CERES Software Bulletin 97-02

February 12, 1997

An Overview of ECS Metadata, MCF, and ESDT

Alice Fan (t.f.fan@larc.nasa.gov)
Joe Stassi (j.c.stassi@larc.nasa.gov)

1.0 Purpose

This bulletin is written for the purpose of sharing metadata knowledge with members of the CERES Data Management Team. It begins by explaining ECS metadata terminology. It then gives some details about the Metadata Configuration File (MCF) and Earth Science Data Type (ESDT). The information in this bulletin was extracted from the ECS metadata documents listed in Appendix A. These documents can be accessed at the following web location: http://edhs1.gsfc.nasa.gov

2.0 Metadata Terms Used by ECS

Metadata are information about data. They are used for search, production, distribution, and other services required by users of ECS. Metadata identify the data and describe characteristics such as the origin, quality, and condition of the data.

There is a lot of terminology associated with ECS metadata. Below is a summary of some of the terms, along with a brief description of what they mean.

2.1 Core and Product-specific Metadata

The two major categories of ECS metadata are **core** and **product-specific**.

• Core metadata consist of a fixed number of attributes common to all earth science data sets. A certain subset of core metadata is required by every product in the ECS. This is the **mandatory** metadata.

• Product-specific metadata are data which pertain to a particular data product or multiple products, but not to all.

2.2 Discipline, Granule and Collection Level Metadata

Metadata can also be classified at the following levels: **discipline**, **granule**, and **collection**.

• Discipline level metadata are attributes which associate data with a certain discipline.

• Granule level metadata are metadata associated with a granule, where a granule is the smallest accumulation of data that is individually managed, archived, and distributed. Within the CERES project, a granule can be thought of as an individual instance of a data file. There are a few exceptions to this, such as the input geostationary narrowband radiance data which is received as multi-file granules.

• Collection level metadata are metadata associated with a collection, where a collection consists of one or more related granules. A **single-type collection** contains only one kind of granule type. A **multi-type collection** contains differing types of granules. Examples of collections in CERES include all the files of a certain data product type, such as the SSF. There could also be more than one collection associated with a data product type. For example, there could be one collection of TRMM SSFs and another collection of AM-1 SSFs.

Granule and collection level metadata will include both core and product-specific metadata.

2.3 Inventory and Archive Metadata

Another way of categorizing metadata is in terms of **inventory**, **archive,** and **structural** metadata. Only inventory metadata are required for each product.

• Inventory metadata include all core metadata and the searchable product-specific metadata. During PGE processing, these data are written directly to the data product. They are also written to an ASCII file for input into the inventory database.

• Archive metadata are stored only on the data product and consist of the non-searchable product-specific metadata.

Inventory and archive metadata are written to either HDF-EOS or HDF output files by Toolkit calls in the PGE science software. For non-HDF output files, the current plan is to store metadata on the files in a header format similar to that used for ERBE products.

• Structural metadata describe information about particular components within the granule such as geolocation information and variable name, type, and dimension. These are sometimes called sub-granule information and are in contrast to both inventory and archive metadata which contain information about the entire granule. Structural metadata are used for subsetting and subsampling of components by parameter and location. They apply only to HDF-EOS files. HDF-EOS tools automatically generate the structural metadata for swath, grid, and point structures. The user does not need to take any special action to produce them.

3.0 Metadata Control File

During PGE processing, each output file type requires a Metadata Configuration File (MCF). The MCFs list the metadata attributes to be associated with the outputs. They also list the location where the value for each attribute can be found. Since no dynamic information is expected to be kept within the MCFs, they should remain relatively constant from run to run.

An MCF template is supplied with the SDP toolkit and lists the various possible metadata attributes. The template is located in the toolkit runtime directory: $PGSDIR/runtime.

The format of the MCF is Object Description Language (ODL), a human-readable format. An explanation of ODL is given below for informational purposes. However, keep in mind that constructing an MCF is largely a matter of taking the template provided and editing it for a specific product. Editing must be in line with the rules for mandatory and non-mandatory metadata attributes laid down in Appendix B of DID311. There is also related information in Appendix J of the Toolkit User's Guide. Beware that DID311 is still under review, and the current version of the Toolkit User's Guide applies to Release A only.

3.1 ODL Format Overview

The three basic components of ODL format are **groups**, **objects**, and **parameters**. Metadata attributes are stored in the MCF as objects. The components of objects are parameters. Collection of objects are groups. ODL also allows for superset groups containing nested groups. Any time any of these words ("group", "object", "parameter") occur in this bulletin, the ODL-meaning for the word should be assumed rather than any other specific or general meaning which the word may have.

Three other concepts covered in this section are **master groups**, **multiple-class objects**, and **self-describing objects.** A master group is a special superset grouping used to distinguish between inventory, archive, and structural metadata. A multiple-class object structure allows two or more metadata attributes to have the same attribute name. Self-describing attributes are used for supplying product-specific metadata.

3.2 Master Groups

A master group is the smallest unit of metadata that is written to output. Therefore, all the metadata attributes on an MCF must be contained within master groups.

There are two possible master groups found on the MCF: one for inventory metadata and one for archive metadata. The inventory metadata master group contains attributes which are written to both the inventory database and to the data product. These metadata are required for each MCF. The archive metadata master group contains attributes which are written only to the data product. These metadata are optional. A third master group is for the structural metadata, but this is not handled on the MCF.

Table 1 below shows a high-level, master group view of the MCF.

```
Group = INVENTORYMETADATA
      GroupType = MASTERGROUP
      [objects and/or nested groups]
End_Group = INVENTORYMETADATA

[Group = ARCHIVEMETADATA
      GroupType = MASTERGROUP
      [objects and/or nested groups]
End_Group = ARCHIVEMETADATA]
```

Table 1: An MCF consists of one or two Master Groups

The statement, *Group = INVENTORYMETADATA*, is the identification format used by all groups in the MCF. To distinguish master groups from the other groups, the statement, *GROUPTYPE = MasterGrou*p, must follow the GROUP statement.

3.3 Groups

In general, groups associate related attributes together. We've already seen a special case of groups with the master groups.

A group begins with the statement *GROUP = GroupName* and ends with the statement *END_GROUP = GroupName* (the keywords and values in these statements are case-insensi-

tive). Between the *GROUP* and *END_GROUP* statements are objects and/or nested groups. The format for ODL groups is shown in Table 2 below.

```
Group = GroupName
        [objects and/or nested groups]
End_Group = GroupName
```

Table 2: ODL Group Format

Keep in mind when creating an MCF that the grouping structure and names should be copied from the template.

3.4 Objects

The object is the basic component of the MCF. Each object defines a metadata attribute. The object structure must start with *OBJECT = ObjectName* and end with *END_OBJECT = ObjectName*. The ObjectName relates directly to the attribute name. The general format for the object structure is shown in Table 3 below.

```
Object = ObjectName
       Data_Location = {"MCF", "PCF, "PGE", or "NONE"}
       Type = {"INTEGER", "UNSIGNEDINT", "DOUBLE",
             "STRING",or "DATETIME"}
       Num_Val = {1, 2, ...}
       Mandatory = {"TRUE", or "FALSE"}
       [Class = "M"]
       [Value = "xxxx"]
End_Object = ObjectName
```

Table 3: ODL Object Format

A list of mandatory attributes can be found in Appendix B of the ECS document DID311 or in Appendix J of the Toolkit User's Guide.

3.5 Parameters

In each object, there are only six possible parameters (as seen in Table 3). They are (1) **Data_Location**, (2) **Type**, (3) **Num_Val**, (4) **Mandatory**, (5) **Class**, and (6) **Value**. The first four parameters should appear in every object. The Class parameter is required only for multiple class objects. The Value parameter is required only if the attribute value is defined within the MCF itself. The parameter keywords are case-insensitive. The order of the parameters within the objects is not critical.

Notes on object parameters:

1) **Data_Location**: The Data_Location parameter has only four possible values: *"MCF"*, *"PCF"*, *"PGE"*, and *"NONE"*. These values must be capitalized and in quotes.
   • *"MCF"* implies that the value is defined within the object using the Value parameter. Only constant-valued attributes are stored in the MCF (e.g. product ShortName).
   • *"PCF"* implies that the Process Control File (PCF) has an entry for the attribute.
   • *"PGE"* implies that the PGE will assign a value to the attribute during runtime.
   • *"NONE"* is valid only for containment objects of multiple class objects (see Section 3.6). The *Class = "M"* parameter must be set in this case.

2) **Type**: The Type parameter allows the metadata tools to cast the data type of the attribute. Currently, there are five available types: *"INTEGER"*, *"UNSIGNEDINT"*, *"DOUBLE"*, *"STRING"*, and *"DATETIME"*. These values must be capitalized and in quotes.
3) **Num_Val**: The Num_Val parameter allows an attribute to contain an array of values rather than just a single value. Num_Val should be set to the maximum number of values to be held by the attribute. This value sets a limit only, and any number up to the maximum will be acceptable.
4) **Mandatory**: The Mandatory parameter has possible values of *"TRUE"* or *"FALSE"*. The values must be capitalized and in quotes. Pre-defined attributes specified as mandatory in DID311 should all be flagged as *"TRUE"*. If a mandatory attribute has not had a value assigned to it before the metadata are written, an error message will be written to the LogStatus file.
5) **Class**: The Class parameter should only be present for multiple-class objects. It has only one possible value, *"M"*, which stands for multiple class object. The Class parameter is explained in more detail in the next section.
6) **Value**: The Value parameter is present only when *Data_Location = "MCF"*. Currently, certain mandatory attributes, e.g. ShortName, are required to be defined within the MCF.

3.6 Multiple Class Objects

A multiple-class object is a way to allow two or more metadata attributes to have the same attribute name. A multiple-class object is created by including the Class parameter and setting it equal to “M”. Table 4 shows an example of the multiple class objects, SensorShortName and InstrumentShortName.

```
Group = SensorCharacteristic
      Object = SensorCharacteristicContainer
            Class = "M"
            Data_Location = "NONE"
            Mandatory = "FALSE"
            Object = SensorShortName
                  Data_Location = "PGE"
                  Class = "M"
                  Type = "STRING"
                  Num_Val = 1
                  Mandatory =  "FALSE"
            End_Object = SensorShortName
            Object = InstrumentShortName
                  Data_Location = "PGE"
                  Class = "M"
                  Type = "STRING"
                  Num_Val = 1
                  Mandatory =  "FALSE"
            End_object = InstrumentShortName
      End_Object = SensorCharacteristicContainer
End_Group = SensorCharacteristic
```

Table 4: Two Multiple Class Objects (SensorShortName and InstrumentShortName) with Containment Object (SensorCharacteristicContainer) and Bounding Group (SensorCharacteristic)

Multiply defined objects are usually bounded by a containment object, and they must be bounded by a group name. In the example above, the containment object is SensorCharacteristicContainer and the bounding group is SensorCharacteristic.

The container object does not represent a specific metadata attribute in the sense that normal objects do, and it should therefore contain the parameter *Data_Location = “NONE”*. The class parameter must be set to *CLASS = “M”* for both the container object and the objects within the container.

3.7 Self-Describing Objects

There is a category of objects in the MCF template which have been labelled as “self-describing.” These objects will house attributes relating to Parameter, AdditionalAttribute, SensorCharacteristics, and PlatformCharacteristics, and to a lesser extent VerticalSpatialDomain, Locality, and RegularPeriodic time. Two self-describing objects, AdditionalAttribute and Parameter, are of particular interest to us because they can be used to supply product specific

metadata to the MCF. Only Parameter, a multiple-class object located in the InformationContent group, is supported in Release A. This group is shown in Table 5.

```
GROUP = InformationContent
      OBJECT = InformationContentContainer
            CLASS = "M"
            Mandatory = "FALSE"
            Data_Location = "NONE"
            OBJECT = ParameterName
                  Data_Location = "PGE"
                  TYPE = "STRING"
                  CLASS = "M"
                  NUM_VAL = 1
                  Mandatory = "FALSE"
             END_OBJECT = ParameterName
            OBJECT =  ParameterValue
                  Data_Location  = "PGE"
                  CLASS = "M"
                  TYPE = "STRING"
                  NUM_VAL = 1
                  Mandatory = "FALSE"
            END_OBJECT =  ParameterValue
      END_OBJECT = InformationContentContainer
END_GROUP = InformationContent
```

Table 5: Examples of self-describing objects

As an example of how we can use self-describing objects, in CERES we will want to include production date/time information in our granule metadata. This attribute is not currently available on the MCF template, so we will have to use self-describing objects to get this information into the metadata. This can be done with ParameterName and ParameterValue, setting ParameterName to “productiontime” and ParameterValue to a string representation of the date/time. Specifics on how to write multi-class objects is handled in CERES Software Bulletin 97-03.

4.0 Earth Science Data Type

An Earth Science Data Type (ESDT) is required for every product stored within the Data Server. It has many functions, but as relating to metadata, it is the mechanism by which core and product-specific metadata attributes for each data collection and granule are made known to the Science Data Server. Each collection needs one ESDT.

There are two components which define an ESDT: a Descriptor File and a Dynamic Link Library (DLL). The Descriptor File is an ASCII file which contains collection level metadata in addition to the granule level metadata described in the MCF. It also contains the ECS Data Server services which can be invoked for the collection. Examples of the granule level services are insert, acquire, subsetting, subsampling, and averaging. Examples of the collection level services are search and retrieve. The DLL consists of C++ code which implements the services defined in the Descriptor File. The contents of the Descriptor File and the Dynamic Link Library are stored in

the Data Server.

The Science Data Server, however, does not make direct use of the attribute information contained in the Descriptor. The granule level metadata stored in the Data Server, both core and product-specific, are used by the Planning Subsystem to create the MCF file for each production run.

Collection level metadata are not searchable via the Science Data Server. Collection level information is passed from the Data Server to the Client, Interoperability, Data Management, and Advertising Subsystems to be searched.

Appendix A: Reference Documents

1. Release A SCF Toolkit Users Guide, November 1996
2. An ECS Data Provider's Guide to Metadata, August 1996
3. The Population Process for ECS Metadata in Release A, August 1996
4. Science Data Processing Segment (SDPS) Database Design and Database Schema Specifications for the ECS Project, December 1995
5. Thoughts on HDF-EOS Metadata, December 1995