HDF4 to CF

A practical approach

MuQun Yang

January 8th, 2015

Version 1.0 draft

The HDF Group

Some popular Earth Science visualization and analysis tools cannot visualize NASA HDF4 and HDF-EOS2 files because these tools follow Climate and Forecast (CF) conventions to access data whereas many NASA HDF4 and HDF-EOS2 files don't follow the CF conventions. In order to help users access NASA HDF data by these tools, The HDF Group adopts a practical approach to translate HDF4 and HDF-EOS2 file and object metadata information to follow CF conventions. This document provides the mapping specification and reasons from HDF4 and HDF-EOS2 to CF. Based on the information presented in this document, The HDF Group also implemented two software packages: HDF4 to CF (H4toCF) conversion toolkit and the CF option of the HDF4 OPeNDAP handler.

Contents

1. Introduction

1.1. Background

Thousands of NASA HDF4 [1] and HDF-EOS2 [2] products are widely used in scientific research, high education, weather forecasts and decision making process for policy makers. However, a significant amount of end users find that they cannot access and visualize many of these products in their favorite tools. These tools include free scripting language tools such as FERRET and GrADs and the “click and go” Java tools such as Panoply and IDV. One key reason for the data access failures is that the tools mostly follow the Climate and Forecast (CF) conventions [3] to access the data whereas most NASA HDF4 and HDF-EOS2 products do not follow CF conventions.

The CF conventions are metadata conventions for earth science data, intended to promote the processing and sharing of files created with the NetCDF API, but most of its ideas relate to metadata design in general, not specifically to netCDF. The CF evolves from Cooperative Ocean/Atmosphere Research Data Service (COARDS), which has a similar purpose and is also widely used. CF is backward-compatible with COARDS.

1.2. Motivation

In order to make NASA HDF4 and HDF-EOS2 products accessed by the tools that follow CF conventions (CF tools hereafter), we translate key information required by CF tools to access these products into the format that these tools can understand. Especially we aim to have Panoply and IDV users to be able to simply click and visualize the HDF data contents in a user-friendly geographic map. Rather than covering *all* NASA HDF4 and HDF-EOS2 products we aim to cover *most* popular NASA HDF4 and HDF-EOS2 products requested from NASA data centers and our users.

1.3. Approach

Since our goal is to make CF tools correctly visualize NASA HDF4 and HDF-EOS2 products, our approach is practical rather than theoretical. This can be identified on the following areas:

1) Whether the mapping is CF compliant

The mapping from HDF4 to CF is not fully CF compliant. We focus on the areas that are keys for the CF tools to access and visualize the data. There are also areas that the CF tools accept variations of CF conventions and it will cause confusion if the CF conventions are exactly followed. For those areas, we simply ensure that the CF tools can access the HDF4 and HDF-EOS2 data correctly.

2) How to handle the key information not addressed by the CF conventions

Some information, mostly concentrated on how to handle coordinates, is necessary for CF tools to visualize the data. However, no conventions on how to share the information are provided in the current CF conventions. For such information, we try to provide the mapping for CF tools to visualize the data according to our best knowledge about an individual product.

3) How to map the key missing information to CF

We observed that for some products, the CF information for tools to correctly visualize the data is either hidden inside the HDF file or only available in the corresponding product specification documents. To handle such a case, our first step is to research the information. This step is often done by thoroughly evaluating the file structure, reading the product documents and sometimes by communicating with the corresponding NASA data center developers and even data producers. Then based on our best knowledge resulting from the research, this information will be translated to the form by following the CF conventions so that the CF tools can understand and generate a correct plot. The missing information mainly concentrates on two areas: coordinates and the corresponding dimension information and the information on how to pack the data in order to reduce the file size.

4) How to handle rarely used HDF4 and HDF-EOS2 objects in NASA products

We don't provide the mapping specification for the HDF4 and HDF-EOS2 objects rarely used in NASA HDF4 and HDF-EOS2 products. These objects include HDF-EOS2 point objects, HDF4 image objects, palette objects and annotations.

5) How to verify if the mapping of HDF-EOS2 and HDF4 is sound

We implemented two software packages based on the information in this document. They are the CF option of the HDF4 OPeNDAP handler [4] and H4CF conversion toolkit [5]. NASA sample files are provided in the referred web pages. Readers can check the converted netCDF files or DAP2 output with CF tools such as Panoply.

1.4. About the rest of the document

Section 2 provides the general mapping information from HDF4 and HDF-EOS2 to CF. Section 3 provides the dimension mapping information. Section 4 focuses on the coordinate mapping. Section 5 addresses the translation of HDF4 and HDF-EOS2 metadata information to CF attributes. We also provide four Appendices. Appendix A lists the NASA HDF4 products of which latitudes and longitudes need to be retrieved with special handlings to follow CF conventions. Appendix B provides the special handling of HDF4 andHDF-EOS2 to DAP2. Appendix C provides information of some netCDF-style tools that can be used to supplement information required by CF tools to access and visualize the data. Appendix D lists some NASA HDF4 and HDF-EOS2 products that don't follow the CF data packing rule.

2. General Mapping

In general, CF communities provide conventions for the variables, the associated dimensions and attributes of the variables and the global attributes of the data file. A variable is usually a data array to describe the quantity of a physical variable. A dimension of a variable gives the dimensional information of the array that describes the variable. An attribute of a variable provides the metadata information about the variable. A file attribute provides the metadata information about the whole file. In this section, we will provide the mapping from HDF4 and HDF-EOS2 to general CF variables and file attributes. The dimension and the coordinate handling will be discussed in section 3 and section 4 correspondingly.

2.1. Object and Attribute Mapping

2.1.1. HDF-EOS2

HDF-EOS2 has swath, grid and point objects. Each object consists of fields and attributes. Each field can also have attributes. Since the NASA HDF-EOS2 point files are very rare, we decide not to provide any mapping specification from HDF-EOS2 point objects to CF. HDF4 OPeNDAP handler and HDF4 to CF conversion toolkit will ignore the mapping of HDF-EOS2 point objects. Since there is no equivalent CF counterpart for an HDF-EOS2 object attribute, we map it to a CF file attribute to keep the information. Swath and grid fields are mapped to CF variables. Field attributes are mapped to CF variable attributes.

Table 1 summarizes the mapping of the HDF-EOS2 to CF.

Table 1. HDF-EOS2 Field and Attribute to CF

HDF-EOS2 Object	**HDF-EOS2 Field/Attribute**	**CF Variable/Attribute**
Swath	Data Field	Variable
	Geolocation Field	Variable
	Field Attribute*	Variable Attribute
	Attribute	File Attribute
Grid	Data Field	Variable
	Field Attribute*	Variable Attribute
	Attribute	File Attribute
Point	The mapping specification is not available	

* Applications can only add one attribute _*Fillvalue* for swath or grid fields via HDF-EOS2 APIs.

2.1.2. HDF4

HDF4 has vgroup, vdata, SDS, image, palette and annotation objects. Since we find very few NASA HDF4 objects that include image, palette and annotation, we decide not to provide any mapping specification for these objects. The HDF4 OPeNDAP handler and HDF4 to CF conversion toolkit will ignore the mapping of HDF4 image, palette and annotation.

An HDF4's SDS is like a data array that stores scientific data numbers. So an HDF4's SDS is equivalent to a CF variable. An attribute obtained by calling the HDF4's SD APIs(SD Attribute) is equivalent to a CF file attribute. Naturally an SD attribute is mapped to a CF file attribute, an SDS is mapped to a CF variable and an SDS's attribute is mapped to a CF variable attribute.

An HDF4 vdata is like a table that can have several fields. An HDF4 vgroup is logically like a container that consists of HDF4 objects including another vgroup. Vgroup, vdata and vdata fields all can have attributes. However, there are no equivalent CF objects for vdata and vgroup. To map vdata information to CF, we map a vdata's field to a CF variable. Attributes of a vdata field is mapped to attributes of the corresponding CF variable. Attributes of vdata and vgroup are mapped to CF file attributes.

Table 2 summarizes the mapping of HDF4 object and attribute to CF.

Table 2. HDF4 Object and Attribute to CF

HDF4 Object/Attribute	**CF Object/Attribute**
SD Attribute	File Attribute
SDS	Variable
SDS Attribute	Variable Attribute
Vdata	No equivalent CF Object
Vdata Attribute	File Attribute
Vdata Field	Variable
Vdata Field Attribute	Variable Attribute
Vgroup	No equivalent CF Object
Vgroup Attribute	File Attribute
Image	The mapping specification is not available.
Palette	
Annotation	

2.1.3. HDF-EOS2 Hybrid

HDF-EOS2 objects and attributes can be accessed by any HDF4 APIs. So HDF-EOS2 data producers can use HDF4 APIs to add HDF4 objects or attributes to an HDF-EOS2 file. These added objects or attributes cannot be accessed by HDF-EOS2 APIs. However, users still need to access the original HDF-EOS2 objects via HDF-EOS2 APIs to correctly retrieve the geo-location information of these objects. The HDF-EOS2 files that contain information added by HDF4 APIs are called HDF-EOS2 hybrid files in this document. Both HDF-EOS2 and HDF4 APIs are necessary to access the HDF-EOS2 hybrid files. Many MODIS and MISR NASA HDF-EOS2 products are HDF-EOS2 hybrid files. For an HDF-EOS2 hybrid file, the HDF-EOS2 objects are mapped by following Table 1. The additional HDF4 objects and the added attributes are mapped according to Table 2.

2.2. Name Mapping

According to the CF conventions, variable, dimension and attribute names should begin with a letter and be composed of letters, digits, and underscores ('_'). However, HDF4 object and attribute names can contain any non-alphanumeric characters.

To follow CF conventions, the CF naming rule we follow is to change any non-alphanumeric character inside an HDF4 object or attribute name to an underscore character. It is very possible that an HDF4 object name starts with a non-alphanumeric character and it is changed to an underscore according to the CF naming rule. However, one may note that an underscore is not allowed to be the first character in a CF name. So the above case violates the CF conventions. In reality, the CF tools accept the name that starts with an underscore as a valid object name and generate the correct plot. So we still follow this naming rule regardless of the violation of the CF conventions for this special case. This is one example of the practical approach mentioned in the section 1.3.

2.2.1. HDF-EOS2

One HDF-EOS2 file can have multiple HDF-EOS2 objects such as grids or swaths. Each HDF-EOS2 object can have attributes and fields. Each field can have attributes.

An object field name can be shared among different objects. For example, it is valid that an HDF-EOS2 grid named *grid1* has a field named *pressure* and an HDF-EOS2 grid named *grid2* has the same field named *pressure*. However, the name of a CF variable is a unique key to identify this variable among other variables. So to avoid obvious name clashing for the CF variable names mapped from HDF-EOS2 fields under different HDF-EOS2 objects, the CF variable name consists of an HDF-EOS2 object name applied with the CF naming rule, an underscore character and an HDF-EOS2 field name applied with the CF naming rule. For an HDF-EOS2 file that only has a single HDF-EOS2 object, it is not necessary to distinguish among different objects, the HDF-EOS2 object name is omitted in the CF variable name. One should be aware that it is also valid to keep an HDF-EOS2 object name inside the mapped CF variable name even though there is only one HDF-EOS2 object name inside the HDF-EOS2 file. However, we choose the current way for a practical reason. Long CF variable names may deteriorate the performance for some CF tools to access the data. If saving the object name information is desirable, the application that implements the mapping of HDF-EOS2 to CF can provide an attribute to store the object name.

Since a field attribute is associated with the corresponding field and the field name is mapped to a unique CF variable name, the corresponding attribute name of a CF variable is just the field attribute name applied with the CF naming rule.

As described in section 2.1.1, an HDF-EOS2 object attribute is mapped to a CF file attribute. To distinguish the different object attributes that may share the same name, we form the corresponding CF file attribute name by concatenating the string *HDFEOS*, the HDF-EOS2 object type name and the HDF-EOS2 object name with its attribute name. An underscore character is used to connect the different parts. For example, an HDF-EOS2 grid named *grid1* has an attribute named *gattribute*. The mapped CF file attribute name is *HDFEOS_grid_grid1_gattribute*.

Table 3 and 4 illustrate how HDF-EOS2 names are mapped to CF by examples.

Table 3. HDF-EOS2 Field and Field Attribute Name to CF

Number of HDF-EOS2 objects	HDF-EOS2 Object Name	HDF-EOS2 Field Name	CF variable Name	HDF-EOS2 Field Attribute Name	CF Variable Attribute Name
Multiple	*gd?or_sw*	*gs*field*	*gd_or_sw_gs_field*	*gs-fieldattr*	*gs_fieldattr*
Single	*gd?or_sw*	*gs*field*	*gs_field*	*gs-fieldattr*	*gs_fieldattr*

Table 4. HDF-EOS2 Grid and Swath Attribute Name to CF

HDF-EOS2 Object Type	HDF-EOS2 Object Name	Attribute Name	CF File Attribute Name
Grid	*Mygrid*	*gattr*	*HDFEOS_grid_mygrid_gattr*
Swath	*Myswath*	*Sattr*	*HDFEOS_swath_myswath_sattr*

Since there may be many attributes in an HDF-EOS2 grid or swath, mapping all grid and swath attributes to global attributes may be difficult for CF users to distinguish different attribute information. The potential long attribute names listed in the table 4 make the situation even worse. So individual data format or protocol that represent the CF output of the vdata and vgroup attributes may *not* follow the naming conventions listed in the table 4. For example, DAP2 DAS representation of grid or swath attributes can simply hold all the attributes in a grid or a swath in a DAS attribute container. Under such a case, the application can choose to simply map grid or swath CF attribute names to be the corresponding CF attribute names that apply the CF naming rule. According to our current understanding, this will not affect the CF tools to visualize the data.

2.2.2. HDF4

An HDF4 SDS object can be attached to an HDF4 vgroup. Since individual SDS objects under different HDF4 vgroups may share the same name, to avoid the obvious name clashing and to keep the group hierarchy of the SDS that is attached to a vgroup, the SDS name mapped to CF should include all the ancestral vgroup paths. The mapping procedure is as follows: first, an intermediate CF name of the SDS is generated by using the underscore character to connect the parental vgroup path with the child vgroup path and the last vgroup path with the SDS name retrieved via the SDS APIs; then the CF name of this SDS object is generated by applying the CF naming rule to this intermediate name.

For an SDS object not attached to any HDF4 vgroups, the CF name of this SDS is generated by simply applying the CF naming rule to the SDS names retrieved via the SDS APIs.

Like an SDS, an HDF4 vdata can also attach to an HDF4 vgroup. Although a vdata is not mapped to a CF variable, a vdata field is mapped to a CF variable. Also like the handling of an SDS mapping, to avoid the obvious name clashing and to keep the group hierarchical information of vdata field names, the CF name of a vdata field includes the path of the vgroup that the corresponding vdata is attached to. Moreover, to ensure the CF users understand that this CF variable is mapped from an HDF4 vdata field, string *vdata* is prefixed before the vdata path and the string *vdf* (stands for vdata field) is used to connect the vdata path and the vdata field name. An underscore is used to connect the adjacent components.

Since an SDS attribute is associated with the SDS and a CF variable name is the unique key to identify a CF variable, the attribute name of the corresponding CF variable mapped from this SDS is obtained by simply applying the CF naming rule to the SDS attribute name. The same approach also applies to the mapping of a vdata field attribute.

Since an HDF4's SD attribute is equivalent to a CF file attribute, the CF name of the corresponding SD attribute is simply obtained by applying the CF naming rule to the SD attribute name.

An HDF4's vdata attribute is mapped to a CF file attribute. To help the CF users identify if the attribute is mapped from an HDF4 vdata attribute, the corresponding CF file attribute name is generated by concatenating the string *Vdata*, the full path of the vdata , the string *Attr* and the vdata attribute name. An underscore is used to connect the adjacent components.

An HDF4's vgroup attribute is also mapped to a CF file attribute. For the similar reason like the generation of the CF name of a vdata attribute, the corresponding CF name of a vgroup attribute is generated by concatenating the string *Vgroup*, the full path of the vgroup, the string A*ttr* and the vgroup attribute name. An underscore is used to connect the adjacent components.

Table 5-7 illustrate how HDF4 object and attribute names are mapped to CF by examples.

Table 5. HDF4 SD, Vgroup and Vdata Attribute Name to CF

HDF4 Object	**Vgroup Path**	**HDF4 Object Name**	**HDF4 Attribute Name**	**CF File Attribute Name**
SD	*N/A*	*N/A*	*SD#attr*	*SD_attr*
Vdata	*vg*	*vdata?1*	*vattr*	*Vdata_vg_vdata_1_Attr_vattr*
Vgroup	*vg*	*vg1*	*vgattr*	*Vgroup_vg_vg1_Attr_vgattr*

*If SDS, vdata or vgroup don't attach to any vgroups, simply map their names by following the CF naming rule.

Table 6. HDF4 SDS Object and Attribute Name to CF

Vgroup Path for SDS	**SDS name**	**CF variable name**	**SDS attribute**	**CF Variable Attribute Name**
/vg	*sds*	*_vg_sds*	*sds_attr*	*sds_attr*

*If SDS, vdata or vgroup don't attach to any vgroups, simply map their own names.
*For the supported NASA HDF4 products, the first slash('/') of the vgroup path is ignored. The whole group path may be ignored for some products mainly due to the performance and restriction of some CF tools.

Table 7. HDF4 Vdata Field and Vdata Field Attribute Name to CF

Vgroup Path for vdata	**Vdata Name**	**Vdata Field Name**	**CF Variable Name**	**Vdata Field attribute**	**CF Variable Attribute Name**
vg	*myvdata*	*vdfield*	*Vdata_vg_myvdata_vdf_vdfield*	*VF?attr*	*VF_attr*

Since there may be many HDF4 vgroup and vdata objects in an HDF4 file and each vgroup or vdata object can potentially have many HDF4 attributes, mapping all vgroup and vdata attributes to global attributes may be difficult for CF users to distinguish different attribute information. The potential long attribute names listed in the table 5 and table 7 make the situation even worse. So individual data format or protocol that represent the CF output of the vdata and vgroup attributes may *not* follow the naming conventions listed in the table 5 and table 7. For example, DAP2 DAS representation of vdata or vgroup attributes can simply hold all the attributes in a vdata or a vgroup in a DAS attribute container. Under such a case, the application can choose to simply map vdata or vgroup CF attribute names to be the corresponding CF attribute names that apply the CF naming rule. According to our current understanding, this will not affect the CF tools to visualize the data.

2.2.3. HDF-EOS2 hybrid

We find that some added SDS objects share the same names with HDF-EOS2 grid or swath fields in NASA HDF-EOS2 hybrid products. To avoid the massive name clashing and to distinguish the non-HDFEOS2 objects from the HDF-EOS2 objects, we generate the CF name of an added SDS object by appending a string *NONEOS* to the original SDS name. For example, a field name in an HDF-EOS2 grid is *Temperature*. The data producer decides to add another SDS *Temperature* to this file. Both the HDF-EOS2 field and the added SDS are mapped to CF variables sharing the same name *Temperature*. The CF name of the SDS added by the HDF4 APIs *Temperature* is changed to *Temperature_NONEOS*.

The mapping of other objects and attributes follows section 2.2.1 and 2.2.2.

2.2.4. Handle name clashing

A CF variable name is a unique key to identify this variable. The attribute name of a CF variable is also a unique key to identify this attribute in the variable. File attributes also use attribute names to distinguish this attribute from other file attributes. However, HDF4 allows different objects to share the same name. Although by adding the group path, obvious name clashings may be avoided, still a CF variable or a CF attribute mapped from HDF4 or HDF-EOS2 may have name clashings with other CF variables or other CF attributes. A certain rule needs to be applied to make the clashed names unique so that *any* CF variable or attribute can be identified.

The rule to handle the name clashing is illustrated on the following:

If one CF variable shares the same name with another CF variable, the way we resolve the name clashing is to add an underscore and the index number (1,2,3 etc.) at the end of the second CF variable name. For example, two CF variables share the same name *temperature*. The first picked variable name should still be *temperature*. The second variable name becomes *temperature_1*. This rule will not apply to the added SDS objects in an HDF-EOS2 hybrid file. The handling of the name clashing for added SDS objects in an HDF-EOS2 hybrid file was described in section 2.2.3.

Optionally, when a name clashing occurs, one can add an attribute to store the original variable name if keeping the original variable name information is desirable.

3. Dimension Mapping

A CF variable is usually represented as a data array. According to CF conventions, each dimension of the array is required to present as a CF dimension. Each CF dimension consists of two components: dimension name and dimension length. The length of a dimension is simply the number of elements of the corresponding array dimension. The name of a dimension should be provided by the applications or generated by the data format libraries. CF dimensions need to be provided in order for the tools to visualize the data products properly. Section 3.1 provides common mapping information from HDF4 and HDF-EOS2 dimensions to CF. Section 3.2 provides the specific mapping information related to the HDF4 and HDF-EOS2 dimensions.

3.1. Common Mapping

3.1.1. Name Mapping

The dimension names should also follow CF's naming conventions. So the CF naming rule discussed in section 2 should be applied to all the dimension names mapped from HDF4 or HDF-EOS2. However, no vgroup path needs to be considered since like an attribute, a dimension is attached to a variable.

3.1.2. COARDS Requirement

COARDS conventions require the dimension name to be the same as the coordinate variable name when the coordinate variable is stored as one dimensional array. Since CF tools still follow COARDS conventions to identify coordinate variables, so if applicable, we follow the COARDS conventions when generating the dimension name. For example, the values of coordinate variables *lat* and *lon* are stored in one dimensional array. The dimension name of *lat* is *YDim* and the dimension name of *lon* is *XDim*. To follow COARDS, the dimension name *XDim* for all the dimensions that have such a name is changed to *lat*. The dimension name *YDim* is changed to *lon*. Since CF is backward compatible with COARDS, CF conventions are not violated.

3.1.3. Handle name clashing

Like a CF variable, a CF dimension name is also the key to identify this dimension among all dimensions. It is allowable for HDF4 and HDF-EOS2 to have duplicate dimension names associated with a variable. So the same name clashing rule described in section 2.2.4 is also applied to the dimension names.

3.2. Special mapping

3.2.1. HDF-EOS2

HDF-EOS2 swath and grid require the data producer to provide the same dimension information as the CF conventions require. So the dimension mapping from HDF-EOS2 to CF is one to one mapping.

3.2.2. HDF4

In HDF4, data values of an SDS or a vdata field are also stored in an array. The length of each dimension is specified. However, HDF4 does not require the data producers to provide dimension names.

For each dimension in an SDS object, if an application does not provide a name, HDF4 library will assign a unique dimension name. This library-generated dimension name always has the form *fakeDim1, fakeDim2 etc*. Each library-generated dimension name is different than other library generated dimension names even though they may share the same dimension length. If there are many SDS objects in an HDF4 file and the dimension names of these SDS objects are not specified, one may observe many HDF4 library generated dimension names in the HDF4 file. These library generated dimension names often share the same dimension length. These fake dimension names are not useful to understand the physical meanings of the data. It may even be difficult for users to figure out why so many HDF4 library-generated fake dimension names are assigned to the same dimension length. So to avoid such confusions and still to follow the CF conventions, we condense the fake dimensions that share the same dimension length to one dimension.

For a field in a vdata object, neither an application nor the HDF4 library can add any dimension names to the field. However, the vdata record and the vdata field order are equivalent to dimension lengths. To follow the CF conventions for the variable mapped from the vdata field, dimension names are specified in the following form:

The corresponding dimension name for the vdata record is

*VDFDim0_vdata_*<this vdata name>*_vdf_*<this vdata field name>

If the vdata field order is greater than 1, the corresponding vdata field is equivalent to a two-dimensional array variable. The order of the vdata field is the length of the fastest changing dimension. The corresponding dimension name for this dimension is

*VDFDim1_vdata_*<this vdata name>*_vdf_*<this vdata field name>

The prefixes *VDFDim0*, *VDFDim1, vdata* and *vdf* identify that this dimension is mapped from vdata.

For the illustration purpose, the vdata name and vdata field name listed in the above representations are embraced by the <> bracket.

3.2.3. HDF-EOS2 hybrid

The dimension mapping follows the mapping of HDF4 and HDF-EOS2.

4. Coordinate Mapping

Coordinate conventions are crucial for Earth Science applications to visualize and analyze the data. However, the coordinate conventions in CF are, to some degree, fundamentally different than the conventions specified in HDF4 and HDF-EOS2. Therefore, we dedicate this whole section to describe how the coordinates in HDF4 and HDF-EOS2 are mapped to CF.

4.1. Latitude and Longitude

In this subsection, we address the mapping of the most important coordinates for Earth Sciences: latitude and longitude.

4.1.1. HDF-EOS2

4.1.1.1. Grid

HDF-EOS2 Grid applications don't need to provide latitude and longitude values at each data point. Instead, applications can use the HDF-EOS2 APIs to store several parameters such as the number of elements, the grid boundary values and projection information for the latitude and longitude. However, CF requires that the latitude and longitude values are explicitly represented as data arrays. Latitude and longitude values in an HDF-EOS2 grid need to be retrieved to allow CF tools to identify the locations of data values.

4.1.1.1.1. *General Mapping*

An HDF-EOS2 API can be used to retrieve latitude and longitude values at each grid point. CF two-dimensional latitude and longitude coordinate variables are then defined to store the retrieved values. In general, the *coordinate* attributes described in section 5.1 should be created for each field that has values at every location described by latitude and longitude. The only exception is for the case described in section 4.1.1.2.

4.1.1.1.2. *Dimension Reducing*

For some grid projections (geographic etc.), the latitude value is the same at each data point along one horizontal grid line. The longitude value is the same at each data point along the grid line perpendicular to the equal-latitude grid line. For such a projection, the two-dimensional data arrays that store latitude and longitude are condensed to two orthogonal one-dimensional data arrays. Two one-dimensional latitude and longitude coordinate variables can be then defined. Mapping the latitude and longitude in this way can save the disk space and make CF tools quickly pin down the grid location and generate the plot.

4.1.1.1.3. *Lambert Azimuthal Equal-Area Projection*

Some NASA HDF-EOS2 grid products use Lambert azimuthal equal-area projection to store the grid data. We observed that the longitude values retrieved by using the HDF-EOS2 API contain infinite number around the North Pole and the South Pole. This causes the CF tools fail to open the file. After communicating with the data provider of these data products and the HDF-EOS2 library developer, we found out that having the infinite numbers inside the longitude in the North Pole or the South Pole for this projection is not an HDF-EOS2 library bug. We also found out that it is acceptable to use the nearest longitude values around the Poles to replace the infinite number for the visualization purpose. So for this projection, we use the nearest neighbor method to interpolate the longitude around the South Pole and the North Pole.

4.1.1.1.4. *Space Oblique Mercator projection*

Some NASA HDF-EOS2 grid products also use the space oblique mercator projection to store the grid data. The values of latitude and longitude retrieved by the HDF-EOS2 API for this type of grid have to be stored in three-dimensional arrays. Currently CF conventions only cover the cases when the latitude and longitude values are stored as one or two dimensional arrays. One way to make this type of grid follow CF conventions is to transform a three-dimensional array to a two-dimensional array. While it may not be difficult to transform latitude and longitude, to make the corresponding physical variables visualized by CF tools, all the data representation of these variables must also be transformed from three-dimensional arrays to two-dimensional arrays. This may not be what existing end-users desire. So currently we simply define three-dimensional latitude and longitude variables to store the retrieved latitude and longitude values. The CF tools can open the file but fail to visualize the physical variables. This is a special case that we hope that CF communities can address in the future.

4.1.1.2. Swath

Generally an HDF-EOS2 swath provides two-dimensional latitude and longitude fields. Those fields are naturally mapped to CF's two-dimensional latitude and longitude coordinate variables. The *coordinate* attributes described in section 5.1 should be created for each field that has values at every location described by latitude and longitude. For swath that uses the HDF-EOS2 dimension map technique, the latitude and longitude values will be interpolated according to HDF-EOS2's dimension map formula. The parameters required by the dimension map can be retrieved by using the HDF-EOS2 APIs. The dimension sizes of the CF latitude and longitude variables should be adjusted according to the information derived by the dimension map parameters. We also encounter a small amount of HDF-EOS2 swath products that store the latitude and longitude fields in one-dimensional arrays. Our approach is still to map latitude and longitude to CF variables and ensure the *units* attributes of latitude and longitude to follow the CF conventions. Currently CF tools may not be able to visualize these swath products. This is another special case that we hope that CF communities can address in the future.

4.1.2. HDF4

For an NASA HDF4 file, there is no generic model to map latitude and longitude to CF. Different products may need to be addressed individually. We observe that NASA data centers are still updating the HDF4 data products. Furthermore, we also observe that the way to retrieve latitude and longitude information may be different in the new versions of some HDF4 data products. This document only addresses the versions of the NASA HDF4 products we have investigated. However, one should also note that although the new products may have different layouts of latitude and longitude; the mapping principle basically stays the same.

The main NASA HDF4 products we currently provide latitude and longitude mapping information are Tropical Rainfall Measuring Mission (TRMM) products, Clouds and the Earth's Radiant Energy System (CERES) products and Ocean Biology Processing Group (OBPG) products.

We evaluated these HDF products when we first implemented the HDF4 OPeNDAP handler in 2008. After the implementation, we found that not all NASA data centers that distributed these data products used the HDF4 OPeNDAP handler. As the HDF4 OPeNDAP handler comes into the maintenance mode, we only add the support of the HDF4 products distributed by the data centers that used the most up-todate version of the HDF4 OPeNDAP handler. The following subsections will briefly discuss how latitude and longitude are obtained for these products. More information can be found under Appendix A.

4.1.2.1. TRMM

We support two versions of TRMM HDF4 products: TRMM version 6 and version 7.

4.1.2.1.1. *Version 6*

4.1.2.1.1.1 Swath

TRMM version 6 1B21, 2A12, 2B31 and 2A25 products are supported.

These products store both latitude and longitude in a three-dimensional field. The number of elements of one dimension in this field is always 2. To facilitate the discussion, this dimension is referred as the *latlon* dimension. If setting the index of the *latlon* dimension to be 0, the subset of the three-dimensional array actually stores the values of the latitude of this swath. If setting the index of the *latlon* dimension to be 1, the subset of the three-dimensional array stores the values of the longitude of this swath. In this way we can decompose this field into two coordinate variables, one for latitude and another for longitude. We then map this swath to CF like a general HDF-EOS2 swath described in 4.1.1.2.

4.1.2.1.1.2 Grid

TRMM version 6 3A46, 3B42 and 3B43, CSH products are supported.

These products don't provide latitude and longitude. Instead, latitude and longitude are calculated based on the outside document. One-dimensional latitude and longitude CF variables are defined to store the latitude and longitude values. More information regarding the retrieval can be found under Appendix A.

4.1.2.1.2. *Version 7*

4.1.2.1.2.1 Swath

We support TRMM version 7 1B01, 1B11, 1B21, 1C21, 2A12, 2A21, 2A23, 2A25 and 2B31 swath products.

The latitude and longitude values of these products are stored in the *Latitude* and *Longitude* fields under the vgroup *swath.* Both fields are represented as two-dimensional arrays that describe a typical swath on a horizontal plane. So these latitude and longitude fields are just mapped to two-dimensional latitude and longitude coordinate variables like a general HDF-EOS2 swath described in 4.1.1.2.

4.1.2.1.2.2 Grid

We support TRMM version 7 3A11, 3A12, 3A25, 3A26, 3B31, 3B42 and 3B43 products.

3A11, 3A12, 3A26, 3B42 and 3B43 are all represented as one single grid under the vgroup *Grid*. The 3A25 product is represented as two grids under the vgroup *Grid1* and *Grid2*. Latitude and longitude values for a single grid can be retrieved from a file attribute *GridHeader*. Latitude and longitude values for the 3A25 product can be retrieved from the file attribute *GridHeader1* and *GridHeader2*. The latitude and longitude values retrieved from these attributes can be stored in two orthogonal one-dimensional arrays. Therefore, one-dimensional latitude and longitude CF variables are defined. The rest of the mapping is similar to the HDF-EOS2 dimension reducing grid case described in section 4.1.1.2.

4.1.2.2. CERES

Currently we only support several CERES grid products. The versions of these products are from Edition 1 to 3. The abbreviated letters based on the documentation and the file names are used to distinguish these CERES grid products. More information on how the latitude and longitude of these products are retrieved can be found in Appendix A.

The latitude and longitude of CERES SYN and CERES AVG grids can be directly retrieved from the fields colatitude and longitude. They can be mapped to two-dimensional CF coordinate variables that store the latitude and longitude values. The rest of the mapping is similar to the general HDF-EOS2 grid mapping described in section 4.1.1.1.1.

CERES ISCCP-D2like-day and CERES SRBAVG3 grids are CERES nested grids. Latitude and longitude of these products are calculated according to the CERES nested grid formula. Two-dimensional CF coordinate variables are created to store the latitude and longitude values retrieved by the CERES nested grid formula. The rest of the mapping is similar to the general HDF-EOS2 grid mapping described in section 4.1.1.1.1.

Latitude and longitude values of CERES ISCCP-D2like-GEO and CERES ES4 products are stored as three-dimensional array data fields. However, these three-dimensional arrays can be condensed to one-dimensional arrays and are mapped to one-dimensional CF coordinate variables. The rest of the mapping is similar to the HDF-EOS2 dimension reducing grid case described in section 4.1.1.2.

4.1.2.3. OBPG

We only support the following OBPG products that can be found under the OBPG website [6]. These products are SeaWIFS, OCTS, CZCS, MODISA, MODIST level 2 and level 3 standard Mapped Image Products (l3m).

Level 2 products are equivalent to the HDF-EOS2 swath that uses the dimension map technique to reduce the latitude and longitude space. Similarly, subsets of latitude and longitude values are stored in the corresponding two-dimensional HDF4 fields. The complete latitude and longitude values can be obtained by interpolating the subsetted values according to the parameters that can be retrieved by using the HDF4 APIs. The HDF4 fields that store the subsets of latitude and longitude are mapped to the CF coordinate variables. However, the dimension sizes of the CF coordinate variables should be increased to be consistent with the number of physical data points.

Level 3m products are equivalent to the dimension reducing case of the HDF-EOS2 grid described in section 4.1.1.1.2. However, latitude and longitude values need to be calculated based on the parameters stored in several HDF4 file attributes. Two CF coordinate variables can be created to store the retrieved latitude and longitude values respectively.

4.1.2.4. Other products

Some NASA HDF4 products (AVHRR etc.) use the SDS dimensional scale APIs to store the scales of a SDS dimension. For such a case, SDS dimension scales are retrieved and are mapped to coordinate variables. We do not provide specification on how to check whether these coordinate variables contain latitude and longitude information. However, we try to make these coordinate variables follow COARDS. If the HDF4 products have coordinate variables that store the latitude and longitude values, the physical variables of these products can still be opened and plotted by CF tools.

4.2. Other coordinates

CF requires that each dimension that is not associated with latitude or longitude also has a corresponding coordinate variable. However, many NASA HDF4 and HDF-EOS2 products do not specify these coordinate variables.   Some products do not even provide the information about these coordinate variables at the product specification. For the CF tool to visualize the physical variables that contain such a dimension, a coordinate variable corresponding to this dimension must be provided. We observe that a variable that misses one or two coordinate variables for the corresponding dimensions is often a three or higher dimensional variable. The other coordinates of such a variable always include latitude and longitude. Based on our communications with NASA data center developers, it is desirable that such a variable can be visualized level by level in a horizontal plane. To fulfill this request, if we find a dimension that misses the coordinate variable, a proxy coordinate variable for that dimension will be generated. The value of this coordinate variable is index number 0, 1, 2, 3, etc. Attribute *units* with the value *level* is added to this coordinate variable so that the COARDS conventions are followed. By handling in this way, such a variable can be visualized by CF tools level by level in a horizontal plane.

Some NASA HDF4 and HDF-EOS2 product specifications provide values of some dimensions that are not associated with latitude and longitude. Naturally these dimension values should be used as values of the corresponding coordinate variables. Indeed, we create coordinate variables in this way for some of these products. However, for the rest products, we find that values of some of these dimensions do not conform to the definition of the coordinate variables specified in the CF conventions and thus cannot be visualized by CF tools if we assign these dimension values to the coordinate variables. According to the CF conventions, except for latitude and longitude, a coordinate variable is a one-dimensional variable and it must not have any missing data and must be strictly monotonic. But these dimension data either contain missing values or are not strictly monotonic. For these products, we still provide a proxy coordinate variable that fills with the index number to replace the values specified in the product specification. We add an attribute *comment* to store the information about the dimension based on the product specification.

5. CF attributes

CF conventions list many attribute names and the usage for the user communities. However, to make CF tools visualize the NASA HDF4 and HDF-EOS2 products, only a few key attributes need to be considered. These attributes are *coordinates, units, _FillValue*, *valid_range*(or *valid_min*, *valid_max*),*scale_factor* and *add_offset*, *long_name*. In this section, we provide the mapping information of these key attributes.

5.1. Coordinates

For products such as MODIS swath or MODIS sinusoidal grid, latitude or longitude values are stored as two-dimensional arrays that cover every location a physical variable was measured on a horizontal plane. For these products, the attribute *coordinates* needs to be provided for each physical variable measured at those locations. This attribute specifies the name list of the corresponding coordinate variables of the physical variable. For example, a variable *temperature* is stored as a two-dimensional array as follows:

Float Temperature [AlongTrack][CrossTrack]

The latitude and the longitude are defined as data arrays:

Float Latitude [AlongTrack][CrossTrack]

Float Longitude [AlongTrack][CrossTrack].

To make the CF tools visualize *Temperature*, the CF attribute *coordinates* needs to be added. The values of the *coordinates* should contain the names of the variables that store the latitude and longitude values. In this case, they are *Latitude* and *Longitude* as the form:

coordinates = "Latitude Longitude"

For cases such as the one discussed in section 4.1.1.1.2, the latitude and longitude values are stored as two orthogonal one-dimensional arrays. The COARDS conventions are followed and the *coordinates* attribute is not necessary.

5.2. Units

We enforce that the *units* attributes of the CF variables that store the latitude and longitude values exist and they should strictly follow CF since these attributes are crucial for CF tools to plot the data. The attribute *units* of the variable that stores latitude is *degrees_north* and the attribute *units* of the variable that stores longitude is *degrees_east*.

As described in section 4.2, we may need to add proxy coordinate variables for the dimensions that don't have any dimension values or that have dimension values but not conforming to CF. For these proxy coordinate variables, the *units* should always be *level*.

5.3. _FillValue

CF requires using attribute *_Fillvalue* to describe missing or undefined data. Without providing the*_FillValue* attribute, the CF tools may treat the undefined or missing data as real data, thus generate wrong plots. Some NASA HDF-EOS2 and HDF4 products contain undefined data values but don't provide the *_FillValue* attribute. Some products use an attribute name other than *_FillValue* to represent the undefined data or missing data. When mapping such a file to CF, the attribute *_FillValue* is created or the equivalent attribute is renamed to *_FillValue* to store the undefined or missing data.

CF also requires that the *_FillValue* attribute must be the same type as its associated variable. However, for some NASA HDF-EOS2 and HDF4 products, the data type of the *_Fillvalue* attribute is different than the associated variable, so the data type of *_Fillvalue* attribute should be correct to be the data type of the associated variable.

5.4. valid_range(valid_min and valid_max)

CF attribute *valid_range* represents smallest and largest valid values of a variable. *valid_range* can be replaced by *valid_min* and *valid_max*. *Valid_min* is the smallest valid value of a variable. *Valid_max* is the largest valid value of a variable. Attributes *valid_min* and *valid_max* are required if all the following conditions apply:

- There are multiple missing or undefined values or other special values.
- These values should be smaller than *valid_min* and larger than *valid_max.*
- Attribute *valid_range* is not presented.

Some NASA HDF4 and HDF-EOS2 products use different attribute names to represent smallest and largest valid values of a variable. To make the CF tools obtain the correct information of the valid data range, the attribute names need to be changed to follow CF. We also find that some products also use the same *scale_factor* and *add_offset* attributes as CF conventions specify but they don't follow the CF data packing rule. This will make their *valid_range* value invalid if using CF tool to plot the data. *Valid_range* needs to be recalculated. More information regarding this topic is discussed in section 5.5.

We also encounter some NASA products that only have the minimum valid values but don't have the *valid_min* attribute. For such a case, an attribute *valid_min* is added to store the minimum valid value.

5.5. scale_factor and add_offset

Scale_factor and *add_offset* are used by Earth Science users to pack the data in order to reduce the file size. To pack the data, certain rules need to be provided.

The CF conventions enforce the rule listed below to pack the data and the CF tools will follow this rule to obtain the final data.

*Final_data_value = scale_factor * Raw_data_value + add_offset*

However, some HDF4 and HDF-EOS2 data products have the same attribute names *scale_factor* and *add_offset* but use different rules to pack the data. This will make the CF tools apply the wrong packing rule to the original data, thus will generate the wrong plot. In order to make the CF tool correctly plot the data, we first check if the data packing rule of the HDF data is the same as the CF. If the rule is different, two options can be applied.

The first option is unpacking the data with the product's own data packing rule and then removing the *scale_factor* and *add_offset* attributes. If *valid_range*(or *valid_min*, *valid_max*) attribute is also present for the variable, we also calculate the new *valid_min* and *valid_max* based on the product's own data packing rule. As long as we know how the data is packed, we can unpack the data for all the data products that don't follow the CF packing rule. However, this method may demand huge computation time and thus may cause the performance deterioration.

Because of the performance issue of the first option, we also provide a second option. For many products we evaluate, although the data packing rule doesn't follow the CF conventions, the values of *scale_factor* and *add_offset* attributes can be transformed to new values so that CF tools can correctly interpret the data by following the CF data packing rule. For example, Some MODIS products adopt the following data packing rule:

Final_data_value = scale_factor(raw_data_value – add_offset)*

To transform the original *scale_factor* and *add_offset* to a new pair of *scale_factor* and *add_offset* so that the CF packing rule can be conformed, we do the following:

new scale_factor = scale_factor

*new add_offset = -1 * scale_factor*add_offset*

In this way, the CF data packing rule can be applied with the new *scale_factor* and *add_offset* values.

The restriction of the second option is that it requires the data packing rule follows a certain pattern. This may not be always true. For example, the following data packing rule cannot be transformed to the CF packing rule:

*Final_data_value = 10^ ((scale_factor*raw_data_value+add_offset)*

Currently for the examined NASA HDF-EOS2 and HDF4 products that don't follow the CF data packing rules, new *scale_factor* and *add_offset* values can be calculated to fulfill the CF data packing rule. Appendix D lists these NASA HDF-EOS2 and HDF4 products that the *scale_factor* and the *add_offset* values can be translated to follow the CF conventions.

5.6. long_name

CF attribute *long_name* aims to provide a long descriptive name for the variable. It is not required but it is strongly recommended. Some CF tools will display the *long_name* as the plot title. When mapping HDF4 to CF, if the *long_name* attribute is not present in the HDF4 variable and the original field name contains the non-alphanumeric characters not allowed by the CF conventions, the *long_name* attribute can be used to store the original field name.

6. References

1. HDF Specification and Developer's Guide

http://www.hdfgroup.org/release4/doc/DSpec_html/DS.pdf

2. HDF-EOS2 user's guide

http://newsroom.gsfc.nasa.gov/sdptoolkit/userguide.html

3. CF conventions

http://cfconventions.org/

4. HDF4 OPeNDAP handler

http://hdfeos.org/software/hdf4_handler.php

5. HDF4 to CF conversion toolkit

http://hdfeos.org/software/h4cflib.php

6. Ocean Biology Processing Group data site

http://oceandata.sci.gsfc.nasa.gov/

7. Revision History

January 8, 2015	Version 1.0.draft

8. Acknowledgements

The author especially would like to thank Mr. Dan Marinelli of NASA ESDIS project for his persistent support of this work. The author also likes to thank Dr. Fan Fang, Dr. James Johnson, Dr. Zhong Liu, Dr. Chris Lynnes and Dr. Abe Taaheri at the Raytheon for providing many technical suggestions and advice during the process of implementing the CF option of the HDF4 OPeNDAP handler.

This work was supported by Subcontract number 114820 under Raytheon Contract number NNG10HP02C, funded by the National Aeronautics and Space Administration (NASA). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Raytheon or the National Aeronautics and Space Administration.

Appendix A: Obtain latitude and longitude for some NASA HDF4 products

The following table lists the information on how to retrieve latitude and longitude values of some NASA HDF4 products to follow the CF conventions.

Table 8. How to retrieve latitude and longitude for some NASA HDF4 products

Mission	Product Name	Description	Documents*	Example files
TRMM	Version 6 Swath: 1B21,2A12,2A25,2B31	Latitude and longitude values are stored in one field *geolocation*.	TRMM-v6-to-v7	2B31v6.HDF
	Version 6 Grid: 3A46,3B42,3B43,CSH	Latitude and longitude are calculated according to the outside document.	3B42-43-V6-latlon 3A46-readme CSH-readme	3B42-V6.HDF 3A46-V6.HDF CSH-V6.HDF
	Version 7 Swath: 1B01,1B11,1B21,1C31,2A12, 2A21,2A23,2A25,2B31	Latitude and longitude are directly retrieved from the corresponding fields.	TRMM-V7-readme	2A12-V7.HDF
	Version 7 Grid: 3A11,3A12,3A25,3A26,3B31, 3B42, 3B43	Latitude and longitude are retrieved from a file attribute *GridHeader*.	TRMM-V7-readme	3B43-V7.HDF (single grid) 3A25-V7.HDF (multiple grids)
CERES	Edition 1-3 Grid: SYN , AVG	Latitude and longitude can be directly retrieved from the fields *colatitude* and *longitude.*	CERES-SYN-AVG-ZAVG	CER-SYN.hdf CER-AVG.hdf
	Edition 1-3 Grid: ISCCP-D2like-day, SRBAVG3	Latitude and longitude are calculated according to the outside document.	CERES-nestedgrid CERES-ISCCP-D2-Day	CER-ISCCP-D2-Day.hdf CER-SRBAVG3.hdf
	Edition 1-3 Grid: ISCCP-D2like-GEO , ES4	Latitude and longitude are stored in fields *colatitude and longitude*. These fields are stored as three-dimensional arrays. These three-dimensional arrays can be condensed to one-dimensional latitude and longitude arrays.	CERES-ISCCP-D2-GEO CERES-ES4	CER-ISCCP-D2-GEO.hdf CER-ES4.hdf
	Edition 1-3 Grid: ZAVG	Latitude can be calculated according to the documentation. Longitude is not needed.	CERES-SYN-AVG-ZAVG	CER-ZAVG.hdf
OBPG	L2 swath: SeaWIFS, OCTS, CZCS, MODISA,MODIST	Subset of latitude and longitude values can be retrieved from the fields *latitude* and *longitude*. For the older version, the final latitude and longitude fields need to be interpolated based on file attributes "*Number of Pixel Control Points*", "*Pixels per Scan Line*" and "*Number of Scan Line*".	OBPG-l2	CZCS-L2.hdf MODIST-L2.hdf
	L3SMI grid: SeaWIFS ,OCTS, CZCS, MODISA,MODIST	The latitude and longitude are calculated based on the file attributes "*Latitude Step*", "*Longitude Step*", "*SW Point Latitude* " and "*SW Point Longitude*".	OBPG-l3m	MODISA-L3m.hdf

Note: The document source is based on the best effort we can find on-line at the time this document is written. Not all documents contain clear information on how to retrieve latitude and longitude.

Appendix B: Special handling from HDF4 and HDF-EOS2 to DAP2

One software package implemented according to the specification is the CF option of the HDF4 OPeNDAP handler. OPeNDAP is a web service software package. Users can display HDF4 and HDF-EOS2 data via OPeNDAP's clients. DAP stands for Data Access Protocol. The current version of the HDF4 OPeNDAP handler maps the HDF4/HDF-EOS2 to version 2 of DAP(DAP2). Roughly DAP2 represents the metadata in two entities: Data Descriptor Structure (DDS) to store the variable type and dimension information. Data Access Structure (DAS) to store the file and variable attribute information. One can review more about the basic DAP concept at http://docs.opendap.org/index.php/UserGuide. This appendix only describes some special handling related to the mapping from HDF4 and HDF-EOS2 to DAP2.

1. ECS Metadata

Many NASA HDF-EOS2 and HDF4 products store EOSDIS Core System (ECS) metadata inside HDF-EOS2 and HDF4 files. To make the DAS output of DAP2 easy to read, a special parser is applied to all the ECS metadata attributes.

Generally *structmetadata* refers to an HDF-EOS2 library generated file attribute that stores the dimension and geo-location information of the HDF-EOS2 objects in an HDF-EOS2 file. If an HDF-EOS2 file contains many fields, the size of *structmetadata* may be very large. To speed up the performance, by default the DAS output of the *structmetadata* is turned off since the information in the structmetadata has been used by the handler to generate DAP's DDS and DAS.

2. Data Type

For most HDF4 datatypes, there is one to one mapping from HDF4 to DAP2. However, for the HDF4 datatype DFNT_CHAR, DFNT_UCHAR, DFNT_UINT8 and DFNT_INT8, special consideration is needed. Based on our evaluation of NASA HDF4 and HDF-EOS2 products, we will carry out the following mapping for the above datatypes.

HDF4 datatype DFNT_CHAR is used to store ASCII characters in NASA HDF4 or HDF-EOS2 products. So DFNT_CHAR is mapped to DAP2 string for variables and attributes. DFNT_UCHAR is equivalent to unsigned char in C. Although we have not observed the usage of DFNT_UCHAR in NASA HDF4 and HDF-EOS2 products we have evaluated. To make the mapping complete, we do the following: map DFNT_UCHAR to DAP2 Byte for variables and to DAP2 string for attributes. The reason to map DFNT_UCHAR to DAP2 string for attributes is because HDF4 explicitly provides DFNT_INT8 and DFNT_UINT8 for applications to store numeric 8-bit attributes and the usage of DFNT_UCHAR to store numeric 8-bit values is unnatural. However, it is possible for applications to use DFNT_UCHAR for character attributes. On the other hand, using DFNT_UCHAR instead of DFNT_CHAR for a character array in variables is unnatural to the writer of this document. One conjecture is that applications may be more careful in creating a variable. They may be more carefully to choose the appropriate character type compared with the creation of an attribute with the character type. In other words, they may misuse DFNT_UCHAR to represent characters in attributes but the chance to misuse DFNT_UCHAR to represent characters in variables is much smaller.

Datatype DFNT_UINT8 is equivalent to DAP2 Byte. There is no ambiguity to carry out the mapping. DFNT_UINT8 on the other hand, need to map to a bigger size integer datatype to avoid the data overflow. To conform to the default option of the HDF4 OPeNDAP handler, DFNT_UINT8 is mapped to 32-bit integer in DAP2.

There is an exception for the CF attribute *_FillValue*. Regardless of the data type of this attribute, it should always be mapped to a numeric type. If the data type is DFNT_CHAR, it will be mapped to 32-bit integer in DAP2. If the data type is DFNT_UCHAR, it will be mapped to Byte in DAP2. Furthermore, the attribute data type of the *_FillValue* should be the same as the data type of the variable. Table 8 summarizes the HDF4 to DAP2 datatype mapping.

Table 9. HDF4 to DAP2 data type mapping

HDF4 Data type	**DAP2 Data type**
8-bit unsigned integer(DFNT_UINT8)	Byte
8-bit signed integer(DFNT_INT8)	Int32
16-bit unsigned integer	UInt16
32-bit unsigned integer	UInt32
16-bit signed integer	Int16
32-bit signed integer	Int32
32-bit floating-point	Float32
64-bit floating-point	Float64
char(DFNT_CHAR)	String
unsigned char(DFNT_UCHAR) for variable	INT32
unsigned char(DFNT_UCHAR) for attribute	String

3. HDF4 vdata

In general, HDF4 vdata fields are mapped to DAP2 Arrays. However, sometimes the HDF4 vdata field that has a smaller number of records may logically mean metadata that is normally represented as an attribute. So in our implementation, we provide a DAP Back End Server(BES) dynamical operation key to map HDF4 vdata fields to attributes in DAS if the user sets that key value to be *true*.

Appendix C: Using tools to supplement CF information

Although the HDF4 OPeNDAP handler and the H4toCF conversion toolkit follow this specification to make non-CF attributes and even variables of many NASA HDF-EOS2 and HDF4 products follow CF, attributes and variables of some unexamined products may still need to be corrected so that CF tools can successfully visualize the data. In this appendix, we list two netCDF tools that can help users to correct the information by themselves.

1. Using NcML

NcML is an XML representation of metadata in a netCDF file. Both OPeNDAP Hyrax's NcML module and NcML local and web services based on Unidata's netCDF Java library can be used to do the following:

- Add, modify, or remove both attributes and variables
- Combine multiple datasets into a single "virtual" data set using "union",
- "joinNew", or "joinExisting" aggregations.

For more information, one can check the following NcML example page provided by The HDF Group for more information on how to use NcML to correct attributes.

http://hdfeos.org/examples/ncml.php

2. Using NCO

NetCDF Operator (NCO) can be used to edit variables and attributes of a netCDF file. It can also be used to supplement CF information to ensure that CF tools can correctly visualize the converted netCDF files.

For more information one can check the NCO example page provided by The HDF Group
http://hdfeos.org/software/nco.php

Appendix D: NASA HDF products that don’t follow CF data packing rules

We will only list the supported products that don’t follow the CF scale and offset data packing rule in this appendix.

1. MODIS products

There are two types of data packing rules for MODIS products.

Type A: *final_data_value = scale_factor*(raw_data_value-add_offset)*

Type B: *final_data_value = (raw_data_value – add_offset)/scale_factor*

Table 10 lists the detailed MODIS product names that follow either Type A or Type B to pack the data.

Table 10. MODIS products that don’t follow CF data packing rules

Type A Product	Type B Product
MODIS level 1B	MOD/MYD09 CMG
(MOD/MYD02HKM, MOD/MYD021KM etc.)	MOD/MYD09 GA
MOD/MYD 03	(Fields under grid *MODIS_Grid_500m_2D*)
MOD/MYD ATML2	MOD/MYD09 GHK
MOD/MYD 05	(Fields under grid *MODIS_Grid_500m_2D*)
MOD/MYD 06	MOD/MYD 09GQK
MOD/MYD 07	MOD/MYD 13A
MOD/MYD 08_D3	MOD/MYD 13A1 MOD/MYD 13C
MOD/MYD 09Q1	MOD/MYD 13Q1
MOD/MYD 09A1	MOD/MYD 13C1
MOD/MYD 09GA	
(Fields under grid *MODIS_Grid_1km_2D*)	
MOD/MYD 09GHK	
(Fields under grid *MODIS_Grid_1km_2D*)	
MOD/MYD 15A2	
MOD/MYD 15A2GFS	
MOD/MYD 17A2	
MOD/MYD 29E1	
MOD/MYD 43B4	
MCD43B4	
MCD43C1	

Note: Terra MODIS products are abbreviated “MOD”, Aqua MODIS products are abbreviated “MYD”. The combined Terra and Aqua MODIS products are abbreviated “MCD”. Normally for any MOD product, it will have a corresponding MYD product. We evaluate at least one MOD or MYD product listed in the MOD/MYD product pair.

2. TRMM version 7 Swath

Some TRMM version 7 swath products use the following rule to pack their data.

final_data_value = raw_data_value/scale_factor

These products are TRMM 1B11, 1B21,1C21, 2A25 and 2B31 products.