gridId, "windspeed", "altitude_2_hPa, YDim, XDim",
DFNT_INT32, HDFE_NOMERGE);

/* On the same grid define a new field for each z-dimension */
rtn = GDdefield (gridId, "altitude_1_hPa", "altitude_1_hPa", DFNT_INT16,
HDFE_NOMERGE);
rtn = GDdefield (gridId, "altitude_2_hPa", "altitude_2_hPa", DFNT_INT16,
HDFE_NOMERGE);

/* Detach and reattach to the grid */
rtn = GDdetach(gridId);
gridId = GDattach(fileId, "SampleGrid");

/* Write the science data to the appropriate fields */
rtn = GDwritefield(gridId, "percipitation", NULL, NULL, NULL, (VOIDP)
percipitationData);
rtn = GDwritefield(gridId, "windspeed", NULL, NULL, NULL, (VOIDP)
windspeedData);
rtn = GDwritefield(gridId, "temperature", NULL, NULL, NULL, (VOIDP)
temperatureData);

/* Now write actual altitude values to these dimension fields */
rtn = GDwritefield (gridId, "altitude_1_hPa", NULL, NULL, NULL,
(VOIDP)altValues1);
rtn = GDwritefield (gridId, "altitude_2_hPa", NULL, NULL, NULL,
(VOIDP)altValues2);

/* Detach from the grid */
rtn = GDdetach (gridId);

/* Close the file */
rtn = GDclose(fileId);
exit (0);
}

```

## 4.4 Writing Grid Data with String Type Altitude Values

As stated previously, some altitude values may not be numerical, and hence require a slightly different approach. Subsetting is still possible, with the restriction that it will not be possible to specify a range of altitude values (e.g. subset all of the data between 500 and 1000m). It will be possible to pass in an array of values, however (e.g. subset all the data at the tropopause and at TOA).

In this case, each field must be only two dimensional (YDim and XDim). There is no third dimension to each field. The grid name(s) in this case become important and must be assigned the parameter value previously applied to the field, i.e. given the same name as the one of the collection level metadata attributes Parameter of ECSVariableKeyword. This of course means that each grid can now only contain a single parameter. To allow for the case when a single parameter must appear on two different grids, each grid should be given a name of the form *ParameterName\_UniqueID*, where *UniqueID* can be anything the user wants to use to distinguish between the grids (for example a simple integer or an abbreviation indicating the projection or region). The fields within the grid are then named after the corresponding altitude levels. These altitude levels should appear in the collection level metadata attribute VerticalSpatialDomainValue.

In this case, to write the data the following steps should be followed:

Step 0: Open the file with GDopen.

Step 1: Create a grid within the file using GDcreate. The grid name must represent the parameter contained in the grid, which must correspond to the value of either Parameter or ECSVariableKeyword metadata attribute at the collection level, with an optional unique identifier (for example, "windspeed\_geo" or "relativeHumidity\_1").

Step 2: For each atmospheric level at which data is present on the grid, define a field using GDdeffield. The field name must map to the collection level attribute VerticalSpatialDomainValue. The field must only have the dimensions "XDim" and "YDim"

Step 3: Detach from the grid using GDdetach in order to fix the size and information relating to the grid.

Step 4: Reattach to the grid using GDattach.

Step 5: Write the science data to the appropriate field defined in step 2 using GDwritefield.

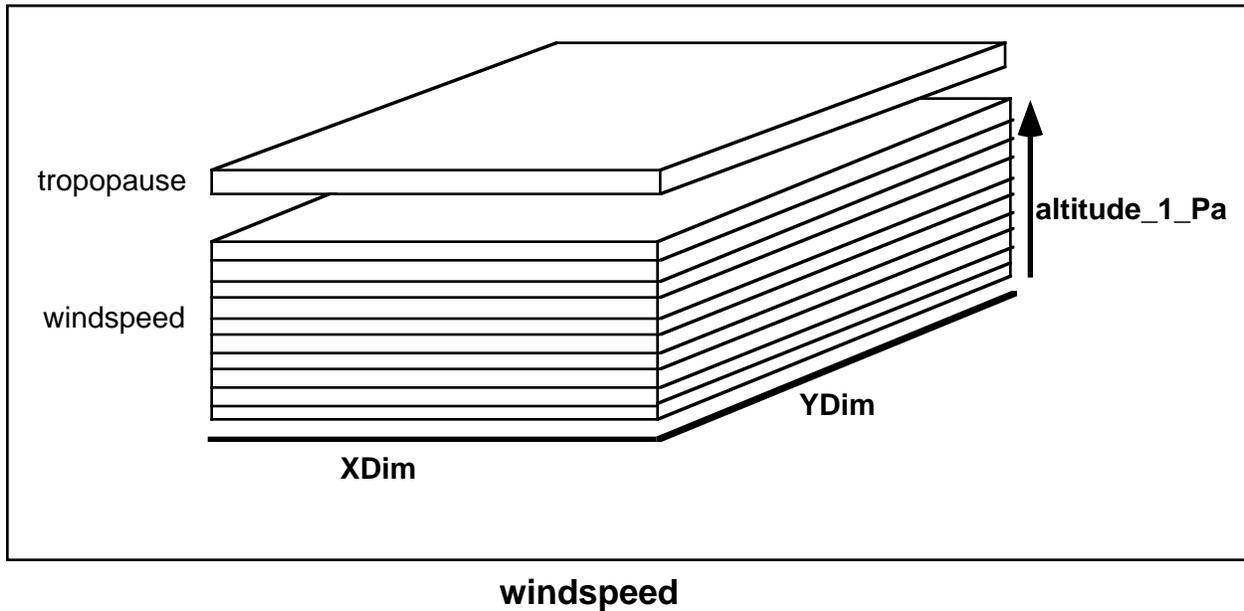
Step 6: Detach from the grid.

Step 7: Repeat steps 1 through 6 for each grid required.

Step 8: Close the file using GDclose.

As stated earlier, this method can be combined with using true three-dimensional fields (as described in Section 4.2). In fact, it is possible to combine the two methods in a single grid. The only limitation would be that all of the fields in the grid relate to the same parameter (since it is

important to ensure that the grid name will equate to parameter). This is shown in Figure 4-1 below. In this figure we show a single grid object. The grid object itself, named windspeed, has only two dimensions (XDim and YDim), and consists of two fields (windspeed and tropopause). The windspeed field has three dimensions (XDim, YDim and altitude\_1\_Pa) while the field tropopause only has two dimensions, the same as those of the grid itself.



**Figure 4-1. Combining Data with both Numerical and Textual Altitude Levels**

## 5. Writing Data Sets with One or More Parameters With Two Additional Dimensions

---

In this case it is assumed that the data to be written to the file not only lie on a suitable projection, but have both altitude and time components. For example, the data set may consist of a time sequence of vertical profiles for a number of parameters. Other dimension types aside from altitude and time can be supported, but the instrument teams must provide the dimension information and content to ECS to allow services to be performed.

There are two preferred methods of storing data in this format and there is somewhat of a trade off in deciding which approach to use. One method is to store the data in 4 dimensional fields (latitude, longitude, altitude and time) with each field representing a parameter, or to make each grid either altitude or time specific (so that all of the data in a grid contains information at the same altitude or time). In the latter case, each field will then be three dimensional and will again represent a parameter. The former method has the advantage of storage efficiency and coding simplicity, the latter has the advantage that higher level services will be more efficiently executed and simplicity of visualization and understanding for non-science users, but has significant disadvantages in terms of storage efficiency.

The latter method is preferred for storing this type of data, unless the number of grids required becomes too excessive (say more than 30) in which case the former method should be selected to reduce the file volume. This is only a rough guide in deciding between the two methods and the data provider should weigh the advantages and disadvantages of each method on a case by case basis.

### 5.1 Writing Data as Four Dimensional Fields

In this case it is assumed four dimensions represent latitude, longitude, altitude and time. The method of constructing a grid with this type of data is as follows:

Step 0: Open the grid file using GDopen.

Step 1: Create a grid using GDcreate. In this case the grid name has no significance.

Step 2: Define the projection and origin for the grid using GDdefproj and GDdeforigin.

Step 3: For each field (parameter) to be written to the grid, define the size (number of layers) the altitudinal and temporal dimensions using GDdefdim. One of these dimensions should be named *altitude\_n\_units* while the other should be *time\_n\_units*. These follow the same limitations defined in Section 4.2.1. More than one field may share a dimension.

Step 4: For each parameter, define a field using GDdeffield with dimensions "YDim", "XDim", *altitude\_n\_units*, and *time\_n\_units* (as defined in step 2). The name assigned

to a field should match the value of either Parameter or ECSVariableKeyword collection level metadata attributes.

Step 5: For each dimension created in step 3, define a field using the same name as the dimension. The dimension list should contain only the dimension associated with this field. For example, if in step 2 a dimension was defined named "altitude\_1\_m", then it is necessary to define a field with the call GDdeffield(gridID, "altitude\_1\_m", "altitude\_1\_m", DFNT\_INT16, HDFE\_NOMERGE).

Step 6: Detach from the grid using GDdetach in order to fix the size and information relating to the grid.

Step 7: Reattach to the grid using GDattach

Step 8: For each field write the appropriate science data using GDwritefield.

Step 9: Write the data values associated with the dimensions to the fields defined in step 6 using GDwritefield.

Step 10: Detach from the grid using GDdetach.

Step 11: Repeat steps 1 to 10 for each required grid.

Step 12: Close the file using GDclose.

### 5.1.1 Sample Code

Sample code for writing 4 dimensional data to a field is given below. This example is written in "C" and, so clarity, error checking is omitted. This code gives an example of how to write a single four dimensional field onto a grid with an altitude and time dimension. It is assumed that the data is on a simple geographic projection on the standard GCTP spherical earth model and covers the whole world.

```
#include <stdio.h>
#include <hdf.h>
#include <mfhdf.h>
#include <HdfEosDef.h>

#define TIME 31
#define ALTITUDE 5
#define XDIMSIZE 128
#define YDIMSIZE 64

int main()
{
float64 scienceData[TIME][ALTITUDE][XDIMSIZE][YDIMSIZE];
```

```

int32  gridId;
int32  fileId;
int32  rtn;
int16  altitudeValues[ALTITUDE] = {10, 50, 100, 250, 500};
float64 timeValues[TIME] = {19950101, 19950102, 19950103, 19950104, 19950105,
19950106, 19950107, 19950108, 19950109, 19950110, 19950111, 19950112,
19950113, 19950114, 19950115, 19950116, 19950117, 19950118, 19950119,
19950120, 19950121, 19950122, 19950123, 19950124, 19950126, 19950127,
19950128, 19950129, 19950130, 19950131};
float64 ulCorner[2] = {-180000000.0, 90000000.0};
float64 lrCorner[2] = {180000000.0, -90000000.0};
float64 projParameters[13] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                                0.0, 0.0, 0.0, 0.0, 0.0};

/* In the following, it is assumed that the array scienceData has already been
filled by the users software */

/* Open the file.  In this case it is assumed that the file already exists. */
fileId = GDopen("myFile.dat", DFACC_RDWR);

/* Create the grid */
gridId = GDcreate(fileId, "Daily CO2 Profiles", XDIMSIZE, YDIMSIZE, ulCorner,
lrCorner);

/* Set the projection and origin for the grid */
rtn = GDdefproj(gridId, GCTP_GEO, NULL, 19, projParameters);
rtn = GDdeforigin (gridId, HDFE_GD_UL);

/* Define the altitude and time dimensions */
rtn = GDdefdim(gridId, "altitude_1_m", ALTITUDE);
rtn = GDdefdim(gridId, "time_1_UTC", TIME);

/* Create the field */
rtn = GDdeffield(gridId, "CO2", "time_1_UTC, altitude_1_m, YDim, XDim",
DFNT_FLOAT64, HDFE_NOMERGE);

/* Define a field for each dimension in the grid */
rtn = GDdeffield(gridId, "altitude_1_m", "altitude_1_m", DFNT_INT16,
HDFE_NOMERGE);

```

```

rtn = GDdefield(gridId, "time_1_UTC", "time__1_UTC", DFNT_FLOAT64,
HDFE_NOMERGE);

/* Detach and reattach to the grid */
rtn = GDdetach(gridId);
gridId = GDattach(fileId, "Daily CO2 Profiles");

/* Write the science data to the field */
rtn = GDwritefield(gridId, "CO2", NULL, NULL, NULL, (VOIDP)scienceData);

/* Write the data values of the dimensions to the fields */
rtn = GDwritefield(gridId, "altitude_1_m", NULL, NULL, NULL,
(VOIDP)altitudeValues);
rtn = GDwritefield(gridId, "time_1_UTC", NULL, NULL, NULL, (VOIDP)timeValues);

/* Detach from the grid */
rtn = GDdetach(gridId);

/* Close the file */
rtn = GDclose(fileId);

exit(0);
}

```

## 5.2 Writing Data as a Series of Three Dimensional Fields

To create a grid file in this format it is first necessary to decide whether it is preferable to split the data in terms of altitude or time. To a large extent this may depend on how users are going to access the data. If it is anticipated that users will want the data time sliced, it is more convenient to make each grid a unique time. If, on the other hand, users are going to request data from a single atmospheric level, then making each grid reference a single altitude is more sensible. If either case is equally likely, or there is insufficient information regarding the user needs, then the selection is more a matter of choice. Clearly, splitting the smallest dimension makes the file more manageable.

Once the decision has been made regarding which dimension to split, the following steps should be performed to create the file.

Step 0: Open a file using GDopen.

Step 1: Create a grid using GDcreate. The name of the grid should be of the form "time\_n\_units:value" or "altitude\_n\_units:value". The units have the same restrictions given in Section 4.1.2. Each grid created MUST have a unique name.

Step 2: Define the projection and origin for the grid using GDdefproj and GDdeforigin.

Step 3: Define each field (parameter) define a dimension named either "altitude\_n\_units" or "time\_n\_units", as applicable, using GDdefdim. The size of the field should be the number of levels in the data (either the number of altitude levels or time slices). Note that more than one field can share a dimension.

Step 4: For each parameter create a field on the grid using GDdeffield. This field name should be the same as the value for either ECSVariableKeyword or ParameterName collection level metadata attributes. The dimensions for each field should be one of those defined in step 3 plus XDim and YDim.

Step 5: For each dimension defined in step 3, define a field using GDdefdim of the same name and size (e.g. GDdefdim(gridId, "altitude\_1\_hPa", "altitude\_1\_hPa", DFNT\_INT16, HDFE\_NOMERGE))

Step 6: For each parameter, write the data to the appropriate field defined in step 4 using GDwritefield.

Step 7: Detach from the grid using GDdetach in order to fix the size and information relating to the grid.

Step 8: Reattach to the grid using GDattach.

Step 9: Write the values associated with each dimension to the fields created in step 6 using GDwritefield.

Step 10: Detach from the grid using GDdetach.

Step 11: Repeat steps 1 to 10 for each altitude or time slice, and for each grid required (differing projections and/or geolocations).

Step 12: Close the grid file using GDclose.

### 5.2.1 Sample Code

The sample code below is for the same case as that given in 5.1.1. It is assumed that the data is split in altitude (this being the smallest dimension). Thus there will be one grid for each altitude level each containing a single field. Again, for simplicity, error checking is omitted.

```
#include <stdio.h>
#include <hdf.h>
#include <mfhdf.h>
#include <HdfEosDef.h>
```

```

#define TIME 31
#define ALTITUDE 5
#define XDIMSIZE 128
#define YDIMSIZE 64

int main()
{
float64 scienceData[ALTITUDE][TIME][XDIMSIZE][YDIMSIZE];
int32  gridId;
int32  fileId;
int32  rtn;
int16  altitudeValues[ALTITUDE] = {10, 50, 100, 250, 500};
float64 timeValues[TIME] = {19950101, 19950102, 19950103, 19950104, 19950105,
19950106, 19950107, 19950108, 19950109, 19950110, 19950111, 19950112,
19950113, 19950114, 19950115, 19950116, 19950117, 19950118, 19950119,
19950120, 19950121, 19950122, 19950123, 19950124, 19950126, 19950127,
19950128, 19950129, 19950130, 19950131};
float64 ulCorner[2] = {-180000000.0, 900000000.0};
float64 lrCorner[2] = {180000000.0, -900000000.0};
float64 projParameters[13] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
char gridName[50];

/* In the following, it is assumed that the array scienceData has already been
filled by the users software */
/* Open the file. In this case it is assumed that the file already exists. */
fileId = GDopen("myFile.dat", DFACC_RDWR);

/* Loop through for each altitude value */
for (i=0; i<ALTITUDE; i++)
{
    /* Create a grid for the altitude level */
    sprintf(gridName, "altitude_1_m:%d", altitudeValue[i]);
    gridId = GDcreate(fileId, gridName, XDIMSIZE, YDIMSIZE, ulCorner,
lrCorner);

    /* Define the projection and origin for the grid */

```

```

rtn = GDdefproj(gridId, GCTP_GEO, NULL, 19, projParameters);
rtn = GDdeforigin (gridId, HDFE_GD_UL);

/* Create the temporal dimension for the field(s) to be written */
/* to the grid. Since each grid has a unique dimension, the name */
/* need not be different for each grid */
rtn = GDdefdim (gridId, "time_1_UTC", TIME);

/* Create a field to contain the dimension values */
rtn = GDdeffield(gridId, "time_1_UTC", "time_1_UTC", DFNT_FLOAT64,
HDFE_NOMERGE);

/* Create the field on the grid */
rtn = GDdeffield(gridId, "CO2", "time_1_UTC, YDim, XDim", DFNT_FLOAT64,
HDFE_NOMERGE);

/* Detach and reattach to the grid */
rtn = GDdetach(gridId);
rtn = GDattach(fileId, gridName);

/* Write the data to the grid */
rtn = GDwritefield(gridId, "CO2", NULL, NULL, NULL,
(VOIDP)scienceData[i]);

/* Write the dimension values to the field */
rtn = GDwritefield(gridId, "time_1_UTC", NULL, NULL, NULL,
(VOIDP)timeValues);

/* Detach from the grid */
rtn = GDdetach(gridId);
} /* End of for loop for altitude values */

/* Close the grid file */
rtn = GDclose(fileId);
exit(0);
}

```

This page intentionally left blank

# Abbreviations and Acronyms

---

atm	atmospheres (unit of pressure measurement)
ECS	EOSDIS Core System
ESDT	Earth Science Data Type
HDF	Hierarchical Data Format
hPa	hectoPascal (unit of pressure measurement)
JD	Julian Date
kPa	kiloPascal (unit of pressure measurement)
MJD	Modified Julian Date
Pa	Pascal (unit of pressure measurement)
sbt	satellite binary time
UTC	Universal Time Coordinates