YAML Encoding ASCII format for GRACE/GRACE-FO mission Data

Status of this Memo

This RFC provides information to the NASA Earth Science Data Systems (ESDS) community. This RFC does not specify an Earth Science Data Systems (ESDS) standard. Distribution of this memo is unlimited.

Change Explanation

Not Applicable.

Copyright Notice

Abstract

This document introduces and describes the YAML encoding as the standard format for NASA GRACE/GRACE-FO mission ASCII data. YAML encoding provides a simple and user-friendly structure for representing earth science data in the ASCII format. YAML, technically, a super set of JSON, is designed to be useful and friendly for people working with data. Its software agonistic accessibility adds great power to data stored in an ASCII file that is being used across different platforms and programming languages. The YAML format has been successfully applied to the NASA GRACE/GRACE-FO Missions (a series of NASA missions to measure the earth's gravity) ASCII products, mainly focused on the file header structure. Adhering to a standardized metadata model from the PO.DAAC metadata best practices document [8] that is in compliance with CF, ACDD and ISO conventions, the YAML header enables the ASCII file to become more discoverable, accessible, interoperable, and reusable. This document provides detailed guidance for constructing a YAML formatted ASCII file for earth science data.

Table of Contents

1 Introduction

The American Standard Code for Information Interchange (ASCII) data format is widely used and adopted in the Earth science community for its simplicity and accessibility. Many NASA Earth observing missions are required to have the ASCII format for their mission product data. However, the ASCII format for different products can vary dramatically due to the complexity of Earth

science data. It is a challenge to develop a standard format which can fit most NASA mission ASCII products. There have been previous efforts to develop standard ASCII formats. So far, the NASA Earth Science Data and Information System (ESDIS) Standards Office (ESO) has approved *ASCII File Format Guidelines for Earth Science Data* [1] as an overall guidance for developing ASCII encoded data files. Several ASCII formats have been approved by the ESO, including *SeaBASS Data File Format* [4], *NASA Aerogeophysics ASCII File Format Convention, and International Consortium for Atmospheric Research on Transport and Transformation* (ICARTT) File Format Standards [3], which are all in compliance with the ASCII file format guidance. However, most of these formats are designed for the specific mission/project data and their domain specific resources (research/tools/services), which may not be suitable for re-use. In this document, we propose the YAML ASCII encoding as a format for GRACE/GRACE-FO missions, which could be used for a broader set of Earth science data in the future.

"YAML Ain't Markup Language" (YAML) is a Unicode-based data serialization language designed to be a human-friendly, text-based interface that works well with modern programming languages for passing information [2]. It is built upon a few basic core components which make the text clean and readable. The core syntax includes indentation, a key-value pair scheme, scalars (data type), and collections (sequence and mapping); all of these features enable users to create a YAML encoding file with minimal effort. YAML format structure is very much like the Common Data Form Language (CDL): a human-readable text representation of netCDF data, giving a netCDF user a familiar reference.

YAML is closely related with JavaScript Object Notation (JSON), another ASCII encoding format, which is widely used across the internet in web tools and services. Technically YAML is a superset of JSON. This means that, in theory at least, a YAML parser can understand JSON, but not necessarily the other way around. However, the YAML format has a much cleaner and more human-friendly format structure than JSON, while the JSON is simpler to generate and parse.

Currently, the ESO approved ASCII formats that are under the umbrella of the *ASCII File Format Guidelines for Earth Science Data* [1] are designed for specific data and missions. These formats require dataset specific readers to parse and extract values and information. Parsing such ASCII files with cross-platform generic tools is often difficult to implement. These limiting factors all drive the need for generic ASCII encoding formats which will work with multiple programs on different platforms, thus ensuring sustained interoperability.

Unlike many other ASCII formats, the YAML format not only provides a user-friendly interface but also can be easily parsed with many programming languages, including Python, Perl, JAVA, C/C++, and more. This flexibility across languages and platforms enhances the utility of the YAML-encoded ASCII file. Enabling the YAML parsing function for any programming language is easy. Users just need to install the appropriate YAML library following directions described at YAML official website: http://yaml.org/. These characteristics of YAML files allow them to be more accessible and usable by web services and tools.

In this document, we will first briefly describe the basic YAML syntaxes which are used for constructing a YAML ASCII file (section 2). Second, a generic YAML file structure is presented with information from the GRACE Level-2 RL06 product (section 3). Third, we present an example to demonstrate how the YAML file can be opened/read/parsed and information can be

extracted at any given key and time/location using Python scripts (section 4). Finally, we summarize the YAML encoding implications to Earth science data (section 5).

2 Basic YAML syntax

In this section, we focus on the core YAML syntax used for constructing an ASCII file. For a complete list of syntax, please see the YAML official website: http://yaml.org/. The YAML syntax is basically composed with several key components, including the key-value pair scheme, indentation scheme, hierarchy nested structure (layer), sequence (array/list), mapping (dictionary), and common data type. In this document, for clarity and simplicity, we encourage the users to use the most basic and common YAML syntax and try to avoid the use of complex syntax as much as possible. The goal is to make the ASCII file implementation simple and readable.

The YAML language accepts the entirety of the Unicode character set, except for a small subset of the control characters. A YAML document can be encoded in UTF-8, UTF-16 and UTF-32. (Though UTF-32 is optional). We recommend to use the standard US-ASCII character set without extensions, and avoid the use of ASCII control characters, except the end-of-line (EOL). Each row should be terminated with the same EOL character. No escape mechanism is used in the YAML encoding except appearing in string. An empty line (row) can be allowed for clarity between the header and data section, but it is not necessary. Empty lines have no affects in parsing the YAML files.

2.1 Key-value pair scheme

YAML format is built upon a series of key-value pairs separated by a colon (Example 2.1). The "key" is on the left of the colon, referring to any item name which is defined by the "value" on the right side of the colon. The "key" is just a string without a quote. The "value" can be any data type which will be explained in later sections. The structure requires that the right side of the colon must have at least one space (":_"), which is an important feature to separate from an unquoted string containing the colon, such as an url string (Example 2.3.3 String4). Often, for clarity, more spaces can be added to either side of the colon. The key-value pair scheme is the fundamental element in YAML syntax.

Example 2.1. Kay-value pair separated by a colon.

```
long_name   :  mean equator radius
units       :  m3/s2
value       :  3.9860044150e+14
```

2.2 Indentation scheme

The scope of a YAML structure is defined by the indentation scheme, which is generated with spaces rather than tabs. To maintain portability, tab characters are never allowed as indentation, since different systems treat tabs differently. Each item that is a child of the parent item should be indented with a fixed number of spaces, which must be consistent throughout each YAML file

(snippets:https://yamllint.readthedocs.io/en/stable/rules.html#module-yamllint.rules.indentation). In this document, we adopt a two-space indentation scheme for all examples. The indentation is the unique indicator to nest a child YAML structure (Example 2.2.1) or an array/list sequence (Example 2.2.2), while for JSON, the curved bracket ("{}") is used instead. YAML achieves a unique cleanness by minimizing the amount of structural characters and allowing the data to present itself in a natural and meaningful way.

Example 2.2.1. Child structures with two-space indentation.

```
header :
  dimensions :
    lat  :  180
    lon : 360
  global_attributes :
    title  :  This is GRACE Level-2 GAA product
```

Example 2.2.2. Sequence list with two-space indentation.

```
unused_days   :
  -  2003-02-08
  -  2003-02-20
  -  2003-02-21
```

2.3 Scalar data types

The basic YAML data types include integer, float, double, Boolean, string, and date. The key on the left colon can be assigned by any of these data types on the right of the colon. The data type can be used implicitly (Example 2.3.1) or explicitly (Example 2.3.2) by using the "key-value" pair scheme. For clarity, we recommend using implicit typing

Example 2.3.1. Implicit typing scheme (recommended).

```
integer   :  12345
float     :  123.456
booleans  :  yes
```

Explicit typing is denoted with a tag using the exclamation point ("!") but this is not recommended in the YAML ASCII encoded files.

Example 2.3.2. Explicit typing scheme (not recommended)

```
integer   :  ! integer 12345
float     :  ! float 123.456
booleans  :  ! booleans yes
```

Assigning a string value to the key is very straightforward. There are three ways to express a string in YAML format. The string as a value can be unquoted or quoted with single (‘’) or double (“”) quotes (Example 2.3.3). Within double-quotes, special characters may be represented with C-style escape sequences starting with a backslash (\). According to the documentation the only octal escape supported is \0.

If the colons or commas appear in an unquoted string, they can be distinguished from being used as separators by adding a space right after, so that scalar values containing embedded punctuation (such as 5,280 or http://www.wikipedia.org) can generally be represented without needing to be enclosed in quotes (Example 2.3.3, string4).

Example 2.3.3. Three string assignments.

```
string1  :  string without any quote
string2  : 'string with single quote'
string3  : "string with double quote"
string4  :  https://podaac.jpl.nasa.gov
```

2.4 Line continuation symbols “|” or “>”

String values can be written in block notation with multiple lines using a literal style (“|”) or a folded style (“>”). Using a “Literal Block Scalar” (“|”), spanning multiple lines will include the newlines and any trailing spaces, while using a “Folded Block Scalar” (“>”), they will fold newlines to spaces. In either case, it’s used to make what would otherwise be a very long line easier to read and edit. Note that the indentation will be ignored in both cases.

Example 2.4.1. Using literal style “|”

```
summary  : |
           Spherical harmonic coefficients that represent anomalous contributions
           of the non-tidal atmosphere to the Earth's mean gravity field during
           the specified timespan
```

Example 2.4.2. Using folded style “>”

```
summary   : >
            Spherical harmonic coefficients that represent anomalous contributions
            of the non-tidal atmosphere to the Earth's mean gravity field during
            the specified timespan
```

2.5 Collections

YAML collections cover both the sequences (indexed arrays or lists in Python) and the mapping (dictionaries in Python) structures. The syntax for these is a data block or nested blocks containing multiple pieces of information, similar to a structure definition in C/C++ language.

2.5.1 Sequences

YAML sequences refer to arrays or lists depending on the content type of the elements and the interpreters (such as Python/Perl/C). In addition to assigning a single scalar to a key, one could assign an array of numerical values and/or a list of mixed data types. For a short list of data values, there are two ways to assign the values to a key. The first case is that all elements in the array/list are separated by comma (,) and are placed in one line closed with square brackets ([ ]) (example 2.5.1) – called compact in-line notation. This is generally useful for a short array/list. The second case is that each element will be placed in a single line with leading hyphen ('- ') and following spaces (example 2.5.2) – called nested block notation. For a big data array/list, one can break the data array/list into multiple lines. This will be discussed in more detail in section 3.2 data-block.

Example 2.5.1. Compact in-line notation

```
single_data_type1  : [ 10, 20 ,  30, 40 ]      mix_data_typr 3 : [ string1, 10,   string2, 20 ]
single_data_type2  : [[10, 20], [30, 40]]      mix_data_typr 4 : [[string1, 10], [string2, 20]]
```

Example 2.5.2. Nested block notation

```
single_data_type1 :      single_data_type2 :      mix_data_typr3 :      mix_data_typr4 :
 - 10                      -                        - string1              -
 - 20                        - 10                   - 10                     - string1
 - 30                        - 20                   - string2                - 10
 - 40                      -                        - 20                   -
                             - 30                                             - string2
                             - 40                                             - 20
```

2.5.2 Mapping

Mapping refers to the dictionary structure in Python. It gives a user the ability to list and group key-value pairs with a collection of information. This is useful in cases in which the user is passing a group of names and properties to a specific element (attribute). The example (Example 2.5.2.1) below shows the "header" key is a top-level dictionary with two child structures (*dimension and global_attributes*), which in turn have their own children. A dictionary can have multiple nested layers of sub-structures.

Example 2.5.2.1 Mapping structure

```
header :
  dimensions :
    lat  :  180
    lon  : 360
  global_attributes :
    title  :  This is GRACE Level-2  GAA product
```

2.6 Comment symbol "#"

Comments begin with the number sign (#), can start anywhere in a line and continue until the end of the line. It is a presentation detail and must not be used to convey content information. Comments must be separated from other tokens by white space characters. If they appear inside of a string, then they are number sign (#) literals.

Example 2.6

```
# Begin of the header
header:
 - dimensions              # describe the geolocation and time
 - global_attributes       # file scape of attributes
 - project_attribute       # project specific attributes

show_bumber: "there are '#' of elements" # string contains the "#"
# End of header
```

Above, we only describe very basic YAML encoding syntax which can satisfy 90% of the YAML ASCII file formatting. There is more YAML syntax which is not described here. Some of them are not so common for use in presenting Earth data in ASCII format. To keep the YAML ASCII file simple and consistent across other YAML-formatted products, we recommend avoiding the use of the special YAML syntax as much as possible.

3 ASCII file Structure in YAML format

Satellite ASCII products commonly contain two major components: a header and a data-block. The header is composed of rich, complete metadata information which summarize and describe the product, while the data-block contains all the data records for the product, including the instrument measurements, model derived results, data quality, and ancillary data. In this section, the construction of header and data blocks with YAML encoding is shown using examples from GRACE/GRACE-FO Level-2 products.

Overall, the YAML formatted ASCII file recommendations are in compliance with the requirements proposed by the *ASCII File Format Guidelines for Earth Science Data* [1], which has been approved by ESO. The ASCII syntax recommendations in that document are applicable and will not be discussed in detail here. In this document, we just focus on the YAML architecture and format.

3.1 Header

The header should be located at beginning of the file, containing the metadata associated with the data products. To clearly separate the header section with the data-block, the header should start and end with specific comments, such as "# BEGIN HEADER" and "# END HEADER". A YAML formatted header is very similar to the CDL format, which is a human-readable text representation of netCDF data, containing at least three key sections: *"dimension", "global_attributes",* and *"variables".*

The GRACE/GRACE-FO mission products have adopted the metadata conventions which are recognized in the earth science community, including the Climate Forecast (CF) metadata conventions, the Attribute Conventions for Data Discovery (ACDD) and International Standards Organization (ISO) conventions. For a YAML header designed to meet the NASA requirements for standardizing the ASCII data, we recommend that the header should follow an accepted NASA metadata model such as the PO.DAAC Metadata Best Practices [8] which utilizes the aforementioned CF, ACDD, and ISO conventions. Most of the elements (attributes) described in the baseline ASCII File Format Guidelines have been defined and addressed.

An example of YAML format application for GRACE Level-2 RL06 dataset is presented in Appendix B for reference. GRACE mission requires that each product file name must be uniquely defined following the GRACE filename convention described in Level-2 Gravity Field Product User Handbook [6]. The detailed descriptions of the data products, including the data type, uncertainty (standard deviation), solution-flag, time_stamps, and others, are documented in the Product Specification Document [7].

Using the examples of the YAML format applications in the GRACE/GRACE-FO Missions, the header has been divided into four sections: "*dimensions", "non-standard_attributes", "global_attributes'', and "variables",* of which the "*non-standard_attributes*" section is designed for project specific global attributes. Figure 1 shows the basic skeleton of the header structure in YAML format.

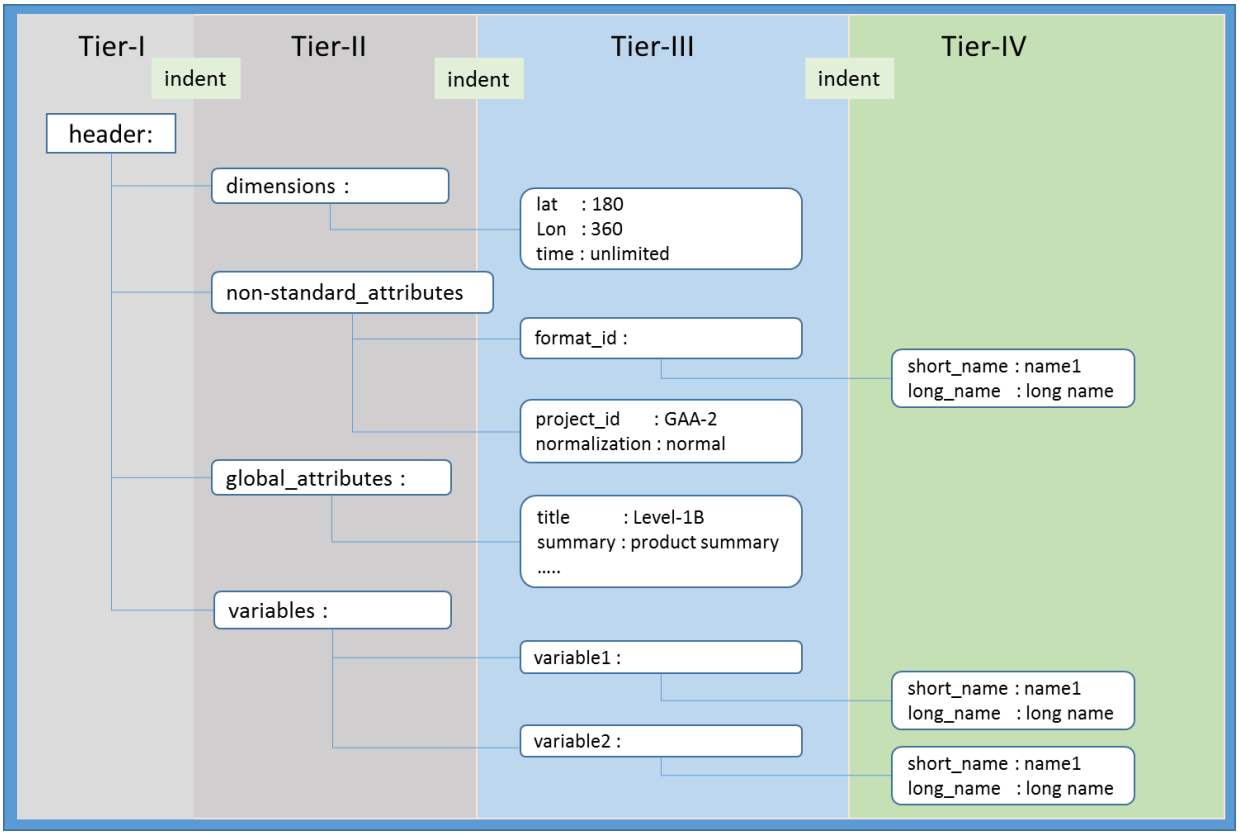

Figure 1, YAML header structure scheme, which contains four major sections (*dimensions, non-standard_attributes, global_attribute, and variables*), and four layers of nested sub-structures (denoted by four Tiers).

3.1.1 Dimensions

The "*dimensions*" section usually defines the spatial and temporal dimension. For the geolocation, it could be latitude and longitude, following UDUNITS conventions [9] in unit of degree (Appendix C, Table 3). The time refers to the satellite scan time (data recording time), following the ISO 8601 convention (Appendix C, Table 4). However, in some special cases like GRACE, the dimension could refer to the degree and order of a geopotential series (Example 3.1.1).

Example 3.1.1.

```
dimensions :
  degree  :  90
  order   :  90
```

3.1.2 Global_attributes

The "*global_attributes*" covers common standard metadata attributes adhering to a metadata model as described in PO.DAAC metadata best practices [8] recommendations that are in compliance with the most recent versions of CF, ACDD and ISO conventions. It is recommended to use these global attributes conventions as much as possible for all NASA mission products for maintaining the data's interoperability and reusability. Non-standard global attributes are allowed as necessary to meet mission specific requirements.

The complete list of global attributes (elements) is summarized in Appendix C, Table 1, which meet the majority of metadata requirements listed in the ASCII File Format List. Many of the attributes are strongly recommended or required for most earth science data, such as the *'title', 'summary', 'keywords', 'conventions', 'id/uuid', 'history', 'source', 'processing_level', 'creator_name', 'time_coverage_start', 'time_coverage_end', 'date'_created', 'geospatial_(lat, lon, vertical)_(min, max, unit, resolution)*' and so forth. For the best practice of constructing the header metadata, we encourage using all the attributes which are applicable. However, for some earth science products, not all of the attributes are suitable. For example, the GRACE mission measures the spatial distribution of earth mass movement, so there are no vertical properties presented in the datasets (see example in Appendix B).

In YAML format, the "*global_attributes*" section is a collection which contains multiple scalars and sequences for assignment to the attribute keys. All the file global metadata can be defined with attribute properties listed in Appendix C, Table 1 by using the YAML key:value pair syntax. See example 3.1.2.

Example 3.1.2.

```
global_attributes:
   title        :  AOD1B ATM Coefficients JPL
   summary  : |
      Spherical harmonic coefficients that represent anomalous
      contributions of the non-tidal atmosphere to the Earth's mean
      gravity field during the specified timespan.
   ……..
   unused_days :  [2003-02-08, 2003-02-20]
```

In this example, the "*unused_days*" attribute is not standard, but it is included here by the GRACE project request.

3.1.3 Non-standard_attributes

The "*non-standard_attributes*" section is also covering file scope (global) metadata, similar to the "*global_attributes*", except defining any attributes that are specific to projects/missions, such as GRACE/GRACE-FO. They are unique only to the particular mission/project, and will not be suitable for other products. The example 3.1.3. shows a partial of the non-standard attributes from a GRACE/GRACE-FO mission Level 2 product and a full version is included in Appendix B. In

this example, it includes the mission specific product_ID, format_ID, earth gravity parameter and earth radius value, each of which is further defined by *long name, standard name, unit, value,* and *comment* attributes.

Example 3.1.3.

```
non-standard_attributes:
   product_id       : GAA-2
   format_id :
     short_name     : SHM
     long_name      : Earth Gravity Spherical Harmonic Model Format
   normalization    : fully normalized
```

3.1.4 Variables

All the variables in the ASCII product must be defined in a "*variables*" section and the variable names must be unique, following earth science community standards, such as the GRACE mission product specification [7]. In the YAML format, each variable has its own single nested-substructure defining the properties of the variable. For example, the variables in example 3.1.4 include the *long_name, units*, *coverage_content_type, valid_range,* and *comment*, and also identifies the column which is presented in the data section.

The Appendix C Table 2 summarizes the general comprehensive attributes used to define each data variable (parameter), which are defined in the CF and ACDD conventions. In reality, some attributes may not be applicable to certain products, so often, not all the attributes are used. We recommend that each variable should have at least a *standard_name* or the *long_name* clearly defined, and the variable unit should be included and defined whenever it is applicable. Missing data should be clearly flagged in data (e.g. with *_Fillvalue*) or identified in global_attribute section (e.g. *unused_days*). Again, we recommend the use as many attributes as applicable for all variables. Georeferencing and temporal coordinates should also be defined as separate variables, whose attributes are summarized in Appendix C Table 3 and 4.

Example 3.1.4.

```
variables:
   biased_range:
      long_name: |
           CRN-filtered biased inter-satellite range with ionospheric correction
      units: meter
      coverage_content_type: physicalMeasurement
      valid_range:  [-999.0f, 999.0f]
      comment: 2nd column
  range_rate:
      long_name: First time derivative of biased_range
      units: meter/second
      coverage_content_type: modelResult
      comment: 3rd column
```

3.2 Data-Block

3.2.1 YAML standard data-block structure

The YAML standard data-block should contain all data records for the variables in the product, including sensor measurements, Level-1, 2, and 3 products, model derived physical parameters, and other associated datasets. In ASCII format, these data record follow a certain order in space or time or some sequential order, such as monotonically increasing in time presented in GRACE Level 2 product. For most cases, each variable present in a single column, and each data record takes a single row, forming a two-dimensional array/list (Figure 2). In most ASCII products, the file contains only a single data-block. However, the YAML formatted file is capable of multiple data-blocks. In this document, we just focus on the single data-block case (Figure 2)

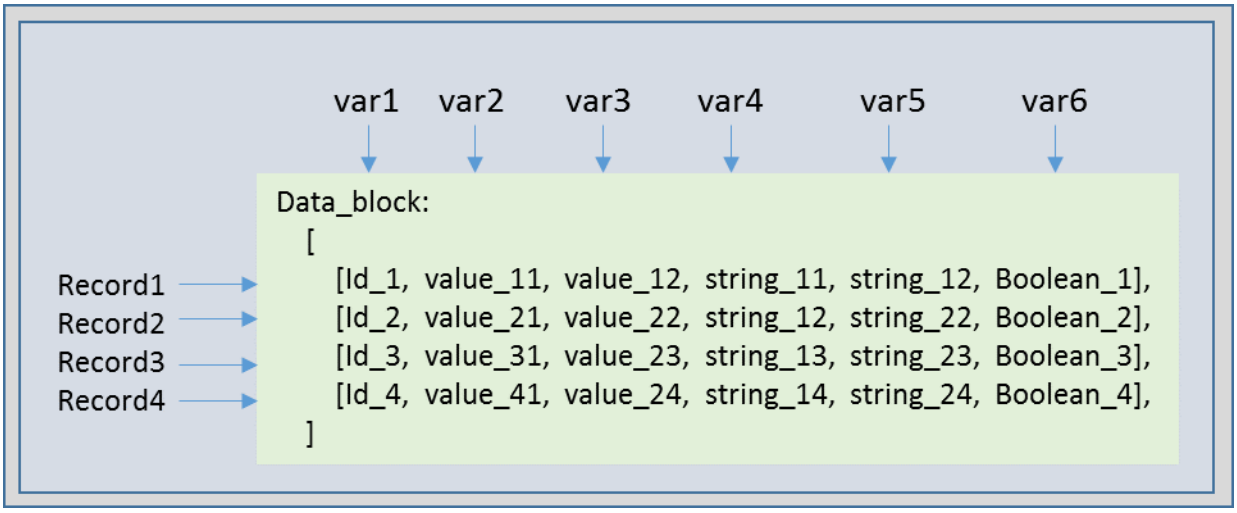

Figure 2. Standard data-block structure diagram for two-dimension data records, row for each record and column for each variable.

The syntax for standard data-block is basically related with YAML sequence methods for capturing the array and/or list. The data-block should be closed in a square bracket ([]) with two-space indentation. Each row represents a single data record with multiple variable values, which are separated by a comma and closed again with squared brackets ([]). Thus, for a 2-D data-block, it can be packed into a 2-D array/list with double squared brackets, like [[], [], …., []].

Commonly, the data-block may contain variables with different data types, including string, digital numbers, and Booleans. However, the mixed data type doesn't affect the YAML sequence syntax, other than simply transform it into a list after parsed in Python as shown in Figure 2.

3.2.2 GRACE legacy data-block

The current GRACE ASCII files contain a data section that is not compliant with the recommended YAML data block (array) format. Only the header sections in GRACE Level-1 and Level-2 ASCII data are upgraded into YAML format, but not the data-block due to a mission requirement for compatibility with GRACE legacy software. The GRACE legacy data-block contains a 2-D data list with each data element separated by single or multiple spaces without any indentation (Figure 3.2.2) but no spaces are allowed in each data element. So, GRACE ASCII files are not 100% YAML formatted files but can easily be converted as such (see 3.2.3).

3.2.3 Convert to YAML formatted data-block

The GRACE data-block could be made fully YAML-compatible by simply adding the data-block key and square bracket around the entire data section (Figure 3.2.3). When the YAML file is first loaded into Python, the data-block is not a 2-D list, but a 1-D list with only one string element, which is holding the entire data records. Extra-steps are needed to convert this single string to a 2-D list. A user could parse the string into a list of single elements by using space(s) as a separator and then can be reshaped into a 2-D list (using Python for example). The value at a given location can be extracted accordingly. Alternatively, indentation, square brackets, and commas between

values can be inserted into each line of the data block to simplify the interpretation of values in the code.

3.2.2 GRACE legacy data-block:

```
GRCOF2   0   0 -1.0068e-11  0.0000e+00 0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
GRCOF2   1   0 -2.6640e-11  0.0000e+00 0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
GRCOF2   1   1  3.5867e-11  7.1248e-11 0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
GRCOF2   2   0  6.0144e-11  0.0000e+00 0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
GRCOF2   2   1  7.6731e-12  3.949e-11   0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
GRCOF2   2   2 -3.702e-12   1.8947e-11  0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
```

3.2.3 Convert to YAML formatted data-block:

```
Data-block :
 [
GRCOF2   0   0 -1.0068e-11  0.0000e+00 0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
GRCOF2   1   0 -2.6640e-11  0.0000e+00 0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
GRCOF2   1   1  3.5867e-11  7.1248e-11 0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
GRCOF2   2   0  6.0144e-11  0.0000e+00 0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
GRCOF2   2   1  7.6731e-12  3.949e-11   0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
GRCOF2   2   2 -3.702e-12   1.8947e-11  0.0000e+00 0.0000e+00 20030201.0000 20030301.0000 nnnn
 ]
```

YAML requirements for the data-block are flexible, as it could convert most data-block forms into YAML compatible forms with minimal efforts, but additional efforts are needed to parse the YAML file in Python. We will not cover all the special cases.

4 Parse YAML file in Python

One of the big advantages using YAML format is that it can work with many programming languages, including C/C++, Ruby, Java, Python, and Perl. For a complete list, please see the website at http://yaml.org.

In this document, we present some examples of parsing the YAML file in Python for GRACE Level-2 ASCII products. The functionality of parsing YAML file in Python is relatively simple and straight forward. The variation of parsing the YAML file depends on how the YAML file is formatted, especially for the non-standard YAML data-block.

4.1 Opening/reading a YAML file

To “Open” and “Read” functions in the YAML file via Python takes only three steps as shown in the example below.

Example 4.1.

```
1 > import yaml
2 > stm = open("full_path/filename.yaml", 'r')
3 > data = yaml.load(stm)
```

The entire YAML file is loaded into the 'data' dictionary structure—the top layer, which has two major children: header and data-block.

4.2 Outputting a YAML file

Similar to Opening/Reading a YAML file commands, saving a YAML file in python can be implemented with the following two commands (Example 4.2.1).

Example 4.2.1.

```
1 > out = open("full_path/filename.yaml", 'w')
2 > yaml.dump(data, out, indent = 2, default_flow_style=False)
```

A YAML format can also be converted to JSON serializable format by executing the following commands in Python (Example 4.2.2).

Example 4.2.2.

```
1 > import sys
2 > import json
3 > with open('outfile.json', 'w') as out1:
        json.dump(data,out1, indent=2, separators=(',', ': '))
```

However, if the YAML structure includes the 'datatime' data type, the YAML to JSON conversion does not work, because the object of type 'datetime' is not JSON serializable. For example, in the GRACE ASCII product, the "unused_days" (Example 4.2.3) has value of 'datetime' list, which cannot be converted into JSON format.

Example 4.2.3.

```
unused_days   : [ 2003-02-08, 2003-02-20, 2003-02-21, 2003-02-26 ]
```

4.3 Extracting the metadata information

After a YAML file is open/read into a Python environment, the header as a child of "data" leads to a sub-dictionary structure, which contains all the metadata information. Users can then extract any metadata value for a given key. For example, if a user wants to know what the first 'unused_days' is, he/she can do the following (Example 4.3).

Example 4.3.

```
1   > undays = data['header']['global_attributes']['unused_days'][0]
2   > print ('Unused_days =', undays)
3   > Unused_days = 2003-02-08
```

Remember that the Python index starts with '0' not '1'.

4.4 Extracting the data record values

Depending on the formats of the data-block (shown in section 3.2), the methods to extract the data record value in Python will be different. With the YAML standard 2-D array/list data-block (Figure 2), the data value can be extracted with the following Python scripts (Example 4.4.1).

Example 4.4.1.

```
1   > data['standard_data'][0][3]      # value at 1st row and 4th col
```

For GRACE legacy data-block format (shown in Figure 3), more complicated Python scripts are needed to extract the values (Example 4.4.2).

Example 4.4.2.

```
1   > coef = data['data-block']      # assign the grace data to coef
2   > nList = coef[0].split()        # split the single string with space
3   > print(nList[3])                # print value at 1st row and 4th col
```

If a user desires to extract the values from a column, they may need to convert the 1-D list to 2-D list and then perform indexing on the first dimension. In practice, there are many more ways of doing the same work in Python, which are out of the scope for this document.

5 YAML encoding implications to Earth Science Data

5.1 Standardizing the YAML encoding for an ASCII file format

YAML encoding has few limitations for formatting Earth science observations. Its syntax is flexible enough to capture the variations of these datasets, including those of satellite mission products, airborne data records, and in-situ measurements. It has the advantage of presenting structured metadata and data records in a human-friendly interface, much more than other ASCII data formats. It is in full compliance with ESO approved "*ASCII File Format Guidelines for Earth Science Data*" [1], making itself a robust candidate as a standard ASCII data format representation for Earth Science Data.

5.2 Making and promoting FAIR data

Formatted with a YAML encoding, Earth observing mission ASCII products more readily meet the requirements for FAIR (Findable, Accessible, Interoperable, and Reusable) data status. The YAML format with the inclusion of metadata best practices from CF and ACDD enables an ASCII file to be more discoverable and understandable. This also allows the YAML file to be accessed and used by much broader communities, increasing the interoperability and reusability.

5.3 Feasible for developing generic access tools

It is always handy to have generic tools for quickly opening/reading and extracting needed information from a large dataset and to load the file into a user's preferred programing language for data analysis. For most ASCII files there is often simple but tedious programming preparation that must be implemented to deal with the ever-changing nuances of heterogeneous file structures from different providers and missions. YAML encoding is a candidate for standardized metadata and data structures in the ASCII data format, allowing software developers a more straightforward approach to build tools and services for these products.

6 References

Normative References

[1] Keith Evans, Aubrey Beach, Gao Chen, Peter Leonard, Emily Northup, Anne Wilson, 2016, ASCII File Format Guidelines for Earth Science Data, Technical Note, ESDS-RFC-027v1.1.

[2] B-K, Oren et al., 2009, YAML Ain't Markup Language (YAML™) Version-1.2, 3rd Edition, Patched at 2009-10-01, http://yaml.org/spec/1.2/spec.html.

Informative References

[3] E. Northup, G. Chen, K. Aikin, C. Webster, 2017, ICARTT File Format Standards V2.0, Technical Note, ESDS-RFC-029v2.

[4] P. Jeremy Werdell and Sean W. Bailey, 2002, The SeaWiFS Bio-Optical Archive and Storage System (SeaBASS): Current Architecture and Implementation, NASA/TM-2002-211617.

[5] H. Butler et al, IETF RFC7946 The GeoJSON Format, August 2016, https://tools.ietf.org/html/rfc7946.

[6] S. Bettadpur, 2018, Level-2 Gravity Field Product User Handbook, GRACE 327-734, Center for Space Research, The University of Texas at Austin

[7] S. Bettadpur, 2012, GRACE Product Specification Document, GRACE 327-720, Center for Space Research, The University of Texas at Austin

[8] PO.DAAC Metadata Best Practices: https://podaac.jpl.nasa.gov/PO.DAAC_DataManagementPractices#Metadata%20Conventions

[9] UDUNITS: https://www.unidata.ucar.edu/software/udunits/#home

7 Authors' Addresses

Wen-Hao Li, Raytheon IIS, 300 N Lake Ave Suite 1120, Pasadena, CA 91765, Tel: 626-744-5536, email: wen-hao.li@jpl.nasa.gov

Edward M. Armstrong, NASA Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Dr., Pasadena, CA 91109, Edward.M.Armstrong@jpl.nasa.gov

Christopher J. Finch, NASA Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Dr., Pasadena, CA 91109, Christopher.J.Finch@jpl.nasa.gov

David N. Wiese, NASA Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Dr., Pasadena, CA 91109, David.N.Wiese@jpl.nasa.gov

Appendix A

Glossary of acronyms

Acronym	*Description*
ACDD:	*Attribute Convention for Data Discovery*
ASCII:	*American Standard Code for Information Interchange*
CF:	*Climate and Forecast*
CDL:	*Common Data Form Language*
ESDIS:	*Earth Science Data and Information System*
ESDS:	*Earth Science Data Systems*
ESO:	*ESDIS Standard Office*
GRACE:	*Gravity Recovery and Climate Experiment*
GRACE-FO:	*Gravity Recovery and Climate Experiment Follow-On*
ISO:	*International Standard Office*
JSON:	*JavaScript Object Notation*
NASA:	*National Aeronautics and Space Administration*
NetCDF:	*Network Common Data Form*
PO.DAAC:	*Physical Oceanography Distributed Active Achieve Center*
XML:	*eXtensible Markup Language*
YAML:	*YAML Ain't Markup Language*

Appendix B, An example of JPL GRACE Level-2 Release-06 dataset (data-block only contains a partial data record)

```
header:
 dimensions:
  degree              : 180
  order               : 180

 non-standard_attributes:
  product_id          : GAA-2
  format_id           :
   short_name         : SHM
   long_name          : Earth Gravity Spherical Harmonic Model Format
  normalization       : fully normalized
  earth_gravity_param  :
   long_name          : gravitational constant times mass of Earth
   units              : m3/s2
   value              : 3.9860044150e+14
  mean_equator_radius  :
   long_name          : mean equator radius
   units              : meters
   value              : 6.3781363000e+06
  comments            : Degree 0 and Degree 1 terms are excluded from JPL Level 2 Processing
```

```
global_attributes:
  title            : AOD1B ATM Coefficients JPL RL06
  summary          : |
                   Spherical harmonic coefficients that represent anomalous contributions
                   of the non-tidal atmosphere to the Earth's mean gravity field during
                   the specified timespan.  This includes the contribution of atmospheric
                   surface pressure over the continents, the static contribution of
                   atmospheric pressure to ocean bottom pressure elsewhere, and the contribution
                   of upper-air density anomalies above both the continents and the oceans.
                   The anomalous signals are relative to the mean field from 2003-2014.
  project          : NASA Gravity Recovery And Climate Experiment (GRACE)
  program          : NASA Earth Science System Pathfinder
  keywords         : |
                   GRACE, Level-2, SHM, Spherical Harmonic Model, Gravitational Field, GSM,
                   Geopotential, Gravity Field, Mass, Mass Transport, Total Water Storage,
                   Time Variable Gravity, Mass Balance, Gravity Anomaly, Satellite Geodesy,
                   Atmosphere, Non-Tidal Atmosphere, AOD, Dealiasing Product
  keywords_vocabulary  : NASA Global Change Master Directory (GCMD) Science Keywords
  institution      : NASA/JPL
  conventions      : CF-1.6, ACDD-1.3, ISO 8601
  id               : 10.5067/GRGAA-20J06
  naming_authority : org.doi.dx
  history          : Original solution produced on 2018-05-19T08:41:19
  source           : All data from AOD1B RL06
  processing_level : 2
  acknowledgement  : |
                   GRACE is a joint mission of NASA (USA) and DLR (Germany).
                   Use the digital object identifier provided in the id attribute when
                   citing this data.  See https://podaac.jpl.nasa.gov/CitingPODAAC
  license          : https://science.nasa.gov/earth-science/earth-science-data/data-information-policy
  product_version  : 6.0
  references       : ftp://podaac-ftp.jpl.nasa.gov/allData/grace/docs/AOD1B_PDD_RL06_v6.0.pdf
  creator_name     : GRACE Science Data System NASA/JPL
  creator_email    : grace@podaac.jpl.nasa.gov
  creator_url      : https://grace.jpl.nasa.gov
  creator_type     : group
  creator_institution  : NASA/JPL
  publisher_name   : Physical Oceanography Distributed Active Archive Center
  publisher_email  : podaac@jpl.nasa.gov
  publisher_url    : https://podaac.jpl.nasa.gov
  publisher_type   : group
  publisher_institution : NASA/JPL
  time_coverage_start  : 2003-02-01T00:00:00.00
  time_coverage_end    : 2003-02-28T23:59:59.00
  unused_days      : [ 2003-02-08, 2003-02-20, 2003-02-21, 2003-02-26 ]
  date_created     : 2018-05-19T08:41:19
  date_issued      : 2018-05-19T08:41:19
```

```
# EGM Coefficient Record 2
 variables:
  record_key:
   key_name           : GRCOF2
   long_name          : Earth Gravity Spherical Harmonic Model Format Type
   coverage_content_type : referenceInformation
   data_type          : string
   comment            : 1st column
  degree_index:
   long_name          : spherical harmonic degree l
   coverage_content_type : referenceInformation
   data_type          : int32
   comment            : 2nd column
  order_index:
   long_name          : spherical harmonic order m
   coverage_content_type : referenceInformation
   data_type          : int32
   comment            : 3rd column
  clm:
   long_name          : Clm coefficient; cosine coefficient for degree l and order m
   coverage_content_type : modelResult
   data_type          : double precision
   comment            : 4th column
  slm:
   long_name          : Slm coefficient; sine coefficient for degree l and order m
   coverage_content_type : modelResult
   data_type          : double precision
   comment            : 5th column
  clm_std_dev:
   long_name          : standard deviation of Clm
   coverage_content_type : qualityInformation
   data_type          : double precision
   comment            : 6th column
  slm_std_dev:
   long_name          : standard deviation of Slm
   coverage_content_type : qualityInformation
   data_type          : double precision
   comment            : 7th column
  epoch_begin_time:
   long_name          : epoch begin of Clm, Slm coefficients
   time_format        : yyyymmdd.hhmm
   coverage_content_type : referenceInformation
   data_type          : string
   comment            : 8th column
  epoch_stop_time:
   long_name          : epoch stop of Clm, Slm coefficients
   time_format        : yyyymmdd.hhmm
   coverage_content_type : referenceInformation
   data_type          : string
   comment            : 9th column
  solution_flags:
   long_name          : Coefficient adjustment and a priori flags
   coverage_content_type : auxiliaryInformation
   data_type          : byte
   flag_meanings      :
    - char 1 = Clm adjusted, y for yes and n for no
    - char 2 = Slm adjusted, y for yes and n for no
    - char 3 = stochastic a priori info for Clm, y for yes and n for no
    - char 4 = stochastic a priori info for Slm, y for yes and n for no
   comment          : 10th column
  solution_comment:
   long_name          : Comment when present
   coverage_content_type : referenceInformation
   data_type          : string
   comment            : 11th column
# End of YAML header
```

```
datablock1:
 [
GRCOF2   0   0 -1.00683149816e-11 0.00000000000e+00 0.0000e+00 0.0000e+00 20030201.0000
20030301.0000 nnnn
GRCOF2   1   0 -2.66407102603e-11 0.00000000000e+00 0.0000e+00 0.0000e+00 20030201.0000
20030301.0000 nnnn
GRCOF2   1   1  3.58678636553e-11 7.12487373621e-11 0.0000e+00 0.0000e+00 20030201.0000
20030301.0000 nnnn
GRCOF2   2   0  6.01441591712e-11 0.00000000000e+00 0.0000e+00 0.0000e+00 20030201.0000
20030301.0000 nnnn
GRCOF2   2   1  7.67312701117e-12 3.94989658646e-11 0.0000e+00 0.0000e+00 20030201.0000
20030301.0000 nnnn
GRCOF2   2   2 -3.70293813238e-12 1.89477139706e-11 0.0000e+00 0.0000e+00 20030201.0000
20030301.0000 nnnn
GRCOF2   3   0 -7.79891175661e-11 0.00000000000e+00 0.0000e+00 0.0000e+00 20030201.0000
20030301.0000 nnnn
GRCOF2   3   1 -2.88827239507e-12 4.00099302962e-11 0.0000e+00 0.0000e+00 20030201.0000
20030301.0000 nnnn
GRCOF2   3   2  2.42976475640e-13 6.27492408251e-12 0.0000e+00 0.0000e+00 20030201.0000
20030301.0000 nnnn
 ]
```

Appendix C, PO.DAAC Metadata Best Practice Table.

Taken from :
https://podaac.jpl.nasa.gov/PO.DAAC_DataManagementPractices#Metadata%20Conventions

A. Global Attributes

Attribute Name	Type	Definitions	Priority	Source
title	string	A short phrase or sentence describing the dataset. In many discovery systems, the title will be displayed in the results list from a search, and therefore should be human readable and reasonable to display in a list of such names. This attribute is recommended by the NetCDF Users Guide (NUG) and the CF conventions.	required	ACDD 1.3, CF 1.7
summary	string	A paragraph describing the dataset, analogous to an abstract for a paper.	required	ACDD 1.3
keywords	string	A comma-separated list of key words and/or phrases. Keywords may be common words or phrases, terms from a controlled vocabulary (GCMD is often used), or URIs for terms from a controlled vocabulary (see also "keywords_vocabulary" attribute).	required	ACDD 1.3
keywords_vocabulary	string	If you are using a controlled vocabulary for the words/phrases in your "keywords" attribute, this is the unique name or identifier of the vocabulary from which keywords are taken. If more than one keyword vocabulary is used, each may be presented with a prefix (e.g., "CF:NetCDF COARDS Climate and Forecast Standard Names") and a following comma, so that keywords may optionally be prefixed with the controlled vocabulary key.	suggested	ACDD 1.3
Conventions	string	A comma-separated list of the conventions that are followed by the dataset. For files that follow this version of ACDD, include the string 'ACDD-1.3'. (This attribute is defined in NUG 1.7.)	required	ACDD 1.3, CF 1.7
id	string	An identifier for the data set, provided by and unique within its naming authority. The combination of the "naming authority" and the "id" should be globally unique, but the id can be globally unique by itself also. IDs can be URLs, URNs, DOIs, meaningful text strings, a local key, or any other unique string of characters. The id should not include white space characters.	recommended	ACDD 1.3

uuid	string	A uuid (Universal Unique Identifier) is a 128-bit number used to uniquely identify some object or entity on the Internet. Depending on the specific mechanisms used, a uuid is either guaranteed to be different or is, at least, extremely likely to be different from any other uuid generated until 3400 A.D.	required	PO.DAAC
naming_authority	string	The organization that provides the initial id (see above) for the dataset. The naming authority should be uniquely specified by this attribute. We recommend using reverse-DNS naming for the naming authority; URIs are also acceptable. Example: 'edu.ucar.unidata'.	recommended	ACDD 1.3
cdm_data_type	string	The data type, as derived from Unidata's Common Data Model Scientific Data types and understood by THREDDS. (This is a THREDDS "dataType", and is different from the CF NetCDF attribute 'featureType', which indicates a Discrete Sampling Geometry file in CF.)	suggested	ACDD 1.3
history	string	Provides an audit trail for modifications to the original data. This attribute is also in the NetCDF Users Guide: 'This is a character array with a line for each invocation of a program that has modified the dataset. Well-behaved generic netCDF applications should append a line containing: date, time of day, user name, program name and command arguments.' To include a more complete description you can append a reference to an ISO Lineage entity; see NOAA EDM ISO Lineage guidance.	recommended	ACDD 1.3, CF 1.7
source	string	The method of production of the original data. If it was model-generated, source should name the model and its version. If it is observational, source should characterize it. This attribute is defined in the CF Conventions. Examples: 'temperature from CTD #1234'; 'world model v.0.1'.	recommended	ACDD 1.3, CF 1.7
platform	string	Name of the platform(s) that supported the sensor data used to create this data set or product. Platforms can be of any type, including satellite, ship, station, aircraft or other. Indicate controlled vocabulary used in platform_vocabulary.	suggested	ACDD 1.3
platform_vocabulary	string	Controlled vocabulary for the names used in the "platform" attribute.	suggested	ACDD 1.3

instrument	string	Name of the contributing instrument(s) or sensor(s) used to create this data set or product. Indicate controlled vocabulary used in instrument_vocabulary.	suggested	ACDD 1.3
instrument_vocabulary	string	Controlled vocabulary for the names used in the "instrument" attribute.	suggested	ACDD 1.3
processing_level	string	A textual description of the processing (or quality control) level of the data.	recommended	ACDD 1.3
comment	string	Miscellaneous information about the data, not captured elsewhere. This attribute is defined in the CF Conventions.	recommended	ACDD 1.3, CF 1.7
standard_name_vocabulary	string	The name and version of the controlled vocabulary from which variable standard names are taken. (Values for any standard_name attribute must come from the CF Standard Names vocabulary for the data file or product to comply with CF.) Example: 'CF Standard Name Table v27'.	recommended	ACDD 1.3
acknowledgement	string	A place to acknowledge various types of support for the project that produced this data.	recommended	ACDD 1.3
license	string	Provide the URL to a standard or specific license, enter "Freely Distributed" or "None", or describe any restrictions to data access and distribution in free text.	recommended	ACDD 1.3
metadata_link	string	A URL that gives the location of more complete metadata. A persistent URL is recommended for this attribute.	suggested	ACDD 1.3
product_version	string	Version identifier of the data file or product as assigned by the data creator. For example, a new algorithm or methodology could result in a new product_version.	suggested	ACDD 1.3
references	string	Published or web-based references that describe the data or methods used to produce it. Recommend URIs (such as a URL or DOI) for papers or other references. This attribute is defined in the CF conventions.	suggested	ACDD 1.3, CF 1.7
creator_name	string	The name of the person (or other creator type specified by the creator_type attribute) principally responsible for creating this data.	recommended	ACDD 1.3
creator_email	string	The email address of the person (or other creator type specified by the creator_type attribute) principally responsible for creating this data.	recommended	ACDD 1.3
creator_url	string	The URL of the of the person (or other creator type specified by the creator_type attribute) principally responsible for creating this data.	recommended	ACDD 1.3

creator_type	string	Specifies type of creator with one of the following: 'person', 'group', 'institution', or 'position'. If this attribute is not specified, the creator is assumed to be a person.	suggested	ACDD 1.3
creator_institution	string	The institution of the creator; should uniquely identify the creator's institution. This attribute's value should be specified even if it matches the value of publisher_institution, or if creator_type is institution.	suggested	ACDD 1.3
institution	string	The name of the institution principally responsible for originating this data. This attribute is recommended by the CF convention.	recommended	ACDD 1.3, CF 1.7
project	string	The name of the project(s) principally responsible for originating this data. Multiple projects can be separated by commas, as described under Attribute Content Guidelines. Examples: 'PATMOS-X', 'Extended Continental Shelf Project'.	recommended	ACDD 1.3
program	string	The overarching program(s) of which the dataset is a part. A program consists of a set (or portfolio) of related and possibly interdependent projects that meet an overarching objective. Examples: 'GHRSST', 'NOAA CDR', 'NASA EOS', 'JPSS', 'GOES-R'.	suggested	ACDD 1.3
contributor_name	string	The name of any individuals, projects, or institutions that contributed to the creation of this data. May be presented as free text, or in a structured format compatible with conversion to ncML (e.g., insensitive to changes in whitespace, including end-of-line characters).	suggested	ACDD 1.3
contributor_role	string	The role of any individuals, projects, or institutions that contributed to the creation of this data. May be presented as free text, or in a structured format compatible with conversion to ncML (e.g., insensitive to changes in whitespace, including end-of-line characters). Multiple roles should be presented in the same order and number as the names in contributor_names.	suggested	ACDD 1.3
publisher_name	string	The name of the person (or other entity specified by the publisher_type attribute) responsible for publishing the data file or product to users, with its current metadata and format.	recommended	ACDD 1.3

publisher_email	string	The email address of the person (or other entity specified by the publisher_type attribute) responsible for publishing the data file or product to users, with its current metadata and format.	recommended	ACDD 1.3
publisher_url	string	The URL of the person (or other entity specified by the publisher_type attribute) responsible for publishing the data file or product to users, with its current metadata and format.	recommended	ACDD 1.3
publisher_type	string	Specifies type of publisher with one of the following: 'person', 'group', 'institution', or 'position'. If this attribute is not specified, the publisher is assumed to be a person.	suggested	ACDD 1.3
publisher_institution	string	The institution that presented the data file or equivalent product to users; should uniquely identify the institution. If publisher_type is institution, this should have the same value as publisher_name.	suggested	ACDD 1.3
geospatial_bounds	float	Describes the data's 2D or 3D geospatial extent in OGC's Well-Known Text (WKT) Geometry format (reference the OGC Simple Feature Access (SFA) specification). The meaning and order of values for each point's coordinates depends on the coordinate reference system (CRS). The ACDD default is 2D geometry in the EPSG:4326 coordinate reference system. The default may be overridden with geospatial_bounds_crs and geospatial_bounds_vertical_crs (see those attributes). EPSG:4326 coordinate values are latitude (decimal degrees_north) and longitude (decimal degrees_east), in that order. Longitude values in the default case are limited to the (-180, 180) range. Example: "POLYGON ((40.26 -111.29, 41.26 -111.29, 41.26 -110.29, 40.26 -110.29, 40.26 -111.29))".	recommended	ACDD 1.3
geospatial_bounds_crs	string	The coordinate reference system (CRS) of the point coordinates in the geospatial_bounds attribute. This CRS may be 2-dimensional or 3-dimensional, but together with geospatial_bounds_vertical_crs, if that attribute is supplied, must match the dimensionality, order, and meaning of point coordinate values in the geospatial_bounds attribute. If geospatial_bounds_vertical_crs is also present then this attribute must only specify a 2D CRS. EPSG CRSs are strongly recommended. If this attribute is not specified,	recommended	ACDD 1.3

		the CRS is assumed to be EPSG:4326. Examples: "EPSG:4979" (the 3D WGS84 CRS), "EPSG:4047".		
geospatial_bounds_vertical_crs	string	The vertical coordinate reference system (CRS) for the Z axis of the point coordinates in the geospatial_bounds attribute. This attribute cannot be used if the CRS in geospatial_bounds_crs is 3-dimensional; to use this attribute, geospatial_bounds_crs must exist and specify a 2D CRS. EPSG CRSs are strongly recommended. There is no default for this attribute when not specified. Examples: "EPSG:5829" (instantaneous height above sea level), "EPSG:5831" (instantaneous depth below sea level), or "EPSG:5703" (NAVD88 height).	recommended	ACDD 1.3
geospatial_lat_min	float	Describes a simple lower latitude limit; may be part of a 2- or 3-dimensional bounding region. Geospatial_lat_min specifies the southernmost latitude covered by the dataset.	recommended	ACDD 1.3
geospatial_lat_max	float	Describes a simple upper latitude limit; may be part of a 2- or 3-dimensional bounding region. Geospatial_lat_max specifies the northernmost latitude covered by the dataset.	recommended	ACDD 1.3
geospatial_lat_units	string	Units for the latitude axis described in "geospatial_lat_min" and "geospatial_lat_max" attributes. These are presumed to be "degree_north"; other options from udunits may be specified instead.	suggested	ACDD 1.3
geospatial_lat_resolution	float	Information about the targeted spacing of points in latitude. Recommend describing resolution as a number value followed by the units. Examples: '100 meters', '0.1 degree'. For level 1 and 2 swath data this is an approximation of the pixel resolution.	suggested	ACDD 1.3
geospatial_lon_min	float	Describes a simple longitude limit; may be part of a 2- or 3-dimensional bounding region. geospatial_lon_min specifies the westernmost longitude covered by the dataset. See also geospatial_lon_max.	recommended	ACDD 1.3

geospatial_lon_max	float	Describes a simple longitude limit; may be part of a 2- or 3-dimensional bounding region. geospatial_lon_max specifies the easternmost longitude covered by the dataset. Cases where geospatial_lon_min is greater than geospatial_lon_max indicate the bounding box extends from geospatial_lon_max, through the longitude range discontinuity meridian (either the antimeridian for -180:180 values, or Prime Meridian for 0:360 values), to geospatial_lon_min; for example, geospatial_lon_min=170 and geospatial_lon_max=-175 incorporates 15 degrees of longitude (ranges 170 to 180 and -180 to -175).	Recommended	ACDD 1.3
geospatial_lon_units	string	Units for the longitude axis described in "geospatial_lon_min" and "geospatial_lon_max" attributes. These are presumed to be "degree_east"; other options from udunits may be specified instead.	suggested	ACDD 1.3
geospatial_lon_resolution	float	Information about the targeted spacing of points in longitude. Recommend describing resolution as a number value followed by units. Examples: '100 meters', '0.1 degree'. For level 1 and 2 swath data this is an approximation of the pixel resolution.	suggested	ACDD 1.3
geospatial_vertical_min	float	Describes the numerically smaller vertical limit; may be part of a 2- or 3-dimensional bounding region. See geospatial_vertical_positive and geospatial_vertical_units.	recommended	ACDD 1.3
geospatial_vertical_max	float	Describes the numerically larger vertical limit; may be part of a 2- or 3-dimensional bounding region. See geospatial_vertical_positive and geospatial_vertical_units.	recommended	ACDD 1.3
geospatial_vertical_resolution	float	Information about the targeted vertical spacing of points. Example: '25 meters'	suggested	ACDD 1.3
geospatial_vertical_units	string	Units for the vertical axis described in "geospatial_vertical_min" and "geospatial_vertical_max" attributes. The default is EPSG:4979 (height above the ellipsoid, in meters); other vertical coordinate reference systems may be specified. Note that the common oceanographic practice of using pressure for a vertical coordinate, while not strictly a depth, can be specified using the unit bar. Examples: 'EPSG:5829' (instantaneous	suggested	ACDD 1.3

		height above sea level), 'EPSG:5831' (instantaneous depth below sea level).		
geospatial_vertical_positive	string	One of 'up' or 'down'. If up, vertical values are interpreted as 'altitude', with negative values corresponding to below the reference datum (e.g., under water). If down, vertical values are interpreted as 'depth', positive values correspond to below the reference datum. Note that if geospatial_vertical_positive is down ('depth' orientation), the geospatial_vertical_min attribute specifies the data's vertical location furthest from the earth's center, and the geospatial_vertical_max attribute specifies the location closest to the earth's center.	suggested	ACDD 1.3
time_coverage_start	string	Describes the time of the first data point in the data set. Use the ISO 8601:2004 date format, preferably the extended format as recommended in the Attributes Content Guidance section.	recommended	ACDD 1.3
time_coverage_end	string	Describes the time of the last data point in the data set. Use ISO 8601:2004 date format, preferably the extended format as recommended in the Attributes Content Guidance section.	recommended	ACDD 1.3
time_coverage_duration	string	Describes the duration of the data set. Use ISO 8601:2004 duration format, preferably the extended format as recommended in the Attributes Content Guidance section.	recommended	ACDD 1.3
time_coverage_resolution	string	Describes the targeted time period between each value in the data set. Use ISO 8601:2004 duration format, preferably the extended format as recommended in the Attributes Content Guidance section.	recommended	ACDD 1.3
date_created	string	The date on which this version of the data was created. (Modification of values implies a new version, hence this would be assigned the date of the most recent values modification.) Metadata changes are not considered when assigning the date_created. The ISO 8601:2004 extended date format is recommended, as	recommended	ACDD 1.3

		described in the Attribute Content Guidance section.		
date_modified	string	The date on which the data was last modified. Note that this applies just to the data, not the metadata. The ISO 8601:2004 extended date format is recommended, as described in the Attributes Content Guidance section.	suggested	ACDD 1.3
date_issued	string	The date on which this data (including all modifications) was formally issued (i.e., made available to a wider audience). Note that these apply just to the data, not the metadata. The ISO 8601:2004 extended date format is recommended, as described in the Attributes Content Guidance section.	suggested	ACDD 1.3
date_metadata_modified	string	The date on which the metadata was last modified. The ISO 8601:2004 extended date format is recommended, as described in the Attributes Content Guidance section.	suggested	ACDD 1.3

B. Variable Attributes

Attribute Name	Type	Definitions	Priority	Source
long_name	string	A long descriptive name for the variable (not necessarily from a controlled vocabulary). This attribute is recommended by the NetCDF Users Guide, the COARDS convention, and the CF convention.	required	ACDD 1.3, CF 1.7
standard_name	string	A long descriptive name for the variable taken from a controlled vocabulary of variable names. We recommend using the CF convention and the variable names from the CF standard name table (http://cfconventions.org/Data/cf-standard-names/36/build/cf-standard-name-table.html). This attribute is recommended by the CF convention.	required	ACDD 1.3, CF 1.7
units	string	The units of the variable's data values. This attribute value should be a valid udunits string. The "units" attribute is recommended by the NetCDF Users Guide, the COARDS convention, and the CF convention (http://www.unidata.ucar.edu/software/udunits/udunits-1/udunits.txt).	required	ACDD 1.3, CF 1.7

coverage_content_type	string	An ISO 19115-1 code to indicate the source of the data --MD_CoverageContentTypeCode (https://geo-ide.noaa.gov/wiki/index.php?title=ISO_19115_and_19115-2_CodeList_Dictionaries#CI_PresentationFormCode). For example, image, thematicClassification, physicalMeasurement, auxiliaryInformation, qualityInformation, referenceInformation, modelResult, or coordinate.	required	ACDD 1.3
valid_range	variable type	Comma seperated minimum and maximum values of the physical quantity defining the valid measuremnt range.	required	CF 1.7
coordinates	string	This attribute contains a space seperated list of all the coordinates corresponding to the variable.The list should contain all the arxiliary coordinate variables and optionally the coordinate variables.	required	CF 1.7
scale_factor	variable type	Slope of scaling relationship applied to transform measuement data to appropriate geophysical quantity representations. Should not be used if the scale_factor is '1' and add_offset is '0'	required	CF 1.7
add_offset	variable type	Intercept of scaling relationship applied to transform measuement data to appropriate geophysical quantity representations. Should not be used if the scale_factor is '1' and add_offset is '0'	required	CF 1.7
_FillValue	variable type	Assigned value in the data file desiganting a null or missing observation	required	CF 1.7
grid_mapping	string	Describes the horizontal coordinate system used by the data. The grid_mapping attribute should point to a variable which would contain the parameters corresponding to the coordinate system. There are typically several parameters associated with each coordinate system. CF defines a separate attributes for each of the parameters. Some examples are "semi_major_axis", "inverse_flattening", "false_easting"	recommended	CF 1.7
comment	string	Optional attribute field allowing provision of further free-form information about the variable	recommended	CF 1.7
flag_masks	variable type	A number of independent Boolean (binary) conditions using bit field notation and setting unique bits whose values are associated with a list of descriptive phrases in attribute *flag_meanings*. The *flag_masks* attribute is the same type as the variable to which it is attached, and contains a list of values matching unique bit	recommended	CF 1.7

		fields. (See CF document section 3.5; http://cfconventions.org/Data/cf-conventions/cf-conventions-1.7/cf-conventions.html#flags) In this example variableY has 6 unique *flag_masks* bit flag states which are defined by the strings in *flag_meanings* variableY:flag_masks = 1b, 2b, 4b, 8b, 16b, 32b ; variableY:flag_meanings = "land_contamination open_ocean coastal_ocean lake river ice_contamination " ;		
flag_meanings	string	A list of strings that defines the the physical meaning of each *flag_masks* bit field or *flag_values* scaler field with a single text string. The strings are often phrases with words catenated with underscores, and strings are separated by a single space. CF allows a single variable to contain both flag_values and flag_masks. The interpretation of the flags in such cases is slightly tricky. In such cases flag_masks is used to "group" a set of flag_values into a nested conditional. Please see the example 3.5 in the CF document on how to interpret flag_meanings in such cases. It is recommended that Boolean (*flag_masks*) and enumerated flags (*flag_values*) be kept in separate variables.	recommended	CF 1.7
flag_values	variable type	*flag_values* consists of an enumerated list of status flags indicating unique conditions whose meaning is described by the commensurate list of descriptive phrases in attribute *flag_meanings.* The status flags are scaler of the same type as the variable. In this example, data points in *variableX* have one of 3 possible status flags having values of 0, 1, or 2 variableX:flag_values = 0b, 1b, 2b ; variableX:flag_meanings = "good_quality suspect_data sensor_nonfunctional”	recommended	CF 1.7

C. Georeferencing Coordinate Variable Attributes.

Attribute Name	Type	Definitions	Priority	Source
long_name	string	custom/long descriptive name of variable	required	ACDD 1.3, CF 1.7
standard_name	string	standard variable name used to describe the georefencing variable (eg. latitude, longitude, height)	required	ACDD 1.3, CF 1.7
units	string	standard unit name for the standard georeferencing variable (eg. "degrees_north", "degrees_east", "m"	required	ACDD 1.3, CF 1.7
axis	string	Corresponding variable axis for plotting (eg. X, Y, Z)	required	CF 1.7
_FillValue	variable type	Assigned value in the data file desiganting a null or missing observation. NASA best practices specifies that for satellite datasets there should not be a _FillVlalue for these geolocation variables.	required	CF 1.7
valid_min	variable type	The minimum values of georeferancing variables (eg. latitude, longitude, height)	required	CF 1.7
valid_max	variable type	The maximum values of georeferancing variables (eg. latitude, longitude, height)	required	CF 1.7
comment	string	Optional attribute field allowing provision of further free-form information about the variable	required	CF 1.7

D. Temporal Coordinate Variable Attributes.

Attribute Name	Type	Definitions	Priority	Source
long_name	string	custom/long descriptive name of variable	required	ACDD 1.3, CF 1.7
standard_name	string	standard variable name used to describe the temporal variable (ie. time)	required	ACDD 1.3, CF 1.7
units	string	standard unit descriptor (eg. days, hours, seconds etc) cited against a standard reference date ("since".. date/time in ISO 8601 format)	required	ACDD 1.3, CF 1.7
axis	string	Corresponding variable axis for plotting (eg. T)	required	CF 1.7
_FillValue	variable type	Assigned value in the data file desiganting a null or missing observation. NASA best practices specifies that for satellite datasets there should not be a _FillVlalue for time variables.	required	CF 1.7
comment	string	Optional attribute field allowing provision of further free-form information about the variable	required	CF 1.7